Kernel-based Learning in the Absence of Counterexamples: One-class Classification

Learning to Identify the Unknown

Ph.D. Thesis

By

Chandan Gautam

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2019

Kernel-based Learning in the Absence of Counterexamples: One-class Classification

Learning to Identify the Unknown

A THESIS

submitted to the

INDIAN INSTITUTE OF TECHNOLOGY INDORE

in partial fulfillment of the requirements for

the award of the degree

of

DOCTOR OF PHILOSOPHY

By

Chandan Gautam

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

NOVEMBER 2019

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **Kernel-based Learning in the Absence of Counterexamples: One-class Classification** in the partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted in the **Discipline of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the time period from January 2015 to October 2019 under the supervision of Dr. Aruna Tiwari, Associate Professor, Indian Institute of Technology Indore, Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Signature of the Student with Date

(Chandan Gautam)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Signature of Thesis Supervisor with Date

(Dr. Aruna Tiwari)

Chandan Gautam has successfully given his Ph.D. Oral Examination held on

_____

Signature of Chairperson, OEB Date:	Signature of External Examiner Date	Signature of Thesis Supervisor Date:
Signature of PSPC Member #1 Date:	Signature of PSPC Member #2 Date	Signature of Convener, DPGC Date:
Signature of Head of Discipline Date:		

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable, enjoyable, and bearable. At first, I would like to thank my supervisor **Dr. Aruna Tiwari**, who was a constant source of inspiration during my work. Without her constant guidance and research directions, this research work could not be completed. Her continuous support and encouragement has motivated me to remain streamlined in my research work.

I am thankful to **Dr. Kapil Ahuja** and **Dr. Srivathsan Vasudevan**, my research progress committee members for taking out some valuable time to evaluate my progress all these years. Their valuable comments and suggestions helped me to improve my work at various stages. I am also grateful to HOD of Computer Science for his help and support.

I am especially grateful to Dr. Suresh Sundaram, Dr. M. Tanveer, Dr. Alexadrous Iosifidis, and Mr. Qian Leng, for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives. I am also thankful to Dr. V. Ravi who introduced me in the field of research by his proper guidelines during my masters.

My sincere acknowledgement and respect to **Prof. Pradeep Mathur**, Director, Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities at Indian Institute of Technology Indore.

I would like to appreciate the fine company of my dearest colleagues and friends especially, Piyush Joshi, Sadaf Ali, Navneet Pratap Singh, Mayank Modak, Rajendra Choudhary, Nikhil Tripathi, Mayank Swarnkar, Ram Prakash Sharma, Aditya Shastri, Aaditya Prakash Chouhan, and Vikas Chauhan. I am also thankful to undergraduate students who have also supported me in my research work.

I am also grateful to the institute staffs for their unfailing support and assistance, and the Ministry of Electronics and Information Technology for funding the PhD research.

I would like to express my heartfelt respect to my parents for their love, care and support they have provided to me throughout my life. Special thanks to my sister (Shweta), brothers (Atulaya, Amit, and Shubham), fiancée (Anjila) and friends (Yogesh, Ravi, Ashutosh, and Ravikant) as this thesis would not have been possible without the help of their support and encouragements.

Finally, I am thankful to all who directly or indirectly contributed, helped and supported me. To sign off, I write a quote by Albert Einstein:

"Everything should be made as simple as possible, but no simpler."

Chandan Gautam

To my family and friends

Abstract

This thesis mainly investigates the kernel learning-based approach for outlier (novelty or anomaly or negative) detection using one-class classification (OCC). OCC is a non-traditional way of classification where the model is built using samples from only one class, and samples from this class belong to normal (positive or target) class. The one-class classifier classifies any unknown class other than the normal class as an outlier class. All proposed one-class classifiers in this thesis are developed based on the boundary and reconstruction frameworks.

In recent years, kernel ridge regression (KRR) (or least squares support vector machine with zero bias or kernel extreme learning machine) based one-class classifiers have received quite an attention by researchers. Researcher developed a KRR-based one-class classifier for boundary framework. We have developed it for the reconstruction framework. For further performance improvement, we have combined the concept of both the frameworks in a single multi-layer architecture. This architecture is formed by sequential stacking of various KRR-based Auto-Encoders, followed by a KRR-based one-class classifier. The stacked architecture provides a better representation of the data using representation learning, which helps in obtaining better classification compared to single hidden layer-based architecture. Further, this multi-layer architecture is extended to use structural information between samples using a Graph-Embedding approach. The structural information is generated by different types of Laplacian graphs and embedded into the existing multi-layer architecture. Later, we have explored multiple kernel learning (MKL) for one-class classification, which captures different notions of the data using different types of kernels. Existing MKL-based one-class classifier assigns equal weights to each kernel over the whole input space. In our work, we have developed localized multiple kernel learning based one-class classifiers, which assign weights to each kernel based on locality present in the data. These weights are assigned with the help of a gating function in the optimization problem. In order to handle the privileged information during learning, we have extended the two KRR-based one-class classifiers (boundary and reconstruction

framework-based classifiers) for utilizing privileged information using learning using privileged information (LUPI) framework. In last, we have also enabled these two KRR-based one-class classifiers (boundary and reconstruction framework-based classifiers) to handle the non-stationary data streams efficiently.

Overall, this thesis contributes by developing various kernel-based learning methods for various types of learnings viz; representation learning, multi-layer learning, multiple kernel learning, LUPI framework-based learning, and online learning. All these developed methods are evaluated exhaustively to compare with the various state-of-the-art OCC methods in terms of various performance evaluation criteria.

Keywords: Kernel Ridge Regression, One-class Classification, Kernel Learning, Auto-Encoder, Representation Learning, Multi-layer, Graph-Embedding, Multi-kernel Learning, Support Vector Machine, LUPI Framework, Online Learning

List of Publications

A. Published

A1. In Refereed Journals

1. **C. Gautam**, A. Tiwari, M. Tanveer. *KOC+: Kernel Ridge Regression based One-class Classification using Privileged Information*, Information Sciences, vol. 504, pp. 324-333, 2019 (Elsevier), DOI = "https://doi.org/10.1016/j.ins.2019.07.052". (**IF: 5.524**)

2. **C. Gautam**, A. Tiwari, S. Suresh, and K. Ahuja. *Adaptive Online Learning with Regularized Kernel for One-class Classification*, IEEE Transactions on Systems, Man, and Cybernetics: Systems, pp. 1-16, 2019 (IEEE), DOI = "https://doi.org/10.1109/TSMC.2019.2907672". (**IF: 7.351**)

3. **C. Gautam**, R. Balajia, K. Sudharsan, A. Tiwari, and K. Ahuja. *Localized Multiple Kernel Learning for Anomaly Detection: One-class Classification*, Knowledge-Based Systems, vol. 165, pp. 241-252, 2018 (Elsevier), DOI = "https://doi.org/10.1016/j.knosys.2018.11.030". (**IF: 5.101**)

4. **C. Gautam**, A. Tiwari, and Q. Leng. *On The Construction of Extreme Learning Machine for Online and Offline One Class Classifier - An Expanded Toolbox*, Neurocomputing, vol. 261, pp. 126-143, 2017 (Elsevier), DOI = "https://doi.org/10.1016/j.neucom.2016.04.070". (**IF: 4.072**)

A2. In Refereed Conferences

1. **C. Gautam**, A. Tiwari. *Localized Multiple Kernel Support Vector Data Description*, International Conference on Data Mining Workshops (IEEE ICDMW-2018), Singapore, pp. 1514-1521, November, 2018, DOI = "https://doi.org/10.1109/ICDMW.2018.00224".

2. **C. Gautam**, A. Tiwari, and A. Iosifidis. *Minimum Variance-Embedded Multi-layer Kernel Ridge Regression for One-class Classification*, IEEE Symposium Series on Computational Intelligence (IEEE SSCI-2018), Bengaluru, India, pp. 389-396, November, 2018, DOI = "https://doi.org/10.1109/SSCI.2018.8628692". (Flagship annual international conference sponsored by the IEEE Computational Intelligence Society)

3. **C. Gautam**, A. Tiwari, S. Ravindran. *Construction of Multi-class Classifiers by Extreme Learning Machine Based One-Class Classifiers*, International Joint Conference on Neural Networks (IEEE IJCNN-2016), Vancouver, Canada, July, 2016, DOI = "https://doi.org/10.1109/IJCNN.2016.7727445". (Flagship conference of the IEEE Computational Intelligence Society and the International Neural Network Society)

4. **C. Gautam**, A. Tiwari. *On The Construction of Extreme Learning Machine for One Class Classifier*, International Conference on Extreme Learning Machines (ELM-2015), Hangzhou, China, vol. 6, pp. 447-461, December, 2015, DOI = "https://doi.org/10.1007/978-3-319-28397-5_35". (**Received travel grant by Department of Science and Technology, Govt. of India**)

B. Communicated

In Refereed Journals

1. **C. Gautam**, A. Tiwari, M. Tanveer. *AEKOC+: Kernel Ridge Regression-based Auto-Encoder for One-class Classification using Privileged Information*, Cognitive Computation (Springer). (Submitted after the 2^{nd} revision in October 2019) (**IF: 4.287**)

2. **C. Gautam**, A. Tiwari, S. Suresh, A. Iosifidis, M. Tanveer. *Graph-Embedded Multi-layer Kernel Ridge Regression for One-class Classification*, Cognitive Computation (Springer). (Communicated in November 2018) (**IF: 4.287**)

3. **C. Gautam**, A. Tiwari, S. Suresh, and A. Iosifidis. *Multi-layer Kernel Ridge Regression for One-class Classification*, Expert Systems With Applications (Elsevier). (Submitted after 3^{rd} revision in November 2019) (**IF: 4.292**)

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

AEKOC Kernel Ridge Regression-based Auto-Encoder for One-class Classification

AEKOC+ Learning Using Privileged Information framework with AEKOC

CDA Clustering-based Linear Discriminant Analysis

ELM Extreme Learning Machine

GKOC Graph-Embedding with KRR-based One-class Classifier

GMKOC Graph-Embedded Multi-layer Kernel Ridge Regression for One-class Classification

incSVDD incremental Suport Vector Data Description

KAE Kernel Ridge Regression-based Auto-Encoder

KELM Kernel Extreme Learning Machine

KOC Kernel Ridge Regression-based One-class Classifier

KOC+ Learning Using Privileged Information framework with KOC

KPCA Kernel Principal Component Analysis

KRR Kernel Ridge Regression

LDA Linear Discriminant Analysis

LE Laplacian Eigenmap

LFDA Local Fisher Discriminant analysis

LLE Locally Linear Embedding

LMKAD Localized Multiple Kernel Learning for Anomaly Detection

LMKL Localized Multiple Kernel Learning

LMSVDD Localized Multiple Kernel Support Vector Data Description

LSSVM Least Squares Suport Vector Machine

LUPI Learning Using Privileged Information

MKAD Multiple Kernel Anomaly Detection

MKL Multiple Kernel Learning

MKOC Multi-layer Kernel Ridge Regression for One-class Classification

OCC One-class Classification

OCSVM One-class Suport Vector Machine

OCSVM+ Learning Using Privileged Information framework with One-class Suport Vector Machine

OS-AEKOC Online sequential learning for AEKOC

OS-KOC Online sequential learning for KOC

PCA Principal Component Analysis

RBF Radial Basis Function

SVDD Suport Vector Data Description

SVDD+ Learning Using Privileged Information framework with Support Vector Data Description

SVM Suport Vector Machine

Chapter 1

Introduction

Over the years, advances in machine learning have attracted the research community, particularly for classification. This involves a variety of problems viz., binary class classification, multi-class classification, and one-class classification. In the case of binary class classification, a sample would be classified in either of the existing two classes irrespective of the fact whether it belongs to those two classes or not. There is a situation when samples of only one class are available to build a model, and this model has to identify the unknown (or outlier) sample, which does not belong to this class. In this thesis, we are focusing on handling this situation only. It can be resolved by using the one-class classification (OCC). Following examples provide a better understanding of OCC:

(i) **Fraud detection [16]**: Financial fraud detection is one of the best-suited examples for OCC. Since the nature of fraud keeps changing, it is challenging to characterize all possible nature of fraud. In contrast, user behavior does not change; therefore, we can construct a model based on user behavior. This constructed model treats all behavior other than user behavior as anomalous behavior. There is one more possibility here that how classifier works if user behavior changes? For this purpose, online learning [15] (learning on the fly) is introduced with OCC, which updates the model as per requirement.

(ii) **Face verification [17, 18]**: When you have to verify a face, outlier class can not be defined because all possible faces on the earth except target face belong to this

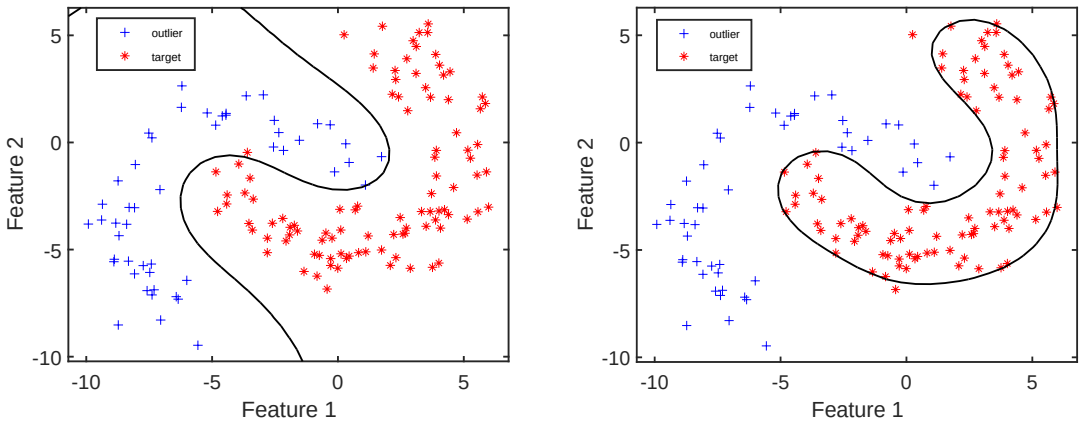
class. Therefore, OCC is the most suitable method for verification purpose [17, 18] as there is no need to define outlier class.

(iii) **Signature verification [19]**: Similar to face verification, there is only need to train the model using the signature of specific person for verification.

Similarly, OCC has been broadly applied in various domains, like information retrieval [2], recommender system [20, 21], remote-sensing [22], biometric fusion [23], machine fault detection [24], bioinformatics [25], and disease detection [26].

1.1 Background

The OCC problem is entirely different from the traditional (binary or multi-class) classification problem. The assumption in OCC is that data from only one class is available for building the model. Here, a one-class classifier constructs a discrimination boundary from the information of only one class. This class is called a target class (or normal or genuine or positive class), and remaining samples, which do not belong to this class, are termed as the outlier (or anomalous or negative). In contrast to the one-class classifier, the binary class classifier constructs the decision boundary by the



(a) Binary Classification (b) OCC

Figure 1.1: Decision boundary constructed by a support vector machine-based binary and one-class classifier

support of samples from both classes, i.e., target and outlier class. Decision boundary construction of binary and one-class classification is visualized in Figure 1.1. Generally, OCC comes into the picture when a sufficient number of samples are available from the target class but not from another class. This unavailability of data is due to various reasons [6], such as the very high measurement costs, the less occurrence of an event, failure of a nuclear power plant, a rare medical disease, and machine fault detection. Moreover, an insufficient amount of samples can not characterize the complete nature of another class, and this might lead to a poor classification model. Overall, a one-class classifier learns from the samples of the target class and minimizes the possibility of accepting the outlier sample.

In literature [27, 28], OCC has been primarily applied for novelty or outlier detection. Moya et al. [29] coined the word 'one-class classification' and employed a one-class classifier for target recognition application. Further, Japkowicz [30] proposed an auto-association-based approach for OCC, which is a neural network-based approach and termed as "concept learning in the absence of counterexamples". In the same year, kernel-based one-class classifiers are proposed [1, 31]. Kernel-based one-class classifiers can be broadly divided into two categories [6]: (i) reconstruction-based (ii) boundary-based. Reconstruction-based methods reconstruct the input space at the output layer and provide more compact representation of the target data. These method perform OCC based on the reconstruction error at the output layer. Hoffmann [4] proposed reconstruction-based one-class classifier by considering kernel principal component analysis (KPCA) as a base method. Boundary-based methods construct classifier's boundary based on the the structure of the dataset. Two popular SVM-based one-classifiers are developed for boundary-based, viz., one-class support vector machine (OCSVM) [1] and support vector data description (SVDD) [31, 6, 3]. Researchers employed both methods (i.e., OCSVM and SVDD) in various discipline for solving different types of problem, like document classification [2], fMRI analysis [32], seizure detection [33], novelty detection [34], and Fabric defect detection [35]. As SVM follows the iterative approach of learning, therefore, SVM-based one-class classifiers are computationally expensive. Choi [36] and Leng et al. [7] addressed this

issue by proposing a least squares-based one-class classifier. Choi [36] and Leng et al. [7] selected least squares SVM and kernel extreme learning machine as a base classifier, respectively. Least squares-based one-class classifier is further enabled to embed structural information within its optimization problem [8, 17]. Structural information provides more representation power to the method.

Further, kernel learning-based OCC has been explored in a few more interesting directions, like multiple kernel learning [11], learning using privileged information (LUPI) framework-based learning [37, 38, 39] and online learning [40, 41, 42]. Multiple kernel learning (MKL) selects the best performing kernel among a predefined set of kernels. These kernels can be generated either by using different types of kernels or same kernel on different sources of data. Das et al. [11] introduced multiple kernel learning-based one-class classifier for anomaly detection in aviation data. In recent years, LUPI framework got quite a popularity due to its non-conventional way of learning. It introduced human teaching into traditional machine learning. In recent years, various researchers [37, 38, 39] developed it for OCC. Another quite interesting learning approach is online learning, which is the requirement of real-time learning in these days. In these days, data is generating continuously, and their characteristics also change as time passes. These data can be either stationary or non-stationary. Above mentioned one-class classifiers can efficiently handle stationary data, but unable to do so for non-stationary data. Researchers addressed this issue by developing online learning-based one-class classifier, like online SVDD [40], online OCSVM [40, 41, 42], and online kernel principal component analysis-based one-class classifier [4].

1.2 Motivation

Researchers have shown that kernel learning-based one-class classifiers have performed very well for various application domains [3, 27, 28]. However, these kernel learning-based one-class classifiers are computationally expensive due to the iterative approach of learning. Hence, there is a need to develop a non-iterative model for OCC. In recent years, few researchers have developed the non-iterative learning-based model

for multi-class classification [43, 44, 45]. They combined representation learning with kernel learning for this purpose. Representation learning is being quite popular in the field of machine learning [46] due to its better data representation capability to perform classification more precisely. By taking a cue from these points, we also endeavor to explore the kernel-based representation learning for OCC. Thus, we propose single hidden layer and multi-layer one-class classifiers based on the representation learning by considering KRR as a base classifier. To provide more expressive power to the proposed multi-layer architecture, we explore structural information of data with this multi-layer architecture using a Graph-Embedding approach [47]. Further, to capture different notions of the data using different types of kernel, we explore multiple kernel learning (MKL) [48] for OCC. Das et al. [11] obtained good performance by combining multiple kernels in a single objective function with equal weight to each kernel. However, each kernel doesn't need to be equally important. We address this issue by developing two multiple kernel learning methods, which assigns the weight to each kernel, based on the underlying localities in the data. At the end, we enable KRR-based existing [7] and developed one-class classifiers to handle two types of information: (a) privileged information [13], (b) non-stationary streaming data [15, 49]. We explore LUPI framework-based learning [13] for handling privileged information, and online learning for handling non-stationary streaming data [15, 49]. Overall, this thesis endeavors to address various issues of the existing kernel-based methods for solving OCC problems.

1.3 Objectives

In this thesis, we aim to achieve the following objectives:

(i) To develop a reconstruction framework-based one-class classifier using kernel learning.

(ii) To develop a method, which can combine the concept of boundary and reconstruction in a single architecture. This architecture should also be capable of embedding structural information within it.

(iii) To develop boundary framework-based methods, which can explore the different localities present in the data by using various types of kernels.

(iv) To enable boundary and reconstruction framework-based methods for handling privileged information efficiently for OCC.

(v) To enable boundary and reconstruction framework-based methods for handling non-stationary streaming data efficiently for OCC.

1.4 Thesis Contributions

Figure 1.2 depicts the connection between the research contribution of this thesis. A brief overview of our research contributions is provided below, and more details are available in the later chapters.

Contribution I:

The performance of machine learning algorithms heavily depends upon the representation of the data [46]. Therefore, a lot of effort has been made in this field and explored for various types of machine learning tasks. Recently, it has been explored for kernel learning-based binary and multi-class classification [44, 45]. In this thesis, we explore representation learning by developing of single hidden-layered vanilla kernel ridge regression (KRR) based Auto-Encoder for OCC. Proposed OCC method is primarily based on reconstruction error. This method is less computationally expensive compared to traditional kernel-based OCC methods (like OCSVM and SVDD) because it follows the non-iterative approach of learning.

Contribution II:

We further explore representation learning for the development of a multi-layer architecture. This architecture is formed by stacking various single hidden-layered KRR-based Auto-Encoders sequentially, followed by a KRR-based one-class classifier at the last layer. These stacked Auto-Encoders provide a better representation of data so that a one-class classifier can classify the data more precisely. Further, the optimization problem is extended to use structural information between samples in its formulation. This information is generated by different types of Laplacian graphs and embedded into the existing multi-layer architecture.

Contribution III:

In order to capture different notions in the data, multiple kernel learning (MKL) is required. Unlike the multi-layer-based method (as discussed in Contribution II), the MKL-based method optimizes multiple kernels simultaneously in a single optimization function. Existing MKL-based one-class classifier does not explore the locality present in the data and also assign equal weight to each kernel. For addressing these issues, we introduce the concept of localization with multiple kernel learning for OCC. Localization assigns different weight to each kernel and optimizes those weights by a two steps alternate optimization scheme [50, 12]. We develop two localized MKL-based one-class classifiers for anomaly detection by taking OCSVM and SVDD as a base classifier.

Contribution IV:

Existing KRR-based one-class classifiers are unable to handle privileged information. This information is generally available for training; however, it is not available for testing. In order to handle this issue, we incorporate the concept of learning using privileged information (LUPI) with two types of KRR-based single hidden-layered one-class classifiers. One of the proposed classifier is boundary framework-based and another is reconstruction framework-based classifier.

Contribution V:

The research contributions discussed till now have assumed that data is stationary in nature and the whole data is available for training before it starts. However, in the real-time scenario, data is available in the form of continuous streams. These continuous streams can be either stationary or non-stationary. To handle this situation, we enhance boundary and reconstruction framework-based one-class classifiers for online sequential learning, which can handle non-stationary and streaming data efficiently. We test the proposed methods for various types of drift in a controlled environment.

1.5 Organization of the Thesis

This thesis is organized into nine chapters. A summary of each chapter is provided below:

Chapter 1 (Introduction)

This chapter describes background knowledge of OCC, the motivation of our work, and the contribution of this thesis.

Chapter 2 (Literature Survey and Research Methodology)

This chapter provides a detailed literature survey and a summary of various state-of-the-art kernel-based methods. It also provides evaluation metrics for performance analysis.

Chapter 3 (Kernel Ridge Regression-based Auto-Encoder for One-class Classification)

In this chapter, a reconstruction framework-based one-class classifier is presented for OCC using KRR.

Chapter 4 (Multi-layer Kernel Ridge Regression for One-class Classification)

In this chapter, we present a multi-layer architecture for OCC. For developing a multi-layer architecture, Auto-Encoder presented in Chapter 3 is stacked sequentially one after another, followed by a one-class classifier.

Figure 1.2: Overall work-flow of this thesis

Chapter 5 (Graph-Embedded Multi-layer Kernel Ridge Regression for One-class Classification)

The method proposed in Chapter 4 does not consider structural information between samples. For addressing this issue, this chapter includes the proposal of using the Graph-Embedding approach, which explores structural information with multi-layer architecture.

Chapter 6 (Localized Multiple Kernel Learning for One-class Classification)

The presented methods in Chapters 4 and 5 employ the kernel-based method at multiple layers sequentially. Instead of a sequential combination of kernels, we optimize multiple kernels simultaneously in a minimization problem, and the same is presented in this chapter.

Chapter 7 (Learning Using Privileged Information with Kernel Ridge Regression for One-class Classification)

None of the methods discussed in the previous chapters are capable of utilizing privileged information with their optimization problem. This chapter presents the extension of two KRR-based one-class classifiers for utilizing privileged information with their optimization models.

Chapter 8 (Adaptive Online Sequential Learning with Kernel Ridge Regression for One-class Classification)

Methods presented in the previous chapters can only handle stationary data. This chapter presents two online learning methods for handling non-stationary streaming data.

Chapter 9 (Conclusions and Future Work)

This chapter briefly describes the contribution of this thesis and the possible future directions of our work.

Chapter 2

Literature Survey and Research Methodology

This chapter provides a detailed literature review for kernel learning-based OCC and identifies the gap in the literature. We have divided the whole literature into six sections. Section 2.1 provides preliminaries on OCC and kernel trick before proceeding to the literature survey in further sections. Section 2.2 discusses various kernel learning-based one-class classifiers and their applications, Section 2.3 provides a brief literature on Auto-Encoder, and Section 2.4 discusses about Graph-Embedding and Graph-Embedding based one-class classifiers. Section 2.5 discusses multiple kernel learning-based one-class classifiers. Further, Section 2.6 describes kernel learning using privileged information for OCC, and the last section (Section 2.7) discusses online learning-based one-class classifiers.

2.1 Preliminaries

This section provides preliminaries on OCC and kernel trick, which helps to understand further sections.

2.1.1 Learning in the Absence of Counterexamples: One-class Classification

As far as learning for OCC is concerned, it is broadly categorized in two ways [51, 6, 28]; (i) the data applied during learning, (ii) types of frameworks used to build the model. For the first one, Khan and Madden [51] further categorized OCC into three parts as follows:

(i) with positive samples only
(ii) with positive and small amount of negative samples only
(iii) with positive and unlabeled data

For the second one, Tax [6] further categorized OCC into three parts as follows:

(i) **Density framework-based approach:** It is a straightforward approach to build a one-class classifier based on the density estimation of the training data. It assumes that there is a low probability of the occurrence of target data in the low-density area of the training set [6, 28]. Overall, It computes the density based on some underlying distribution and set a threshold based on the estimated density. If any sample lies outside of this threshold, then treat it as an outlier sample; otherwise, the target sample.

(ii) **Reconstruction framework-based approach:** In this approach, the weight of the trained model is adjusted according to target samples. This model has learned to reconstruct itself on the output. As the model is trained for target samples only, therefore, the reconstruction error will be less for target samples, but when we pass any anomalous sample to the trained model, then it is obvious that the reconstruction error will be more. Hence, we set some threshold, and if that error is less than the threshold, then treat it as a target sample otherwise as an outlier sample [6, 28].

(iii) **Boundary framework-based approach:** This approach needs only to determine the boundary based on the structure of the dataset. Boundary-based

methods do not rely on any specific distribution or density of the dataset because they describe the boundary or domain of the target class, not the distribution or density [6, 28]. If any sample lies outside of this boundary, then treat it as an outlier sample; otherwise, the target sample.

2.1.2 Kernel Trick

Kernel trick is the core of kernel learning-based methods. It is simply based on the inner product of samples into some new feature space $\boldsymbol{\phi(x)}$. It is defined as follows [52]:

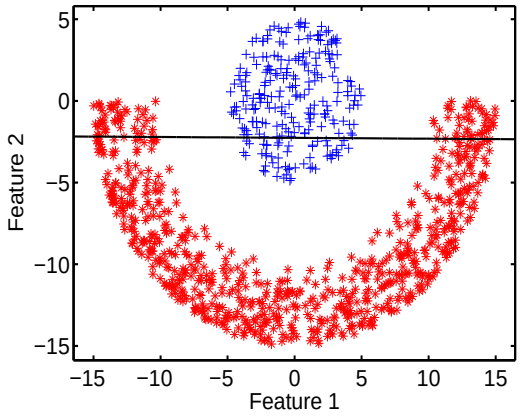
Definition: We say that $\boldsymbol{K(x, y)}$ is a kernel function iff there is a feature map such that for all $\boldsymbol{x}$ and $\boldsymbol{y}$,

$$\boldsymbol{K(x, y) = \phi(x).\phi(y)}$$

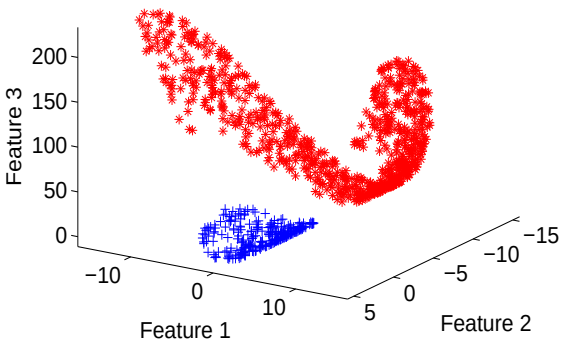
Here, a kernel function generates a matrix, which is called kernel or gram matrix. This matrix needs to be symmetric and positive semi-definite. Any function can be treated as kernel function iff it satisfies Mercer's condition [52]. Mercer's Theorem can be defined as follows:

Mercer's Theorem: A symmetric function $\boldsymbol{K(x, y)}$ can be expressed as an inner product $\boldsymbol{K(x, y) = \phi(x).\phi(y)}$ for some $\boldsymbol{\Phi}$ if and only if $\boldsymbol{K(x, y)}$ is positive semidefinite.

We need to understand, what kernel trick basically does? It is not always possible to separate the two classes from each other in the lower dimensional space using a hyperplane. Hence, data is projected into the higher dimensional space so that a hyperplane can easily separate it. This can be visualized in Figure 2.1. Figure 2.1(a) represents data in low dimension space, and it can be seen that data is not linearly separable. After adding one more dimension, i.e., Feature 3= (Feature 1)^2, data can be linearly separated by a hyperplane as shown in Figure 2.1(b).



(a) Lower Dimension



(b) Data is projected into higher dimensions by creating a new dimension by taking the square of Feature 1 as the third dimension

Figure 2.1: Visualization of data in lower and higher dimensions

2.2 Kernel Learning

Over the last decades, researchers have used kernel learning for different types of classification tasks [53], such as binary, multi-class, and one-class classification. In this thesis, we are focusing on the one-class classification problem. In 1999, Schölkopf et al. [1] developed a kernel-based one-class classifier for novelty detection, and it is popularly known as a one-class support vector machine (OCSVM). This method is developed by taking the support vector machine (SVM) as a base classifier. Further, in the same year, Tax and Duin developed another method for OCC task by again taking SVM as a base classifier and popularly known as support vector domain description [31] or support vector data description [3] (SVDD). The working methodology of both the methods, OCSVM and SVDD, are different from each other. OCSVM constructs a hyperplane, which separates all the target class data points from the origin in feature space. It also maximizes the distance of this hyperplane from the origin. Instead of a hyperplane approach like OCSVM, Tax and Duin [3] proposed a hypersphere-based approach. They developed SVDD by finding a hypersphere, in feature space, of minimum radius around the target class data such that it encloses almost all points of the target class dataset. A diagram is depicted in Figure 2.2, which shows the

difference between OCSVM and SVDD.

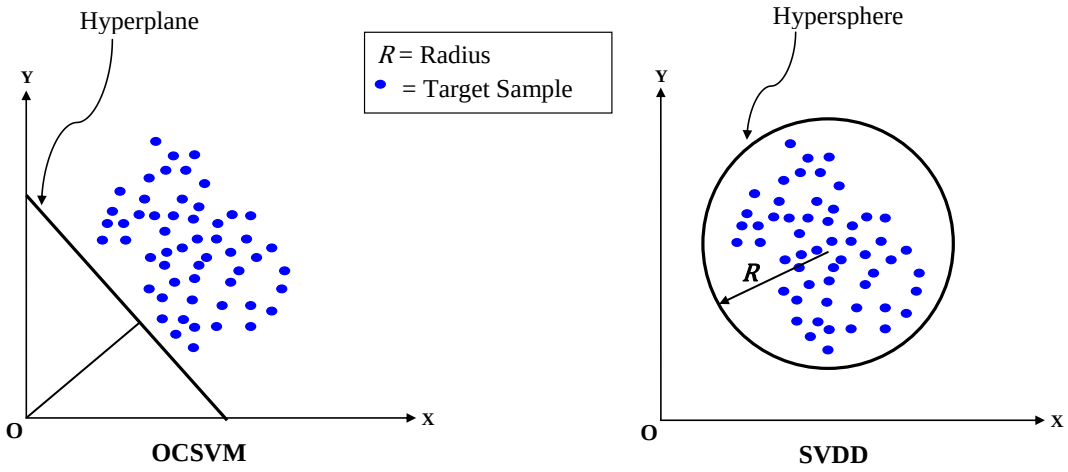


Figure 2.2: Hyperplane of OCSVM and hypersphere of SVDD

Tax and Duin [6] widely studied SVDD and various other one-class classifiers. They developed a toolbox for various one-class classifiers and named as data description toolbox [54]. Data descriptor is another name of one-class classifier because it describes the characteristics of the data and performs classification based on that. Both OCSVM and SVDD are domain-based one-class classifiers. Since both the classifiers hardly use any information from the statistics of the target data, they are insensitive to any specific sampling or density of the target class. These methods describe the boundary or structure or domain of the target class data points. They can be effective in the case where the density distribution of the target class is not known. However, at the same time, they are susceptible to the outliers in the training set. Hence, the appropriate amount of samples from the target class must be available for training to describe the domain of the target class.

There are several contributions made for OCSVM and SVDD in terms of theory and application [28, 51]. In theory domain, Perkins et al. [55] extended OCSVM and enabled it for handling temporal sequence. Yang et al. [56] and He et al. [57] developed OCSVM for multi-task learning. Further, Xue and Beauseroy [58] enabled OCSVM for multi-task learning with additional new features among similar tasks. Khan et al. [59] developed a covariance-guided OCSVM. Perdisci et al. [60] constructed ensembles of

OCSVM classifiers for payload-based anomaly detection systems. Hao [61] introduced the fuzzy concept with OCSVM for novelty detection. In the application domain, researchers have applied OCSVM for document classification [2], network intrusion detection[62, 63] and seizures detection in patients [33]. Further, It is also applied for novelty detection for audio data [64], text data [65], functional magnetic resonance imaging data [32], medical data [65], multi-channel combustion data [66, 67], jet engine health monitoring [68], etc. Similar to the above, SVDD is also extended theoretically and employed in various applications. Density-based SVDD is developed by Lee et al. [69, 70]. Wu and Ye [34] proposed small sphere and large margin approach-based SVDD for novelty detection. Further, it is extended by Le et al. [71] by selecting optimal sphere for two large margins approach-based SVDD. Xiao et al. [72] developed multi-sphere support vector data description for outlier detection on multi-distribution data. In [73] and [74], a fast and efficient SVDD is proposed to speed up the training time of SVDD. SVDD is applied in various disciplines viz., classification of remote sensing images [75], fabric defect detection [35], high-speed inline defect detection for TFT-LCD array process [76], batch process monitoring [77], chillers fault detection [78], novelty detection approach in machinery components [79] and outlier detection with noise or uncertain data [80].

As discussed above, SVM-based one-class classifiers are widely applied in many domains due to their performance. However, they are computationally expensive because of their involvement in solving a quadratic optimization problem. This issue is handled by introducing least-squares to SVM formulation and named as least squares SVM (LSSVM) [81, 82, 36]. The idea of this least squares is directly linked to the kernel ridge regression (KRR)[83]. Suykens et al. [81] introduced a bias term with KRR and reformulated for SVM. Hence, LSSVM with zero bias is identical to KRR. This can be seen from the formulation of LSSVM in (2.1), and KRR in (2.3) and (2.4). There is one more popular method, extreme learning machine (ELM) [84], which has reformulated its optimization problem for kernel [85] and leads to the identical optimization problem as KRR [86]. Kernel formulation of ELM is referred to as kernel extreme learning machine (KELM). We are providing a brief analysis on the

optimization problem of LSSVM, KELM, and KRR in the following section so that we can understand the difference between these methods.

2.2.1 Analysis of LSSVM, KELM, and KRR

Before mentioning the optimization problem, let us do some assumption regarding variables. For a given training set $\{\boldsymbol{x_i}, t_i\}_{i=1,2,...,N}$, where $\boldsymbol{x_i}$ denotes i^{th} training sample and y_i denotes target output of i^{th} sample, optimization problems of LSSVM, KELM, and KRR are as follows.

Optimization problem of LSSVM:

$$\begin{aligned} \underset{\boldsymbol{\omega}, e_i}{\text{Minimize}} : \pounds_{LSSVM} &= \frac{1}{2}\|\boldsymbol{\omega}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ \text{Subject to} : \ & \boldsymbol{\omega^T \phi_i} + b = y_i - e_i, \ i = 1, 2, ..., N, \end{aligned} \tag{2.1}$$

where $\boldsymbol{\omega}$ is the weight coefficients (Just to differentiate from KRR/KELM, we use the different notations of weight coefficient for SVM-based methods in throughout the thesis.), $\phi(.)$ is the mapping in the feature space, e_i denotes training error of i^{th} sample, and C is a regularization parameter.

Optimization problem of KELM:

$$\begin{aligned} \underset{\boldsymbol{\beta}, e_i}{\text{Minimize}} : \pounds_{KELM} &= \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ \text{Subject to} : \ & \boldsymbol{\beta^T h(x_i)} = y_i - e_i, \ i = 1, 2, ..., N, \end{aligned} \tag{2.2}$$

where $\boldsymbol{\beta}$ is the output weight (i.e., weight coefficients) and $h(.)$ is the mapping in KELM feature space. We have used same notation of weight coefficient and feature mapping as used in most of the KELM papers [85, 86].

Optimization problem of KRR can be represented in two ways as follows:

First way,

$$\begin{aligned} &\underset{\boldsymbol{\beta},e_i}{\text{Minimize}} : \pounds_{KRR-I} = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ &\text{Subject to} : \ \boldsymbol{\beta^T \phi_i} = y_i - e_i, \ i = 1, 2, ..., N. \end{aligned} \tag{2.3}$$

Second way,

$$\begin{aligned} &\underset{\boldsymbol{\beta},e_i}{\text{Minimize}} : \pounds_{KRR-II} = \frac{C}{2}\|\boldsymbol{\beta}\|^2 + \frac{1}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ &\text{Subject to} : \ \boldsymbol{\beta^T \phi_i} = y_i - e_i, \ i = 1, 2, ..., N. \end{aligned} \tag{2.4}$$

Differences among the optimization problems of LSSVM, KELM, and KRR:

(i) Difference between two formulations of KRR ((2.3) vs. (2.4)): In (2.3), C is associated with the error term. In (2.4), C is associated with weight term. Here, the parameter C controls a trade-off between low square loss and low norm of the solution [87]. Hence, you can associate C with either weight or with the error term. It does not change the solution.

(ii) KELM vs. KRR ((2.2) vs. (2.3)): KELM uses the notation of $h(x_i)$ for non-linear feature mapping, and KRR uses the notation ϕ_i for the same. That is the only difference between them. Optimization problem will not change just due to the use of different notation for the same thing (i.e., $h(x_i)$ or ϕ_i or any new variable for feature mapping). So, both KELM and KRR yield the same solution. Hence, we conclude that both KELM and KRR are identical in every aspect.

(iii) LSSVM vs. KRR ((2.1) vs. (2.3)): Both optimization problems are just differed by a term 'b', which is added on the left side of the constraints of LSSVM in (2.1). Here, we can obtain the formulation of KRR by simply substituting $b = 0$ in (2.1). Hence, it can be concluded that LSSVM with zero bias is equivalent to KRR.

Based on the above discussion, we reach the following conclusion:

KELM = KRR = LSSVM with zero bias

The proposed methods of this thesis are the variants of KRR, LSSVM(with bias=0), and KELM. Since KRR is older and more generic name compared to LSSVM and KELM, we use name KRR instead of LSSVM or KELM in this thesis for the existing and proposed methods. We have renamed the KELM-based method to KRR-based for keeping the uniform naming convention throughout the thesis. We are providing a brief survey of KRR-based work in the following paragraph.

KRR-based model [83] optimizes the problem rapidly in a non-iterative way by solving a linear system. Therefore, the KRR-based model has received quite attention by researchers in both the dimensions i.e. theory and application. In theory domain, various types of methods have been developed, such as transductive regression [88], multi-task regression [89], distributed KRR [90], randomized KRR [91], co-Trained KRR [92], memory efficient KRR [93], KRR using sketching and preconditioning [94], clustering using KRR [93], and random Fourier features for KRR [95]). It is also employed for various types of applications, such as face recognition [96], face hallucination [97], fMRI pattern prediction [98], wind speed prediction [99] and speech recognition [100]. As discussed above, KRR is broadly applied in various types of tasks viz; multi-class classification, prediction, and clustering. In recent years, it is of immense interest for researchers to apply KRR for OCC, which is discussed in the following section.

2.2.2 KRR-based One-class Classifiers

The KRR-based one-class classifiers can be divided into two categories, namely, (i) without Graph-Embedding (ii) with Graph-Embedding.

(i) **Without Graph-Embedding**: For this, KRR-based single output node architecture is proposed by Leng et al. [7] for OCC and referred to as KOC in this thesis. Leng et al. [7] have shown that it performs well compared to various

state-of-the-art methods. Further, KOC is employed for gas turbine combustor anomaly detection [9] and video anomaly detection [101].

Limitation: KOC does not consider structural information of the data in its optimization problem, which is further discussed in Section 2.2.5.

Remedy: It can be addressed by employing the Graph-Embedding approach with the optimization problem.

(ii) **With Graph-Embedding**: By addressing the limitation of KOC, Iosifidis et al. [8] presented local and global variance-based Graph-Embedded one-class classifier (Graph-Embedding is discussed briefly in Section 2.4). Iosifidis et al. [8] employed two types of Laplacian graphs for OCC. Two types of Graph-Embedding are local (i.e., Locally Linear Embedding and Laplacian Eigenmaps) and global (i.e., linear discriminant analysis and clustering-based discriminant analysis) variance embedding. Mygdalis et al. [102] employed Laplacian graph-based one-class classifier for human action recognition. Later, global variance-based Graph-Embedding has been extended to exploit class variance and sub-class variance information for face verification task by Mygdalis et al. [17].

Limitation: Although, Graph-Embedding methods improved the representational power of KOC by adding structural information, however, its representation learning ability is still limited due to single-layered architecture.

Remedy: Representation learning [46, 43, 44, 45] can be introduced by converting single-layer to multi-layer architecture by stacking Auto-Encoders (Auto-Encoder is briefly discussed in Section 2.3) before classification layer.

Since the proposed methods of this thesis are based on either OCSVM, SVDD, or KOC, we provide a brief discussion of the formulations of these methods in the subsequent Sections 2.2.3, 2.2.4, 2.2.5. We discuss Graph-Embedding-based methods in Section 2.4.2 after the discussion about the basics of Graph-Embedding.

2.2.3 One-class SVM: OCSVM

Scholkopf et al. [1] proposed one-class SVM (OCSVM) for extending the utility offered by SVM to One-class classification. Given a set of training vectors $\boldsymbol{x_i} \in \mathbb{R}^n, i = 1, \ldots, N$, where all training vectors belong to the same class and termed as target class data points. OCSVM constructs a hyperplane that separates all the target class data points from the origin and maximizes the distance of this hyperplane from the origin. It is formulated in the following optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{\omega},\boldsymbol{\xi},\rho}{\text{Minimize}} : \frac{1}{2}\boldsymbol{\omega}^T\boldsymbol{\omega} - \rho + \frac{1}{\nu N}\sum_{i=1}^{N}\xi_i \\ &\text{Subject to} : \ \boldsymbol{\omega}^T\boldsymbol{\phi}(\boldsymbol{x_i}) \geq \rho - \xi_i \quad i = 1, \ldots, N \\ &\text{Subject to} : \ \xi_i \geq 0, \quad i = 1, \ldots, N. \end{aligned} \tag{2.5}$$

The dual of which can be written as

$$\begin{aligned} &\underset{\alpha}{\text{Minimize}} : \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha} \\ &\text{Subject to} : \ 0 \leq \alpha_i \leq \tfrac{1}{\nu N} \quad i = 1, \ldots, N \\ &\text{Subject to} : \ \sum_{i=1}^{N}\alpha_i = 1, \end{aligned} \tag{2.6}$$

where $k_{ij} = \mathbb{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{\phi}(\boldsymbol{x_i})^T\boldsymbol{\phi}(\boldsymbol{x_j})$

In the above two equations, $\boldsymbol{\omega}$ is the weight coefficients, $\phi(.)$ is the mapping in the feature space, $\mathbb{K}$ is a kernel function, $\boldsymbol{K}$ is the kernel matrix where $k_{ij} \in \boldsymbol{K}$ generates an element of $\boldsymbol{K}$ between i^{th} and j^{th} sample, $\boldsymbol{\alpha}$ is the Lagrange multiplier, N is the total number of training samples provided, ν is a parameter that lets the user define the fraction of target class points rejected, ρ is the bias term, and $\boldsymbol{\xi} = \{\xi_i\}$, where $i = 1, 2, \ldots, N$, is the error with respect to the i^{th} sample. The above minimization problem results in a binary function, which returns $+1$ or -1 for target class and outliers, respectively, and is called the decision function. The decision function $\boldsymbol{f(x)}$ thus obtained is as follows:

$$f(\boldsymbol{x}) = sign(\sum_{i=1}^{N} \alpha_i \mathbb{K}(\boldsymbol{x_i}, \boldsymbol{x}) - \rho). \tag{2.7}$$

Based on the formulation OCSVM, Das et al. [11] proposed anomaly detection for more than one kernel, which is described in the next section.

2.2.4 Support Vector Data Description: SVDD

SVDD was proposed by Tax et al. [3] to make SVM compatible for One-class classification task. We are providing an overview of SVDD and discusses its primal, dual, and decision function formulation in this section. For a given set of training samples $\boldsymbol{x_i} \in \mathbb{R}^n, i = 1, \ldots, N$, where all these samples belong to the same class (also known as target class). SVDD constructs a spherical shape boundary that constructs the compact sphere with no superfluous space using only target class samples. This can be written as the following optimization problem [3]:

$$\begin{aligned} &\underset{R,\boldsymbol{a},\boldsymbol{\xi}}{\text{Minimize}} : R^2 + C\sum_{i=1}^{N} \xi_i \\ &\text{Subject to :} \ \|\boldsymbol{\phi}(\boldsymbol{x_i}) - \boldsymbol{a}\|^2 \leq R^2 + \xi_i \quad i = 1, \ldots, N \\ &\text{Subject to :} \ \xi_i \geq 0, \quad i = 1, \ldots, N. \end{aligned} \tag{2.8}$$

The dual of which can be written as

$$\begin{aligned} &\underset{\alpha}{\text{Maximize}} : \sum_{i=1}^{N} \alpha_i k_{ii} - \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \\ &\text{Subject to :} \ 0 \leq \alpha_i \leq C \quad i = 1, \ldots, N \\ &\text{Subject to :} \ \sum_{i=1}^{N} \alpha_i = 1, \end{aligned} \tag{2.9}$$

where $k_{ij} = \mathbb{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \boldsymbol{\phi}(\boldsymbol{x_i})^T \boldsymbol{\phi}(\boldsymbol{x_j})$

In the above two equations, $\phi(.)$ is a function, which is mapping data in the higher dimensional feature space, $\boldsymbol{K}$ is the kernel matrix where $k_{ij} \in \boldsymbol{K}$ generates an element

of $\boldsymbol{K}$ between i^{th} and j^{th} sample, $\boldsymbol{\alpha}$ is the Lagrange multiplier, N is the total number of training samples provided, $\boldsymbol{a}$ is a center, and R is a radius of the hypersphere. Any testing instance can be detected as a target or outlier based on a decision function $\boldsymbol{f(x)}$. This function results in a binary function, which returns either $+1$ or -1. The decision function $\boldsymbol{f(x)}$ can be defined as follows:

$$\begin{aligned} f(\boldsymbol{x}) &= sign(\|\boldsymbol{\phi(x)} - \boldsymbol{a}\|^2 - R^2) \\ &= \begin{cases} -1, \ \boldsymbol{x} \text{ is classified as target} \\ \ \ 1, \ \boldsymbol{x} \text{ is classified as outlier.} \end{cases} \end{aligned} \tag{2.10}$$

Here, R^2 is the distance between the center of sphere and any of the support vectors on the boundary.

2.2.5 KRR-based One-class Classification Using Single Output Node Architecture: KOC

As per the discussion in the previous section, Leng et al. [7] proposed a single output node-based architecture for OCC. A schematic diagram of KOC is shown in Figure 2.3. For KOC, let us assume the input training matrix of size $N \times n$ is $\boldsymbol{X} = \boldsymbol{\{x_i\}}$, where $\boldsymbol{x_i} = [x_{i1}, x_{i2}, ..., x_{in}]$, $i = 1, 2, ..., N$, is the n-dimensional input vector of the i^{th} training sample. The minimization function of KOC is as follows:

$$\begin{aligned} &\underset{\boldsymbol{\beta}, e_i}{\text{Minimize}} : \pounds_{KOC} = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ &\text{Subject to} : \ \boldsymbol{(\beta)^T \phi_i} = r - e_i, \ i = 1, 2, ..., N, \end{aligned} \tag{2.11}$$

where C is a regularization parameter, $\boldsymbol{\beta}$ denotes weight vector, and $\boldsymbol{r}$ is a vector having all elements equal to r. Here, r is any real number. Leng et al. [7] set r at equal to 1. $\phi(.)$ denotes kernel feature mapping function, $\boldsymbol{\phi_i = \phi(x_i)}$, and $\boldsymbol{\Phi = \Phi(X) = [\phi_1, \phi_2, ..., \phi_N]}$. $\boldsymbol{E}$ is an error vector where $\boldsymbol{E} = \{e_i\}$ and $i = 1, 2, ..., N$. Here e_i is a training error corresponding to i^{th} training sample. By using Representer Theorem [103], $\boldsymbol{\beta}$ is expressed as a linear combination of the training data

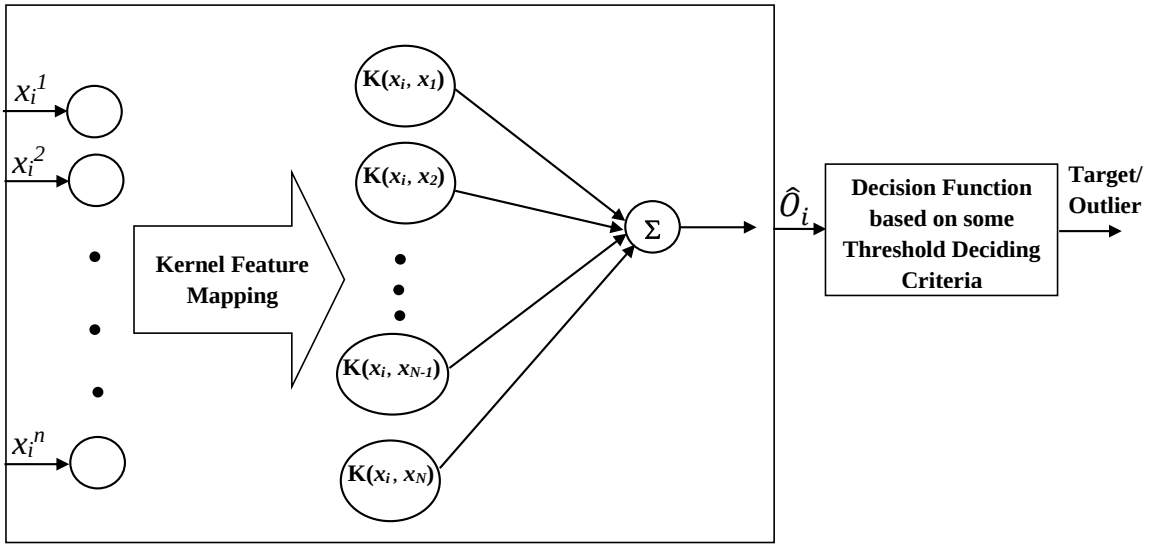


Figure 2.3: A schematic diagram of KOC

representation $\boldsymbol{\Phi}$ and reconstruction weight vector $\boldsymbol{W}$:

$$\boldsymbol{\beta} = \boldsymbol{\Phi}\boldsymbol{W}. \tag{2.12}$$

Hence, the minimization criterion in (2.11) is reformulated to the following:

$$\begin{aligned} \text{Minimize} : \pounds_{KOC} &= \frac{1}{2}(\boldsymbol{W})^{\boldsymbol{T}}(\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\Phi}\boldsymbol{W} + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{W})^{\boldsymbol{T}}(\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\phi_i} = r - e_i, \ i = 1, 2, ..., N. \end{aligned} \tag{2.13}$$

Further, we substitute $\boldsymbol{K}^{=}(\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\Phi}$, and $\boldsymbol{k_i} = (\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\phi_i}$ (where the individual elements of $\boldsymbol{k_i}$ equal to $\boldsymbol{k_{ij}} = (\boldsymbol{\phi_i})^{\boldsymbol{T}}\boldsymbol{\phi_j}, j = 1, 2, \ldots, N$) in (2.13). Now, the optimization problem in (2.13) is written as:

$$\begin{aligned} \text{Minimize} : \pounds_{KOC} &= \frac{1}{2}(\boldsymbol{W})^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{W} + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{W})^{\boldsymbol{T}}\boldsymbol{k_i} = r - e_i, \ i = 1, 2, ..., N. \end{aligned} \tag{2.14}$$

The Lagrangian relaxation of (2.14) is given below:

$$\pounds_{KOC} = \frac{1}{2}(\boldsymbol{W})^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{W} + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 - \sum_{i=1}^{N}\boldsymbol{\alpha_i}((\boldsymbol{W})^{\boldsymbol{T}}\boldsymbol{k_i} - r + e_i), \tag{2.15}$$

where $\boldsymbol{\alpha} = \{\boldsymbol{\alpha_i}\}, i = 1, 2 \ldots N$, is a Lagrangian multiplier. In order to optimize $\pounds_{KOC}$, we compute its derivatives as follows:

$$\frac{\partial \pounds_{KOC}}{\partial \boldsymbol{W}} = 0 \Rightarrow \boldsymbol{W} = \boldsymbol{\alpha}, \tag{2.16}$$

$$\frac{\partial \pounds_{KOC}}{\partial \boldsymbol{e_i}} = 0 \Rightarrow \boldsymbol{E} = \frac{1}{C}\boldsymbol{\alpha}, \tag{2.17}$$

$$\frac{\partial \pounds_{KOC}}{\partial \boldsymbol{\alpha_i}} = 0 \Rightarrow (\boldsymbol{W})^{\boldsymbol{T}}\boldsymbol{K} = \boldsymbol{r} - \boldsymbol{E}. \tag{2.18}$$

The matrix $\boldsymbol{W}$ is obtained by substituting (2.17) and (2.18) into (2.16), and is given by:

$$\boldsymbol{W} = \left(\boldsymbol{K} + \boldsymbol{I}\frac{1}{C}\right)^{-1}\boldsymbol{r}, \tag{2.19}$$

where $\boldsymbol{I}$ is an identity matrix. Further, $\boldsymbol{\beta}$ can be derived by substituting (2.19) in (2.12):

$$\boldsymbol{\beta} = \boldsymbol{\Phi}\left(\boldsymbol{K} + \boldsymbol{I}\frac{1}{C}\right)^{-1}\boldsymbol{r}, \tag{2.20}$$

where $\boldsymbol{r}$ is a vector having all elements equal to r. Since the value r can be arbitrary, we set it equal to $r = 1$.

The predicted output for the training data can be calculated as follows:

$$\widehat{\boldsymbol{O}} = (\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\beta} = (\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\Phi}\boldsymbol{W} = \boldsymbol{K}(\boldsymbol{W})^{\boldsymbol{T}}. \tag{2.21}$$

where $\widehat{O} = \left\{\widehat{O}_i\right\}$, and $i = 1, 2, ..., N$, is the predicted output of the training data.

After obtaining the predicted output value, we compute a threshold value based on the predicted value at the output layer. This threshold value helps in deciding whether a sample is an outlier or not. A threshold ($\theta 1$) is employed with KOC, which is determined as follows:

(i) We calculate the distance between the predicted value of the i^{th} training sample and r, and store in a vector, $\mathbf{\Lambda} = \{\Lambda_i\}$ and $i = 1, 2, ..., N$, as follows:

$$\Lambda_i = \left| \widehat{O}_i - r \right|. \tag{2.22}$$

(ii) After storing all distances in $\mathbf{\Lambda}$ as per (2.22), we sort these distances in decreasing order and denoted by a vector $\boldsymbol{\Lambda_{dec}}$. Further, we reject a few percents of training samples based on the deviation. Most deviated samples are rejected first because they are most probably far from the distribution of the target data. The threshold is decided based on these deviations as follows:

$$\theta 1 = \Lambda_{dec}(\lfloor \nu * N \rfloor), \tag{2.23}$$

where $0 < \nu \leq 1$ is the fraction of rejection of training samples for deciding threshold value. N is the number of training samples and $\lfloor\ \rfloor$ denotes floor operation.

After determining a threshold value by the above procedure, during testing, a test vector $\boldsymbol{x_p}$ is fed to the trained architecture and its output $\widehat{O}_p$ is obtained. Further, compute the distance $(\widehat{\Lambda_p})$, for $\boldsymbol{x_p}$, between the predicted value $\widehat{O}_p$ of the p^{th} testing sample and r:

$$\widehat{\Lambda_p} = \left| \widehat{O}_p - r \right|. \tag{2.24}$$

Finally, $\boldsymbol{x_p}$ is classified based on the following rule:

$$\begin{aligned} &\text{If } \widehat{\Lambda_p} \leq \text{Threshold}, \quad \boldsymbol{x_p} \text{ belongs to normal class} \\ &\text{Otherwise,} \qquad\qquad\qquad\qquad\quad \boldsymbol{x_p} \text{ is an outlier.} \end{aligned} \tag{2.25}$$

Overall, we have observed in Section 2.2 that multi-layer architecture with KRR is not explored for the OCC task. In this thesis, we stack various types of Auto-Encoders to construct a multi-layer architecture. Since we didn't find any KRR-based Auto-Encoder for OCC in the literature, we first proposed a KRR-based single-layered (i.e., single hidden layer) Auto-Encoder for OCC. After that, we develop a vanilla multi-

layered architecture for OCC. Later, we also develop a Graph-Embedded multi-layered architecture for the OCC task.

2.3 Auto-Encoder

Auto-Encoder is one of the most explored architecture over the past decade [46, 104]. Auto-Encoder is a type of generative model. It learns a latent representation in an unsupervised manner from the input and uses this representation to reconstruct input at the output layer [105]. Generally, the reconstructed image seems blurry due to loss of information (see in Figure 2.4). Here, the main concern is how well the Auto-Encoder represents data at the output layer. As it can be seen in Figure 2.5, when we pass the noisy samples to the trained model of the Auto-Encoder, then it reconstructs the correct pattern at the output layer without noise. It provides a better representation of the input data, which improves the classifier's performance. It has been employed for various types of tasks viz., compression / dimensionality reduction [106], semi-supervised learning [107], representation and multi-task learning [108], pattern generation [109], noise reduction [110], anomaly detection /OCC [30, 111, 112, 113], information retrieval for texts [114] and images [115, 116], transfer learning [117], and generating higher resolution images [118]. In this thesis, we are focusing on non-iterative learning-based stacked Auto-Encoder for anomaly detection, which is discussed in detail in the further chapters.

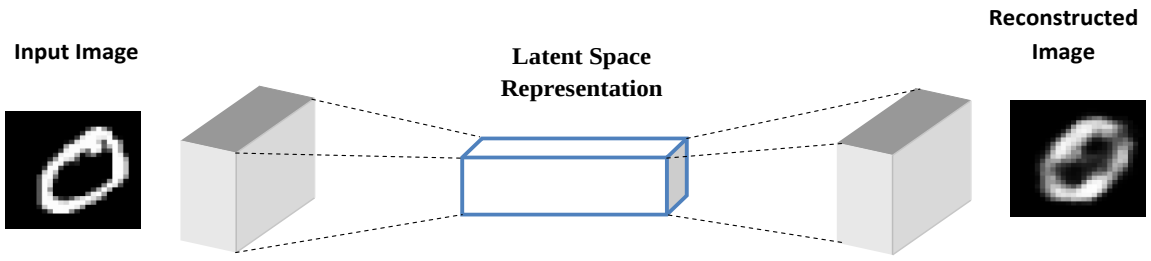


Figure 2.4: A schematic diagram of Auto-Encoder during training

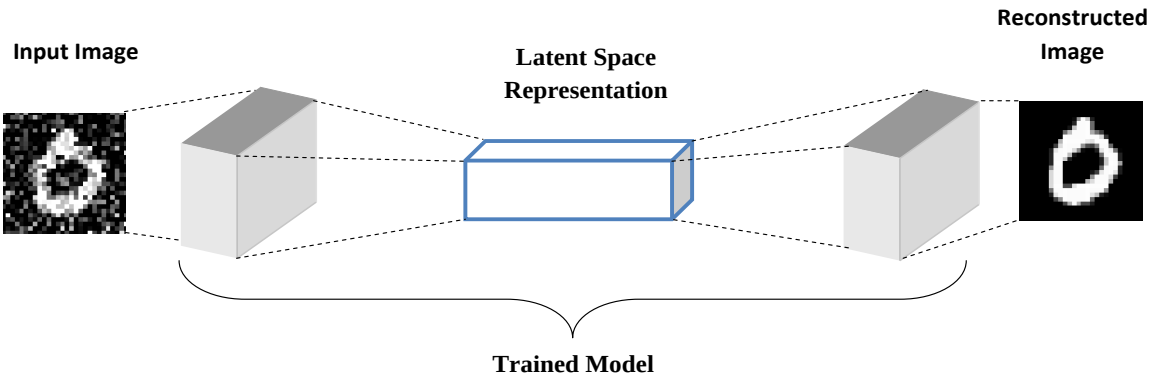


Figure 2.5: A schematic diagram of Auto-Encoder during testing

2.4 Graph-Embedding

Generally, machine learning algorithms build a model based on the statistical information of the data. These algorithms are theoretically well-founded, and various algorithms are available, which can handle statistical information efficiently, but we cannot state the same about structural information. The traditional algorithm generally ignores it while building the model. Conversely, structural information has well-representation power, which can be represented by using a graph [119]. However, the problem is:

How to merge statistical information with structural information for any machine learning algorithm?

This issue is addressed by developing a Graph-Embedding approach [120, 121]. Figure 2.6 provides a schematic diagram of Graph-Embedding. A graph can provide an interesting representation of any real-world dataset [47, 121]. These Graphs capture the graph topology and the relationship between the vertices and other properties of the graph. Various applications are benefited from Graph analytics viz; link-prediction [122], node classification [123], node recommendation [124], node clustering [125], etc. Graph-Embedding methods preserve the structural property of the data in the embedding space [47]. Various types of Laplacian graphs have been explored in Graph-Embedding for preserving different types of structural information. Let us define an

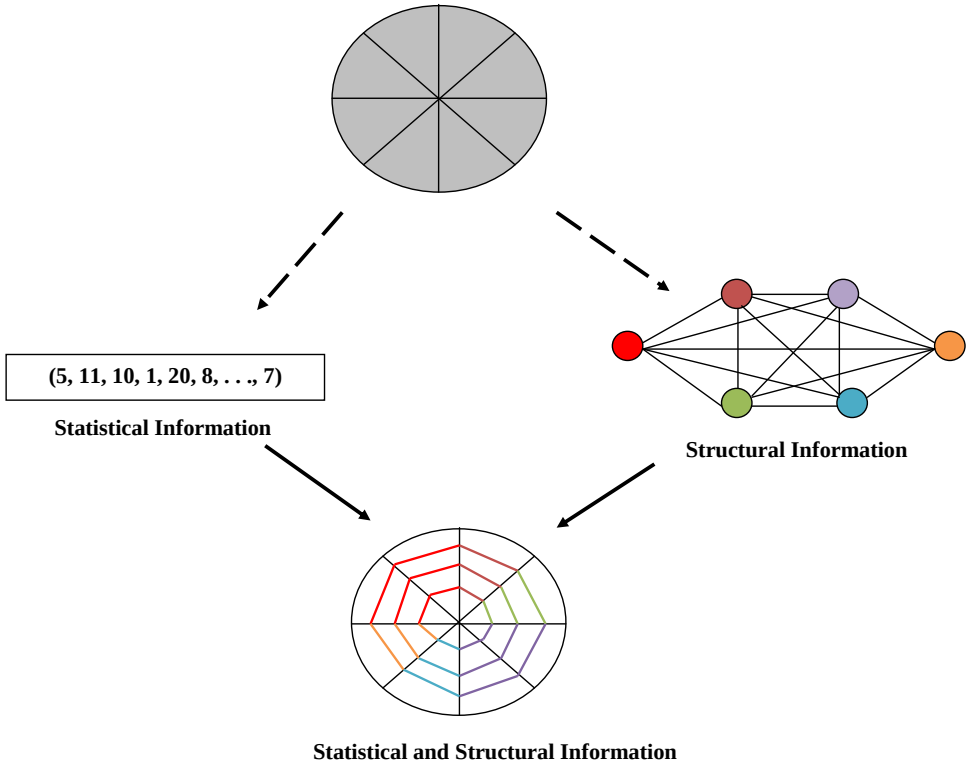


Figure 2.6: A schematic diagram of Graph-Embedding

undirected weighted graph $\mathcal{G} = \{\boldsymbol{X}, \boldsymbol{V}\}$, where $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N}]^{\boldsymbol{T}}$ is a matrix with a set of vertex, and $\boldsymbol{V} \in \mathbb{R}^{N \times N}$ is the weight matrix expressing similarities between the graph vertices.

$$\boldsymbol{D_{ii}} = \sum_{j} \boldsymbol{V_{ij}}. \tag{2.26}$$

The graph Laplacian matrix is computed as $\boldsymbol{\mathcal{L}} = \boldsymbol{D} - \boldsymbol{V}$, where $\boldsymbol{D}$ is a diagonal degree matrix. Laplacian matrix is a compressed representation of the data and presented by an Adjacency matrix, which describes connections between nodes in the graph [47, 121]. If the graph has $|V|$ number of nodes, then the size of the adjacency matrix will be $|V| \times |V|$. A node is represented by a column and a row in this matrix. If two nodes are connected in the graph, then the value between two nodes will be non-zero in the matrix. It is illustrated in the following example using Figure 2.7:

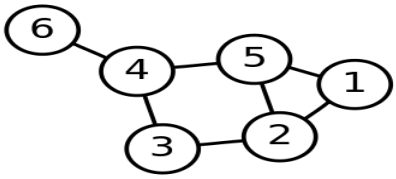


Figure 2.7: Example for representing Laplacian graph

$$\boldsymbol{D} \quad - \quad \boldsymbol{V} \quad = \quad \boldsymbol{\mathcal{L}}$$

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}. \quad (2.27)$$

Entries of Laplacian graph ($\mathbf{L}$) in (2.27) can be represented in the form of following equation:

$$\mathcal{L}_{ij} = \begin{cases} -1, & \text{if } i = j \\ D_{ii} \quad , & \text{if } i \neq j \text{ and vertices are adjacent} \\ 0, & \text{otherwise.} \end{cases} \quad (2.28)$$

Various types of Laplacian graphs can be used for embedding. Few of them, which are used in this thesis for the proposed methods, are discussed in the following sections.

2.4.1 Embedding using Laplacian Graph

For an undirected weighted graph $\mathcal{G} = \{\boldsymbol{X}, \boldsymbol{V}\}$, suppose $\boldsymbol{X}$ is projected into other dimension and termed as $\boldsymbol{Y} = [\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_N}]^{\boldsymbol{T}}$. Consider a problem where connected points on the graph stay as close as possible after embedding. We select $\boldsymbol{y_i} \in \mathbb{R}$ to minimize

$$\sum_{i,j}(\boldsymbol{y_i}-\boldsymbol{y_j})^2 V_{ij}, \tag{2.29}$$

under appropriate constraint [126]. For any $\boldsymbol{Y}$, we can rewrite (2.29) as follows:

$$\frac{1}{2}\sum_{i,j}(\boldsymbol{y_i}-\boldsymbol{y_j})^2 V_{ij} = \boldsymbol{Y}^T\boldsymbol{\mathcal{L}}\boldsymbol{Y}, \tag{2.30}$$

where $\boldsymbol{\mathcal{L}} = \boldsymbol{D}-\boldsymbol{V}$. Here, it is to be noted that from (2.26), $\boldsymbol{V}$ is symmetric and $D_{ii} = \sum_j V_{ij}$. Now, we can derive R.H.S. of (2.30) as follows:

$$\begin{aligned}
\sum_{i,j}(\boldsymbol{y_i}-\boldsymbol{y_j})^2 V_{ij} &= \sum_{i,j}(\boldsymbol{y_i^2}+\boldsymbol{y_j^2}-2\boldsymbol{y_i y_j})V_{ij} \\
&= \sum_i(\sum_j V_{ij})\boldsymbol{y_i^2} + \sum_j(\sum_i V_{ij})\boldsymbol{y_j^2} - 2\sum_{ij}\boldsymbol{y_i y_j}V_{ij} \\
&= \sum_i \boldsymbol{y_i^2}D_{ii} + \sum_j \boldsymbol{y_j^2}D_{jj} - 2\sum_{ij}\boldsymbol{y_i y_j}V_{ij} \\
&= 2\boldsymbol{Y}^T(\boldsymbol{D}-\boldsymbol{V})\boldsymbol{Y} \\
&= 2\boldsymbol{Y}^T\boldsymbol{\mathcal{L}}\boldsymbol{Y}.
\end{aligned} \tag{2.31}$$

For the sake of mathematical convenience, we multiply by $\frac{1}{2}$ to (2.29) and obtain (2.30).

Now, we have understood about the formulation of Graph-Embedding using any Laplacian graph. Different types of Laplacian graphs are briefly discussed below:

(i) Laplacian Eigenmap(LE): It is based on the idea of manifold unsupervised learning [126]. It preserves the similarities of the neighboring points [127] as shown in Figure 2.8. It constructs a neighboring graph $\mathcal{G} = \{\boldsymbol{X}, \boldsymbol{V}\}$ as follows:

$$V_{ij} = \begin{cases} v_{ij}, & \text{if } \boldsymbol{x_j} \in \mathcal{N}_i \\ 0, & \text{otherwise,} \end{cases} \tag{2.32}$$

where $\mathcal{N}_i$ denotes the neighborhood of $\boldsymbol{x_i}$. For computing similarity between $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, heat kernel function [126, 127] is employed as follows:

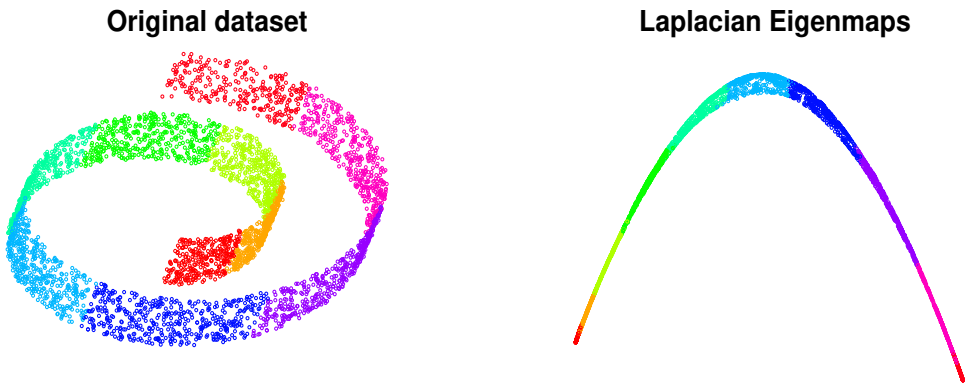


Figure 2.8: Performance of Laplacian Eigenmap on Swiss Roll datasets

$$v_{ij} = exp\left(-\frac{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|_2^2}{2\sigma^2}\right), \tag{2.33}$$

where σ is a hyper-parameter scaling the squared Euclidean distance between $\boldsymbol{\phi_i^h}$ and $\boldsymbol{\phi_j^h}$.

(ii) Locally Linear Embedding (LLE): It also preserves the relationship between the neighboring samples like LE (as shown in Figure 2.9), hence, it also constructs a neighboring graph $\mathcal{G} = \boldsymbol{\{X, V\}}$ using (2.32) . However, the determination of the value of v_{ij} is different from LE. In LLE, each sample is reconstructed only from its neighbors with two constraints:

(a) $v_{ij} = 1$ if $\boldsymbol{x_i} \in \mathcal{N}_i$, otherwise, 0.

(b) The rows of weight matrix sum to one, i.e., $\sum_j v_{ij} = 1$.

Further, optimal weight v_{ij} with these constraints is determined by minimizing reconstruction error as follows [128]:

$$\underset{\sum_{j\in\mathcal{N}_i} v_{ij}=1}{\text{Minimize}} \sum_i \left\| \boldsymbol{x_i} - \sum_{j\in\mathcal{N}_i} v_{ij}\boldsymbol{x_j} \right\|^2. \tag{2.34}$$

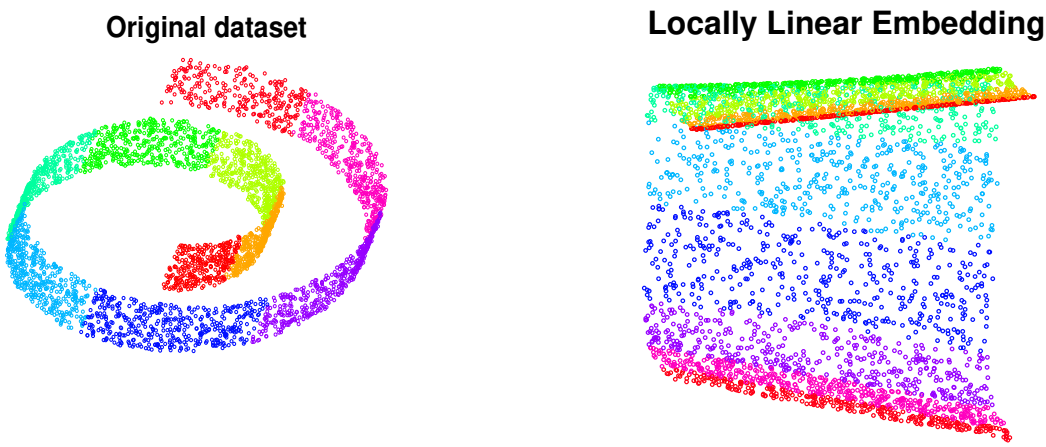


Figure 2.9: Performance of Locally Linear Embedding on Swiss Roll dataset

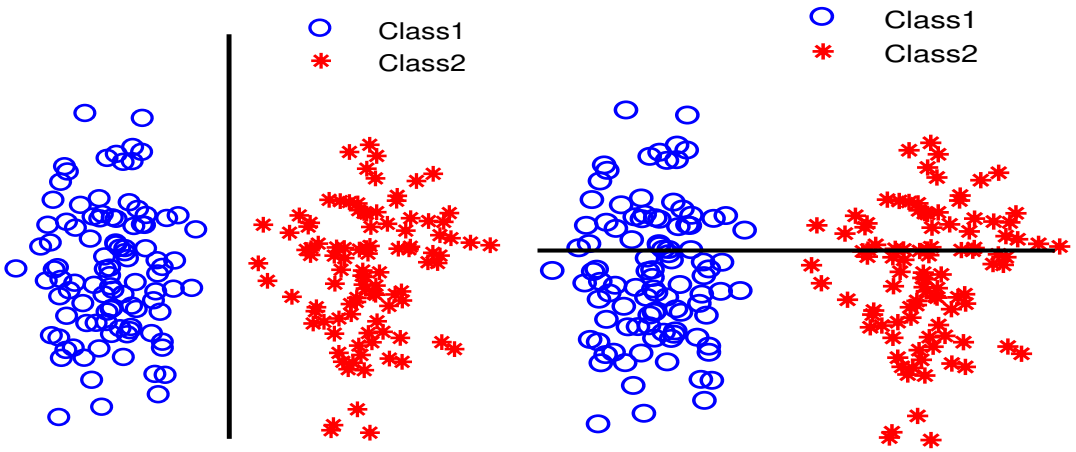
(iii) Linear Discriminant Analysis (LDA): The structure of the LDA [129] graph is determined by using the class information of the training data $\boldsymbol{X}$ as follows:

$$V_{ij} = \begin{cases} \frac{1}{N_{c_i}}, & \text{if } c_j = c_i \\ 0, & \text{otherwise.} \end{cases} \tag{2.35}$$

Here, c_i denotes the class label of i^{th} sample $\boldsymbol{x_i}$. LDA has to achieve two following objectives [129]:

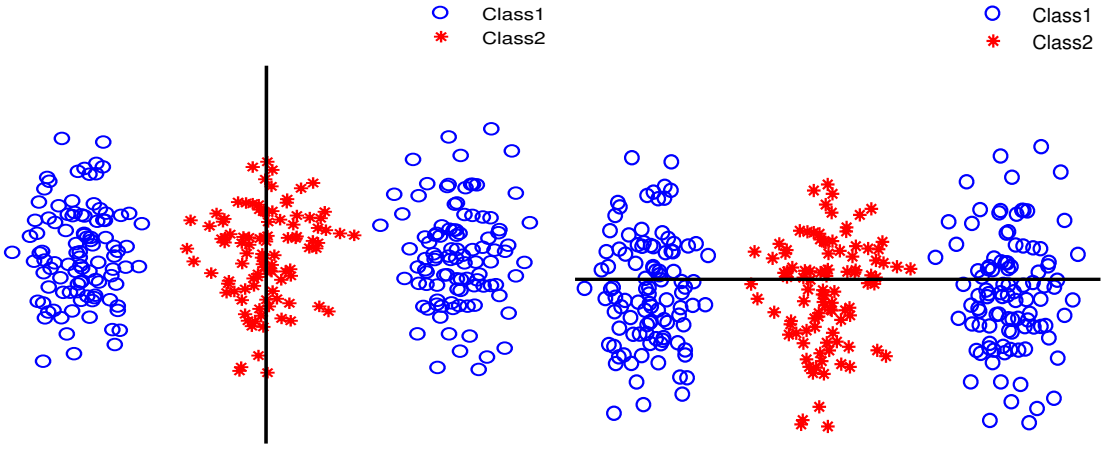
(a) Maximize between-class variance.
(b) Minimize within-class variance if class samples are close, and it does not care if they are far away.

For understanding the concept of class variance, we are taking two different types of datasets viz; unimodal and multi-modal datasets. For the unimodal dataset, principal component analysis (PCA) is employed on this data to understand the effect of maximum variance. PCA will project the data on the black line in Figure 2.10(a) because PCA chooses the direction of maximum variance and considers lower variance as a noise. However, projection on this line will lead to a strong overlap of Class1 and Class2 while LDA takes into account of the class labels and leads to a projection on the horizontal line (see in Figure 2.10(b)), which



(a) PCA (b) LDA

Figure 2.10: Projection line for PCA and LDA on unimodal dataset



(a) LDA (b) LFDA

Figure 2.11: Projection line for LDA and LFDA on multi-modal dataset

provides better class separation compared to PCA. Therefore, LDA minimizes the intra-class variance but maximizes the inter-class variance. Now, consider if multi-modality is present in the data, then LDA does not work properly, as shown in Figure 2.11(a) because it tries to minimize the intra-class variance without considering the multi-modality of that class. To handle this issue, local Fisher discriminant analysis (LFDA) [130] is proposed for the multi-modal dataset as shown in Figure 2.11(b). Graph weights are defined by LFDA as follows:

$$V_{ij} = \begin{cases} \frac{v_{ij}}{N_{c_i}}, & \text{if } c_j = c_i \\ 0, & \text{otherwise.} \end{cases} \tag{2.36}$$

In the above discussion, LE and LLE types of embedding do not depend on the class label. Therefore, both can be easily employed for OCC. However, how will we employ LDA and LFDA for OCC since these embeddings need class labels along with data? LDA and LFDA do not provide effective embedding if all data belongs to the same class, and it can be understood by the (2.35) and (2.36). For addressing this issue, a clustering method is employed with LDA [131]. Generated clusters are treated as subclasses of the target class. Now, information is available for more than one class, and LDA can be employed effectively. Hence, clustering with LDA is performed, and it is termed as CDA [131]. In the case of LFDA, if you consider that all data belongs to the same class, then it will be equivalent to LE (LE has been discussed above in this section), and the same can be analyzed from (2.32) and (2.36).

2.4.2 Graph-Embedding with KOC: GKOC

In this subsection, we are providing an optimization problem for Graph-Embedded KOC. Two types of Embeddings are explored in the literature for OCC:

(i) Local variance information-based Graph-Embedding with KOC (GKOC) [8]: It minimizes the variance by preserving the local geometry of the data after embedding [132]. Iosifidis et al. [8] have used two Laplacian graphs, like LE and LLE,

for embedding. By using these two graphs, two variants have been generated and named as GKOC-LE [8] and GKOC-LLE [8].

(ii) Global variance information-based embedding with KOC (GKOC) [8]: It minimizes the variance by preserving the global structure of the data after embedding [132]. Laplacian graphs like LDA and CDA have been used with GKOC, and two variants have been generated as GKOC-LDA [8] and GKOC-CDA [8].

The optimization problem of Graph-Embedding method can be commonly written for all three types of embedding as follows:

$$\begin{aligned} \underset{\boldsymbol{\beta}, e_i}{\text{Minimize}} : \pounds_{GKOC} &= \frac{1}{2}(\boldsymbol{\beta})^{\boldsymbol{T}}(\boldsymbol{S} + \boldsymbol{\lambda I})\boldsymbol{\beta} + \frac{C}{2}\sum_{i=1}^{N}\|e_i\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{\beta})^{\boldsymbol{T}}\boldsymbol{\phi_i} = r - e_i, \ i = 1, 2, ..., N. \end{aligned} \tag{2.37}$$

Here, the scatter matrix $\boldsymbol{S}$ encodes the local/global variance information, and can be represented as:

$$\boldsymbol{S} = \boldsymbol{\Phi}\boldsymbol{\mathcal{L}}(\boldsymbol{\Phi})^{\boldsymbol{T}}, \tag{2.38}$$

where $\mathcal{L}$ represents the Laplacian matrix for any Graph-embedding. Above minimization problem can be solved similar as discussed in Section 2.2.5.

$$\boldsymbol{\beta} = \boldsymbol{\Phi}\left(\boldsymbol{K} + \frac{1}{C}\boldsymbol{\mathcal{L}}\boldsymbol{K} + \frac{\lambda}{C}\boldsymbol{I}\right)^{-1}\boldsymbol{r}. \tag{2.39}$$

At last, the decision process for a test vector, whether it is outlier or not, is also same as discussed in Section 2.2.5.

2.5 Multiple Kernel Learning

The general idea of kernel-based methods such as SVM is to project the input space to a higher dimension where they become linearly separable. Kernel-based methods have received considerable attention over the last decade due to their success in classification problems. An advance in kernel-based methods for classification problems is to use many different kernels or different parameterization of kernels instead of a

single fixed kernel [133] to get better performance. This method is named as multiple kernel learning (MKL) and it is shown in Figure 2.12.

Multiple kernel learning provides two advantages. Firstly, this provides flexibility to select for an optimal kernel or parameterizations of kernels from a larger set of kernels, thus reducing bias due to kernel selection and at the same time allowing for a more automated approach [48, 134, 135]. Secondly, multiple kernels are also reflective of the need to combine knowledge from different data sources (such as images and sound in video data). Thus they accommodate the different notions of similarity in the different features of the input space. A further advance in multiple kernel learning is to localize the kernel selection process for various tasks, like binary classification [12], image recognition problems [136], regression [137], and clustering [138]. Gonen and Alpaydin [12, 136, 137] achieve this localization by using a gating function, which helps in the selection of the appropriate kernel locally. This method is known as localized multiple kernel learning (LMKL). Various extensions of LMKL method have been developed [139, 140, 141, 142]. Han et al. [139] developed L_p norm LMKL via semi-definite programming. Further, Han et al. [140] formulated it via sample-wise alternating optimization. Lei et al. [141] formulated it as a convex optimization problem over a given cluster structure. Han et al. [142] developed it with dynamical clustering and matrix regularization. During the last decade, there are few attempts by researchers [11, 143, 144] to transfer the idea of multiple kernel learning to the

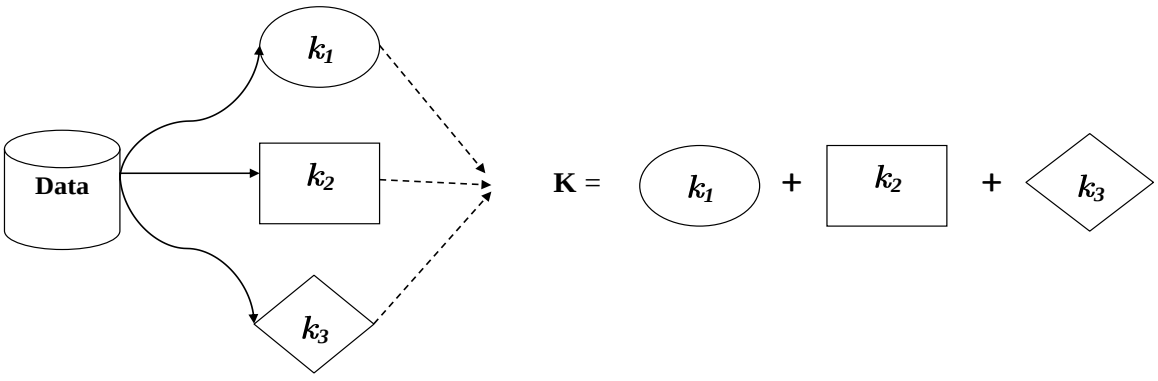


Figure 2.12: A schematic diagram of multi-kernel learning. In this diagram, k_1, k_2, and k_3 represent different kernels. K represents combination of kernels.

domain of OCC. GöNen and Alpaydin [12] also mentioned in future directions of their work that we can develop it for OCC. Das et al. [11] propose a simple weighted sum of two kernels, each of which describes the discrete and continuous streams in aviation data, respectively. This method is named as Multiple Kernel Learning for Anomaly Detection (MKAD) and briefly discussed in the following section.

2.5.1 Multiple Kernel Learning for Anomaly Detection: MKAD

Das et al. [11] proposed MKAD to detect anomalies in aviation data. Aviation data consists of features that can be grouped into two categories - (i) Real-valued data such as flight velocity, altitude, and flap angle, and (ii) Binary valued data such as cockpit switch positions. Single-kernel OCSVM cannot capture the different notions of similarity in the Real and Binary valued data. Instead, a composite kernel $\boldsymbol{K}$ is used as follows:

$$\mathbb{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \sum_{m=1}^{p} \eta_m \boldsymbol{k_m}(\boldsymbol{x_i}, \boldsymbol{x_j}), \tag{2.40}$$

where $\eta_m \geq 0$ and $\sum_{m=1}^{p} \eta_m = 1$. Here $\boldsymbol{k_m}(\boldsymbol{x_i}, \boldsymbol{x_j})$ represents the m^{th} kernel computed for data points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$, and η_m denotes assigned weight to individual kernels. The dual of this optimization problem is similar to that of OCSVM, with the kernel replaced by the composite kernel.

Note that here, the advantage of the multiple kernel learning approach is to incorporate knowledge of the differing notions of similarity in the decision process. Thus, we can achieve an improvement in detecting anomalies in a system that involves various data sources. A fixed combination rule (like a weighted summation or product) assigns the same weight to a kernel, which remains fixed over the entire input space. However, this does not take into account the underlying localities in the data. Assigning different weights to a kernel in a data-dependent way may lead to a further improvement in detecting the anomalies. We explore this possibility in this thesis.

As per our findings in Section 2.5, the method proposed by Das et al. [11] took advantage of multiple kernel learning to incorporate the different notions of similarity for the anomaly detection task in a heterogeneous system. This method exhibited better performance compared to single kernel learning. Although the method proposed by Das et al. [11] took advantage of the multi-kernel learning for OCC, they combined the kernel using only fixed combination rule, i.e., assigning equal weight to each kernel over the whole input space. In this thesis, the MKL-based proposed methods decide the weight of each kernel based on the locality presents among data. Overall, by taking the clue from these papers [145, 12, 146], we have extended the concept of localized multiple kernel learning for both the SVM-based OCC methods (OCSVM and SVDD).

2.6 Learning with Privileged Information

As we have discussed in the previous sections that SVM-based one-class classifiers are broadly based on two different types of methods viz., OCSVM [1] and SVDD [31]. These methods have been further enabled to utilize the privileged information during learning by using a well-known framework, i.e., learning using privileged information (LUPI) framework [147, 13]. This framework is inspired by the way human learns in the real-world. In the real-world, human learns not just by looking at an object but also learns by listening to extra information provided by someone like student-teacher learning. A teacher plays a very crucial role in human learning. LUPI framework introduces human teaching into traditional machine learning. An intelligent teacher provides some explanation along with an example to the student. This additional information is known as privileged or side information. In this framework, some additional information $\boldsymbol{X}^*$ is available corresponding to a dataset X at the training/learning stage [13]. However, this information is not available at the testing stage. We can understand this from the following practical examples [13]:

(i) Suppose we have to build a model to classify biopsy images into two categories: cancer and non-cancer. Here, the model classifies based on the pixel information

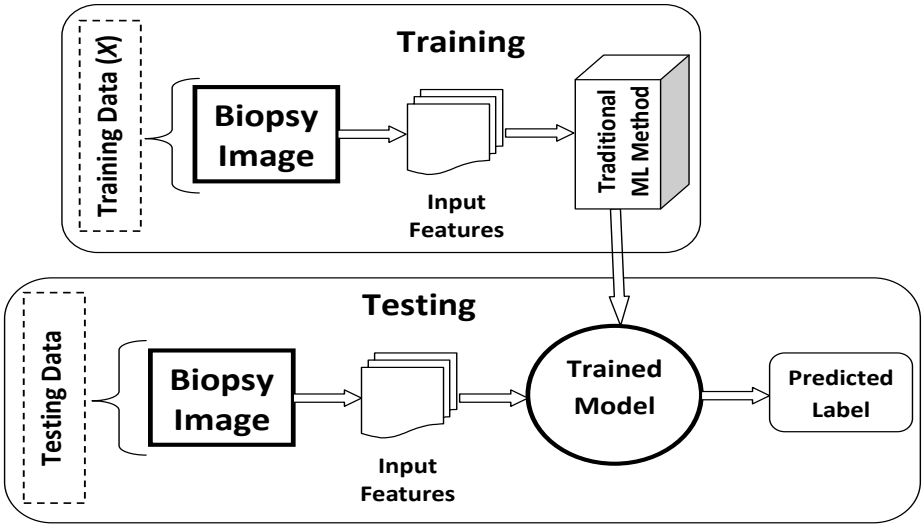
of the image; however, additional information can be provided along with the image [13]. This additional information can be a description of the image, which is written by a pathologist. This description provides additional information, i.e., a description of the pictures using a high-level holistic language. However, this additional information will not be available at the test stage because our goal is to build a model to provide an accurate diagnosis without consulting a pathologist [13]. The traditional and LUPI setting are shown in Figures 2.13(a) and 2.13(b), respectively. These figures also show the difference between traditional machine learning and LUPI setting.

(ii) Suppose that we have to build a model, which can predict the outcomes of treatment in a year based on the current symptoms (X) of disease. However, we can provide additional information (X^*) about the development of symptoms in 3, 6, and 9 months. This additional information can improve the prediction; however, this information will not be available at the testing stage.

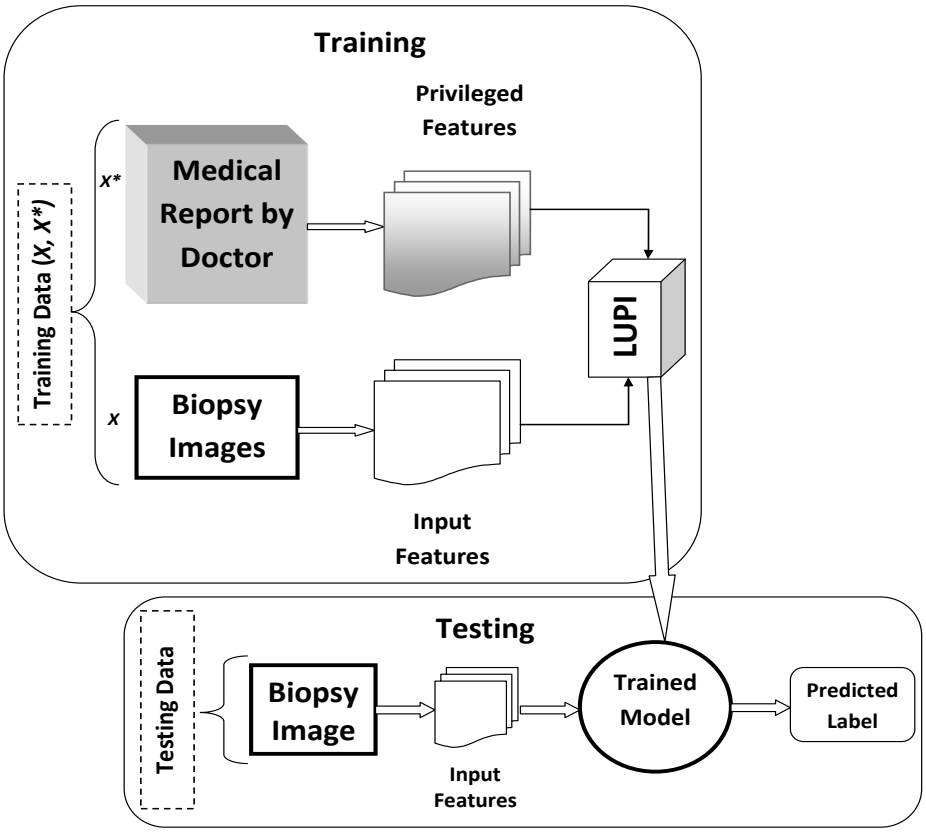
(iii) Vapnik and Vashisht [13] provided an interesting example on MNIST dataset by taking digit 5 and 8. Some sample images of 5 and 8 are provided in Figure 2.14. They generated the holistic (poetic) description of the image and treated it as privileged information. The poetic description can be understood by the following examples, which were originally provided by Vapnik and Vashist [13].

Example-1: For digit 5 **[13]**

> *Not absolute two-part creature. Looks more like one impulse. As for two-partness the head is a sharp tool and the bottom is round and flexible. As for tools it is a man with a spear ready to throw it. Or a man is shooting an arrow. He is firing the bazooka. He swung his arm, he drew back his arm and is ready to strike. He is running. He is flying. He is looking ahead. He is swift. He is throwing a spear ahead. He is dangerous. It is slanted to the right. Good snakedness. The snake is attacking. It is going to jump and bite. It is*



(a) Traditional classification



(b) LUPI-based Classification

Figure 2.13: Setting of traditional classification and LUPI-based classification

free and absolutely open to anything. It shows itself, no kidding. Its bottom only slightly (one point!) is on earth. He is a sportsman and in the process of training. The straight arrow and the smooth flexible body. This creature is contradictory - angular part and slightly roundish part. The lashing whip (the rope with a handle). A toe with a handle. It is an outside creature, not inside. Everything is finite and open. Two open pockets, two available holes, two containers. A piece of rope with a handle. Rather thick. No loops, no saltire. No hill at all. Asymmetrical. No curlings.

Example-2: For digit 8 **[13]**

Two-part creature. Not very perfect infinite way. It has a deadlock, a blind alley. There is a small right-hand head appendix, a small shoot. The right-hand appendix. Two parts. A bit disproportionate. Almost equal. The upper one should be a bit smaller. The starboard list is quite right. It is normal like it should be. The lower part is not very steady. This creature has a big head and too small bottom for this head. It is nice in general but not very self-assured. A rope with two loops which do not meet well. There is a small upper right-hand tail. It does not look very neat. The rope is rather good - not very old, not very thin, not very thick. It is rather like it should be. The sleeping snake which did not hide the end of its tail. The rings are not very round - oblong - rather thin oblong. It is calm. Standing. Criss-cross. The criss-cross upper angle is rather sharp. Two criss-cross angles are equal. If a

Figure 2.14: Sample images of digit 5 and 8

tool it is a lasso. Closed absolutely. Not quite symmetrical (due to the horn).

Here, Vapnik and Vashist [13] have passed this description with the MNIST images during training. They have shown a significant improvement in the performance of the classifier after adding this information as privileged information.

The whole purpose of LUPI framework is to explore the privileged information to improve the performance of the classifier. Further, Vapnik and Izmailov [14] improve the speed of student's learning using privileged information by correction of student's concepts of similarity between examples, and direct teacher-student knowledge transfer. Moreover, this concept has been explored for various types of tasks viz., data clustering [148], face verification [149], visual recognition [150], malware detection [39], multi-label classification [151], deep learning [152], and classification of low-resolution images [153].

In recent years, researchers employed the LUPI framework for OCC task. Zhu and Zhong [37] combined the LUPI framework with OCSVM and named as OCSVM+, and Zhang [38] combined the LUPI framework with SVDD and named as SVDD+. Further, Burnaev and Smolyakov [39] modified the formulation of OCSVM+ and SVDD+ by adding a regularization factor on the privileged feature space. OCSVM+ and SVDD+ exhibited significant improvement over tradition OCSVM and SVDD. Formulations of OCSVM+ and SVDD+ are briefly mentioned below:

2.6.1 LUPI framework with OCSVM: OCSVM+

Let us assume the input training matrix of size N is $\boldsymbol{X} = \{\boldsymbol{x_i}\}$, where $\boldsymbol{x_i} = [x_{i1}, x_{i2}, ..., x_{in}]$, $i = 1, 2, ..., N$, is the n-dimensional input vector of the i^{th} training sample. We assume that privileged information $\boldsymbol{X^*} = \{\boldsymbol{x_i^*}\}$, where $i = 1, 2, ..., N$, is available with feature space of X as $(\boldsymbol{x_i}, \boldsymbol{x_i^*})$ during training. The minimization function of OCSVM+ is as follows [37]:

$$\begin{aligned}
\underset{\boldsymbol{\omega},\boldsymbol{\xi},\rho,\boldsymbol{\omega^*},b^*}{\text{Minimize}} : \frac{\nu N}{2}\|\boldsymbol{\omega}\|_2^2 + \frac{\mu}{2}\|\boldsymbol{\omega^*}\|^2 - \nu N\rho + \sum_{i=1}^{N}[(\boldsymbol{\omega^*}.\boldsymbol{\phi^*}(\boldsymbol{x_i^*})) + b^* + \xi_i] \\
\text{Subject to : } (\boldsymbol{\omega}.\boldsymbol{\phi}(\boldsymbol{x_i})) \geq \rho - (\boldsymbol{\omega^*}.\boldsymbol{\phi^*}(\boldsymbol{x_i^*})) - b^*, \quad i = 1, \ldots, N \\
\text{Subject to : } (\boldsymbol{\omega^*}.\boldsymbol{\phi^*}(\boldsymbol{x_i^*})) + b^* + \xi_i \geq 0, \quad i = 1, \ldots, N \\
\text{Subject to : } \xi_i \geq 0, \quad i = 1, \ldots, N,
\end{aligned} \tag{2.41}$$

where $\boldsymbol{\omega}$ denotes weight matrix, and $\boldsymbol{\omega^*}$ is a correction weight. $\phi(.)$ denotes kernel feature mapping function, and $\phi^*(.)$ is a feature mapping in the privileged space. μ is a regularization parameter, ν is a fraction of rejection, and ξ_i is a slack variable for the i^{th} pattern. ρ and b^* are the bias terms.

2.6.2 LUPI framework with SVDD: SVDD+

Similar as OCSVM+, Zhang [38] developed LUPI framework for SVDD as follows:

$$\begin{aligned}
\underset{\boldsymbol{\xi},\boldsymbol{\omega^*},b^*,\boldsymbol{a},R}{\text{Minimize}} : \nu N R + \frac{\mu}{2}\|\boldsymbol{\omega^*}\|^2 + \sum_{i=1}^{N}[(\boldsymbol{\omega^*}.\boldsymbol{\phi^*}(\boldsymbol{x_i^*})) + b^* + \xi_i] \\
\text{Subject to : } \|\boldsymbol{\phi}(\boldsymbol{x_i}) - \boldsymbol{a}\|^2 \leq R + [(\boldsymbol{\omega}.\boldsymbol{\phi^*}(\boldsymbol{x_i^*})) + b^*], \quad i = 1, \ldots, N \\
\text{Subject to : } (\boldsymbol{\omega^*}.\boldsymbol{\phi^*}(\boldsymbol{x_i^*})) + b^* + \xi_i \geq 0, \quad i = 1, \ldots, N \\
\text{Subject to : } \xi_i \geq 0, \quad i = 1, \ldots, N,
\end{aligned} \tag{2.42}$$

where $\boldsymbol{a}$ is a center, and R is a radius of the hypersphere.

Overall, we have observed in Section 2.6 that only iterative learning-based kernelized one-class classifiers are developed for the LUPI framework. These methods consume more time in training the model due to the iterative nature of learning. Therefore, we develop non-iterative learning-based one-class classifiers to utilize privilege information during training.

2.7 Online Learning

The literature reviewed till now is based on offline learning (i.e., batch learning). In offline learning, whole data is available for training. Therefore, whole data can be processed at once. Conversely, in a real-world scenario, data comes as a sequence of chunks. In these days, data is generating continuously, and their characteristics also change as time passes. These data are non-stationary. Generally, traditional machine learning techniques are compatible with stationary data but can not handle these types of data. For handling such types of data, researchers [15] enabled traditional algorithms for online learning, which can handle non-stationary and streaming data. Online learning has attracted researchers in recent years due to its capability to handle a high volume of streaming data with less computational and storage costs [154, 15, 155]. There are various techniques involve in online OCC, and the complete workflow for online OCC in a non-stationary environment is depicted in Figure 2.15. These techniques can be broadly divided into two steps [15]:

(i) **Change detection**: First, we need to identify whether data distribution is changed or not. If changes occur, then the model needs to be updated. Change

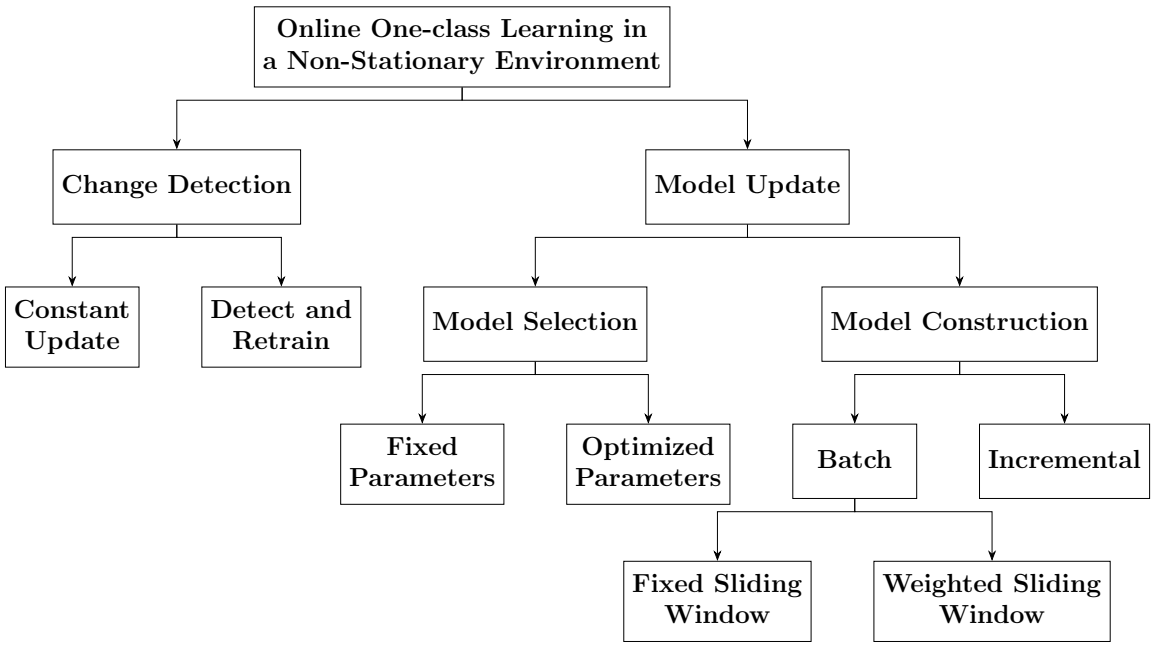


Figure 2.15: Complete work-flow for online OCC in a non-stationary environment

detection techniques can be categorized into two parts:

(a) **Constant update**: In this way of detection, we presume that data distribution is changing at a regular interval, and therefore, the model also needs to update at the regular interval. It reduces the overhead of change detection. However, if the model update time interval is lesser than the time interval of change in the non-stationary distribution, then the model needs to be updated unnecessarily, which also increases computational complexity. On the other hand, if the model update time interval is greater than the time interval of change in the non-stationary distribution, then anomaly detection ability of the classifier reduces. Zhang et al. [156, 157] provided a constant update algorithm for adapting the change in the data distribution. They have implemented their algorithm by using a sliding window.

(b) **Detect and Retrain**: This technique overcomes the issue of constant update method and updates the model only when it is required. Therefore, it reduces the computational complexity of online classifier. In addition to the constant update, Zhang et al. [156, 157] also provided an algorithm for detecting the change in the distribution.

(ii) **Model update**: After detection the change, it has to decide whether the model needs to be updated or not. This process can be categorized into two parts: **model selection** and **model construction**.

In model selection, the goal is to select a model (which is built on the current training set), which can perform well on the testing set. It can be further categorized into two parts as follows [15]:

(a) **Fixed parameters**: In this technique, the parameter is tuned and selected only once before deployment. Once the model is deployed, then the selected parameter does not change irrespective of the change in the data distribution. This technique requires low computational complexity as there is no need to select an optimal parameter for the new training set. Rajasegarar

et al. [158] applied fixed-parameter technique in their work for anomaly detection using OCSVM. There is one drawback in this technique that we are not selecting the optimal parameter once the model is deployed. It means that only one model is constructed for every new training set, which might lead to the degradation of the performance of an online classifier.

(b) **Optimized parameters**: This technique resolves the issue of fixed parameters technique by selecting the optimal model from the set of models. The set of models is constructed by using different values of parameters and selects the optimal parameter to construct the optimal model. Ratsch et al. [159] employed a heuristic approach to select the optimal parameter for OCSVM. Chatzigiannakis et al. [160] developed a kernel principal component analysis (KPCA) [1] based online learning for anomaly detection, which also selects the optimal model among various constructed models.

After model selection, the next part is model construction, which can be categorized into two parts [15]:

(a) **Batch learning**: This learning mode simply discards old training samples and reconstructs the new model from scratch for the current training set. There are two ways of learning, either by using a fixed sliding window or weighted sliding window. The fixed sliding window can be employed in two ways; either fix the size of the sliding window, or samples from the current window should be valid for a fixed period. Xie et al. [161] proposed a fixed sliding window approach for anomaly detection. It is a simple but effective technique for those cases where training data is valid for a fixed period. Any classifier which assumes that data comes from stationary distribution can be employed for the non-stationary data by using this sliding window approach. In a weighted sliding window approach, more weight is assigned to the newly arrived samples. We can assign different weights to the different part of the sliding window based on some predefined rule. Overall, this learning mode

[1] Hoffmann [4] developed a KPCA-based offline learning approach for novelty detection.

is simple but computationally expensive.

(b) **Incremental learning**: Construction of the model by using this learning can be accomplished into two steps. In the first step, at time t, the previous model, $model(t-1)$, is updated by forgetting certain obsolete or old samples. In the second step, the new model, $model(t)$, is constructed by using the model available after the first step (i.e., $model(t-1)$) and the newly arrived samples at time t. This way of learning reduces the computational cost of model construction by reusing the old model. A framework for incremental learning is discussed in detail by Subramaniam et al. [162].

Similar to offline learning, most of the kernel-based online one-class classifiers considered SVM as a base classifier [154, 15, 155]. SVM based online one-class classifier (i.e., incremental SVDD (incSVDD)) has been developed by Tax and Laskov [40] for online learning. Further, researchers employed it for change detection [41], outlier detection in wireless sensor network [163], malware detection in cloud computing infrastructures [164], multiple human tracking [165] and structural health monitoring [42].

In the discussed literature of Section 2.7, SVM is primarily considered as a base classifier for online OCC, and it uses the iterative approach of learning. The iterative approach is very time consuming, and it is a primary concern for online learning as we need to update the model as soon as new samples arrive. In this thesis, we address this issue by developing KRR-based online one-class classifiers. Since KRR uses a non-iterative approach to learning, it consumes very less time and speeds up the training process.

2.8 Performance Criteria

For evaluating the performance of the one-class classifiers, generally, researchers [2, 8] use Geometric mean[2] (η_g) or F1-scores (η_{F1}) as a performance measure. Both mea-

[2]It is also known as Gmean

sures consider Precision (η_{pr}) and Recall (η_{re}) with equal weightage in its formulation as follows:

$$\eta_g = \sqrt{\text{Precision} * \text{Recall}} \tag{2.43}$$

$$\eta_{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.44}$$

where

$$\eta_{pr} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{2.45}$$

$$\eta_{re} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

We consider η_g as the first performance measure in the whole thesis. Since we have to compare multiple classifiers on various datasets, we compute mean of all Gmean (η_m) and Friedman Rank (η_f) [166] over all datasets by taking inspiration from this paper [167]. We consider η_f as a final performance measure to rank all the classifiers as per their performance. Moreover, Friedman testing [166] is also performed to verify the statistical significance of the obtained results. Both performance measures η_m and η_f are computed as shown in the following example:

Example:

For computing η_m and η_f, we take a demo data in Table 2.1. Suppose, this table contains η_g values for 4 models (classifiers) corresponding 14 datasets. We compute Friedman rank by using following two steps:

(i) It ranks the algorithms for each dataset separately, the best performing algorithm getting the rank of 1, the second best rank 2, and so on. The same has been shown in Tables 2.1, 2.2.

(ii) In case of ties (like in Dataset6, Dataset8, Dataset10, Dataset11, and Dataset13), average ranks are assigned.

(iii) Further, we compute average rank of each model over all datasets. This average rank is treated as Friedman rank for each model.

Table 2.1: Demo data for computing η_m and η_f. Each value in table is treated as η_g

	Model1	Model2	Model3	Model4
Dataset1	76.30	76.80	77.10	79.80
Dataset2	59.90	59.10	59.00	56.90
Dataset3	95.40	97.10	96.80	96.70
Dataset4	62.80	66.10	65.40	65.70
Dataset5	88.20	88.80	88.60	89.80
Dataset6	93.60	93.10	91.60	93.10
Dataset7	66.10	66.80	60.90	68.50
Dataset8	58.30	58.30	56.30	62.50
Dataset9	77.50	83.80	86.60	87.50
Dataset10	100.00	100.00	100.00	100.00
Dataset11	94.00	96.20	96.50	96.20
Dataset12	61.90	66.60	61.40	66.90
Dataset13	97.20	98.10	97.50	97.50
Dataset14	95.70	97.80	94.60	97.00
Mean of η_g (η_m)	80.49	82.04	80.88	82.72

By employing the above 3 steps on the demo data available in Table 2.1, we compute η_f as mentioned in Table 2.2. After computing η_f and η_m values of all models, we present them in increasing order of η_f in Table 2.3. Model4 emerges as the best classifier among all four models. In the whole thesis, we compute η_g, η_m and η_f values by using the above-discussed procedure.

Table 2.2: Ranking each model corresponding to each datasets for computing the η_f

	Model1	Model2	Model3	Model4
Dataset1	4	3	2	1
Dataset2	1	2	3	4
Dataset3	4	1	2	3
Dataset4	4	1	3	2
Dataset5	4	2	3	1
Dataset6	1	2.5	4	2.5
Dataset7	3	2	4	1
Dataset8	2.5	2.5	4	1
Dataset9	4	3	2	1
Dataset10	2.5	2.5	2.5	2.5
Dataset11	4	2.5	1	2.5
Dataset12	3	2	4	1
Dataset13	4	1	2.5	2.5
Dataset14	3	1	4	2
Friedman Rank (η_f)	3.1	2.0	2.9	1.9

Table 2.3: η_f and η_m values of all models in increasing order of η_f

	η_f	η_g
Model4	**1.93**	82.72
Model2	2.00	82.04
Model3	2.93	80.88
Model1	3.14	80.49

Chapter 3

Kernel Ridge Regression-based Auto-Encoder for One-class Classification

In this chapter, we present a reconstruction framework-based method for OCC, which is named as KRR-based Auto-Encoder for OCC (AEKOC). This proposed work is inspired by a boundary framework-based one-class classifier [7], which was also developed by taking KRR as a base classifier. The boundary framework-based method only learns from the structure of the data; however, it does not use representation learning in its framework [46, 44]. We merge the concept of representation learning with kernel learning and propose a method AEKOC for OCC. AEKOC represents the target data in a better way by minimizing the noise effect in the data, which can lead to a better classification model. The proposed and existing state-of-the-art methods are experimented on 23 datasets, and their performance is evaluated based on various performance metrics, which are discussed in Section 2.8 of Chapter 2. The details of the proposed method are discussed next.

3.1 KRR-based Auto-Encoder for One-class Classification:AEKOC

The proposed AEKOC is a reconstruction framework-based method. It is a kernel-based Auto-Encoder, which consists of a single hidden layer. It follows least-squares

method [83, 168] for learning. The training of AEKOC is performed by reconstructing the input to the output layer, and it is depicted in Figure 3.1. Since the reconstruction framework-based methods are not primarily developed for a one-class classifier [6], a threshold is set empirically using reconstruction error using training data. A threshold is set in such a way so that if the error is more than this threshold, then that data is treated as an outlier otherwise target data. This procedure works because the obtained model after training on the target data will not be suitable for outlier data. Therefore, reconstruction error for outlier data will be higher compared to target data. Here, we propose a KRR-based Auto-Encoder with a single hidden layer for OCC. In the following section, firstly, some notations are introduced for formulating AEKOC; then, the formulation of the proposed method is described.

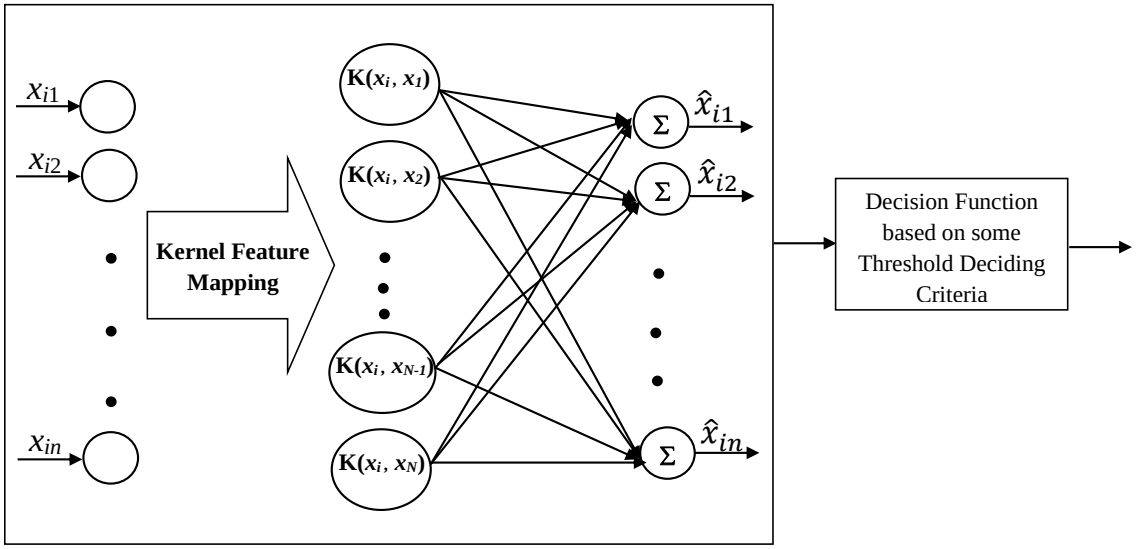


Figure 3.1: A schematic diagram of AEKOC

3.1.1 Formulation of the Proposed Method AEKOC

In this section, some notations are introduced, which are used throughout this thesis. Let us assume the input training matrix of size $N \times n$ is $\boldsymbol{X} = \{\boldsymbol{x_i}\}$, where $\boldsymbol{x_i} = [x_{i1}, x_{i2}, ..., x_{in}]$, $i = 1, 2, ..., N$, is the n-dimensional input vector of the i^{th} training sample. AEKOC minimizes the following criterion, which involves a non-

linear[1] feature mapping $\boldsymbol{X} \rightarrow \boldsymbol{\Phi}$:

$$\begin{aligned} \underset{\boldsymbol{\beta_a}, \boldsymbol{e_i}}{\text{Minimize}} : \pounds_{AEKOC} &= \frac{1}{2} \left\| \boldsymbol{\beta_a} \right\|_F^2 + \frac{C}{2} \sum_{i=1}^{N} \left\| \boldsymbol{e_i} \right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{\beta_a})^{\boldsymbol{T}} \boldsymbol{\phi_i} = (\boldsymbol{x_i})^{\boldsymbol{T}} - (\boldsymbol{e_i})^{\boldsymbol{T}}, \ i = 1, 2, ..., N, \end{aligned} \tag{3.1}$$

where C is a regularization parameter and $\boldsymbol{\beta_a}$ denotes weight matrix. $\boldsymbol{E}$ is an error matrix where $\boldsymbol{E} = \{\boldsymbol{e_i}\}$, for $i = 1, 2, ..., N$. $\|.\|_F$ and $\|.\|_2$ denotes the Frobenius norm and $l_2 - norm$, respectively. $\phi(.)$ denotes kernel feature mapping function, $\boldsymbol{\phi_i} = \boldsymbol{\phi(x_i)}$, and $\boldsymbol{\Phi} = \boldsymbol{\Phi(X)} = [\boldsymbol{\phi_1}, \boldsymbol{\phi_2}, ..., \boldsymbol{\phi_N}]$. $\boldsymbol{e_i}$ is a training error vector corresponding to the i^{th} training sample. Based on the Representer Theorem [103], we express $\boldsymbol{\beta_a}$ as a linear combination of the training data representation $\boldsymbol{\Phi}$ and a reconstruction weight matrix $\boldsymbol{W_a}$:

$$\boldsymbol{\beta_a} = \boldsymbol{\Phi W_a}. \tag{3.2}$$

Hence, by using Representer Theorem [103], minimization problem in (3.1) is reformulated as follows:

$$\begin{aligned} \underset{\boldsymbol{W_a}, \boldsymbol{e_i}}{\text{Minimize}} : \pounds_{AEKOC} &= \frac{1}{2} Tr\Big((\boldsymbol{W_a})^{\boldsymbol{T}} (\boldsymbol{\Phi})^{\boldsymbol{T}} \boldsymbol{\Phi W_a} \Big) + \frac{C}{2} \sum_{i=1}^{N} \left\| \boldsymbol{e_i} \right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{W_a})^{\boldsymbol{T}} (\boldsymbol{\Phi})^{\boldsymbol{T}} \boldsymbol{\phi_i} = (\boldsymbol{x_i})^{\boldsymbol{T}} - (\boldsymbol{e_i})^{\boldsymbol{T}}, \ i = 1, 2, ..., N. \end{aligned} \tag{3.3}$$

Here, 'Tr' represents trace of a matrix. Further, we substitute $\boldsymbol{K} = (\boldsymbol{\Phi})^{\boldsymbol{T}} \boldsymbol{\Phi}$, and $\boldsymbol{k_i} = (\boldsymbol{\Phi})^{\boldsymbol{T}} \boldsymbol{\phi_i}$ (where the individual elements of $\boldsymbol{k_i}$ equal to $\boldsymbol{k_{ij}} = (\boldsymbol{\phi_i})^{\boldsymbol{T}} \boldsymbol{\phi_j}, j = 1, 2, \ldots, N$) in (3.3). Now, the optimization problem in (3.3) is written as:

$$\begin{aligned} \underset{\boldsymbol{W_a}, \boldsymbol{e_i}}{\text{Minimize}} : \pounds_{AEKOC} &= \frac{1}{2} Tr\Big((\boldsymbol{W_a})^{\boldsymbol{T}} \boldsymbol{K W_a} \Big) + \frac{C}{2} \sum_{i=1}^{N} \left\| \boldsymbol{e_i} \right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{W_a})^{\boldsymbol{T}} \boldsymbol{k_i} = (\boldsymbol{x_i})^{\boldsymbol{T}} - (\boldsymbol{e_i})^{\boldsymbol{T}}, \ i = 1, 2, ..., N. \end{aligned} \tag{3.4}$$

[1]Linear case can be easily derived from (3.1) by substituting $\boldsymbol{\phi_i} \rightarrow \boldsymbol{x_i}$

The Lagrangian relaxation of (3.4) is written as follows:

$$\pounds_{AEKOC} = \frac{1}{2}Tr\Big((\boldsymbol{W_a})^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{W_a}\Big) + \frac{C}{2}\sum_{i=1}^{N}\|\boldsymbol{e_i}\|_2^2 - \sum_{i=1}^{N}\boldsymbol{\alpha}_i((\boldsymbol{W_a})^{\boldsymbol{T}}\boldsymbol{k_i} - (\boldsymbol{x_i})^{\boldsymbol{T}} + (\boldsymbol{e_i})^{\boldsymbol{T}}), \tag{3.5}$$

where $\boldsymbol{\alpha} = \{\boldsymbol{\alpha_i}\}, i = 1, 2 \ldots N$, is a Lagrangian multiplier. In order to optimize $\pounds_{AEKOC}$, we compute its derivatives as follows:

$$\frac{\partial \pounds_{AEKOC}}{\partial \boldsymbol{W_a}} = 0 \Rightarrow \boldsymbol{W_a} = \boldsymbol{\alpha}, \tag{3.6}$$

$$\frac{\partial \pounds_{AEKOC}}{\partial \boldsymbol{e_i}} = 0 \Rightarrow \boldsymbol{E} = \frac{1}{C}\boldsymbol{\alpha}, \tag{3.7}$$

$$\frac{\partial \pounds_{AEKOC}}{\partial \boldsymbol{\alpha_i}} = 0 \Rightarrow (\boldsymbol{W_a})^{\boldsymbol{T}}\boldsymbol{K} = \boldsymbol{X}^{\boldsymbol{T}} - \boldsymbol{E}^{\boldsymbol{T}}. \tag{3.8}$$

The matrix $\boldsymbol{W_a}$ is obtained by substituting (3.7) and (3.8) into (3.6), and is given by:

$$\boldsymbol{W_a} = \left(\boldsymbol{K} + \boldsymbol{I}\frac{1}{C}\right)^{-1}\boldsymbol{X}. \tag{3.9}$$

Now, $\boldsymbol{\beta_a}$ is derived by substituting (3.9) into (3.2):

$$\boldsymbol{\beta_a} = \boldsymbol{\Phi}\left(\boldsymbol{K} + \boldsymbol{I}\frac{1}{C}\right)^{-1}\boldsymbol{X}. \tag{3.10}$$

The predicted output for the training data is calculated as follows::

$$\widehat{\boldsymbol{O_a}} = (\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\beta_a} = (\boldsymbol{\Phi})^{\boldsymbol{T}}\boldsymbol{\Phi}\boldsymbol{W_a} = \boldsymbol{K}\boldsymbol{W_a}, \tag{3.11}$$

where $\widehat{\boldsymbol{O_a}}$ is the predicted output matrix of the training data and $\widehat{\boldsymbol{O_a}} = \{\widehat{\boldsymbol{x_i}}\}$, where $\widehat{\boldsymbol{x_i}} = [\widehat{x}_{i1}, \widehat{x}_{i2}, ..., \widehat{x}_{in}]$, $i = 1, 2, ..., N$.

After obtaining the predicted output value, we compute a threshold value based on the predicted value at the output layer. This threshold value helps in deciding whether a sample is an outlier or not. It is discussed in the following section.

3.1.2 Decision Function

A threshold ($\theta 1$) is employed with the proposed method at the output layer, which is determined as follows:

(i) We calculate the sum of square error as reconstruction error between the predicted value of the i^{th} training sample and $\boldsymbol{x_i}$, and store the distance in a vector, $\boldsymbol{\Lambda} = \{\Lambda_i\}$ and $i = 1, 2, ..., N$, as follows:

$$\Lambda_i = \sum_{j=1}^{n} (\widehat{(O_a)}_{ij} - x_{ij})^2. \tag{3.12}$$

(ii) After storing all distances in $\boldsymbol{\Lambda}$ as per (3.12), we sort these distances in decreasing order and denoted by a vector $\boldsymbol{\Lambda_{dec}}$. Further, we reject a few percents of training samples based on the deviation. Most deviated samples are rejected first because they are most probably far from the distribution of the target data. The threshold is decided based on these deviations as follows:

$$\theta 1 = \Lambda_{dec}(\lfloor \nu * N \rfloor), \tag{3.13}$$

where $0 < \nu \leq 1$ is the fraction of rejection of training samples for deciding threshold value. N is the number of training samples and $\lfloor\ \rfloor$ denotes floor operation.

After determining a threshold value by the above procedure, during testing, a test vector $\boldsymbol{x_p}$ is fed to the trained architecture and its output $\boldsymbol{\widehat{(O_a)}_p}$ is obtained. Further, compute the distance $(\widehat{\Lambda_p})$, for $\boldsymbol{x_p}$, between the predicted value $\boldsymbol{\widehat{(O_a)}_p}$ of the p^{th} testing sample and $\boldsymbol{x_p}$:

$$\widehat{\Lambda_p} = \sum_{j=1}^{n} (\widehat{(O_a)}_{pj} - x_{pj})^2. \tag{3.14}$$

Finally, $\boldsymbol{x_p}$ is classified based on the following rule:

$$\begin{aligned} &\text{If } \widehat{\Lambda_p} \leq \theta 1, \quad \boldsymbol{x_p} \text{ belongs to normal class} \\ &\text{Otherwise,} \qquad\qquad\quad \boldsymbol{x_p} \text{ is an outlier .} \end{aligned} \tag{3.15}$$

The whole procedure of training the proposed AEKOC is provided in Algorithm 3.1.

Algorithm 3.1 Training algorithm for AEKOC

Input: Training set $\boldsymbol{X}$, regularization parameter (C), kernel feature mapping ($\boldsymbol{\Phi}$)

Output: Target class or Outlier

1: Perform kernel feature mapping using a Gaussian kernel.
2: Minimize the training error and weight simultaneously using (3.1).
3: After training the model, compute the predicted output of training data using (3.11).
4: By using the predicted output of training data, compute a threshold $\theta 1$ using (3.12) and (3.13).
5: At final step, whether a new input is an outlier or not, decides based on the rule discussed in 3.15.

3.2 Experiments

All experiments in this section are carried out with MATLAB 2016a on Windows 7 (Intel Xeon 3 GHz processor, 64 GB RAM) environment. All existing and proposed one-class classifiers are implemented and tested in the same environment. For evaluating the performance of these classifiers, we have experimented over 23 one-class datasets. A brief description of these datasets is provided in Table 3.1. These one-class datasets are retrieved from the web-page[2], which are made available by Tax and Duin [169] in the preprocessed form for OCC. These one-class datasets are originally generated for the binary or multi-class class classification. For OCC experiments, Tax and Duin [169] made it compatible with OCC in the following ways. If a dataset has two or more than two classes, then alternately, we use each of the classes in the dataset as the target class and the samples of the remaining classes as an outlier. In this way, 23 one-class datasets are generated for OCC from 11 multi-class[3] datasets. These

[2] `http://homepage.tudelft.nl/n9d04/occ/`
[3] These multi-class datasets are available at UCI repository [170]

datasets belong to various disciplines viz; finance, medical, radar-emission, flower, and glass datasets. In all our experiments on these datasets, 5-fold cross-validation (CV) procedure is used for optimal parameter selection and repeated these experiments 5 times. We compute Geometric mean or Gmean (η_g) for each fold in the experiments for evaluating the performance of each of the classifiers. We also compute mean of all η_g values obtained by a classifier over all datasets and denote as η_m. Moreover, we have also performed the Friedman test and computed Friedman Rank (η_f) for ranking these classifiers as per their performance over all datasets. These three performance criteria (η_g, η_m, η_f) are briefly discussed in Section 2.8 of Chapters 2. It is to be noted that we follow the same experimental setup, datasets, and state-of-the-art methods for experimentation in Chapter 3, 4, 5, and 6.

3.2.1 Existing Kernel-based Methods

For comparing our proposed method from the existing kernel-based methods, we have selected 4 state-of-the-art existing kernel-based one-class classifiers, which are mentioned as follows:

(i) OCSVM[1] is implemented using LIBSVM library[4] [171].

(ii) SVDD[31] is implemented by using DD Toolbox[5][54].

(iii) The code of KOC[7] is provided by the author of the paper and made available at GitHub[6].

(iv) The code of KPCA[4] is obtained from this link[7] .

Here, KPCA is a reconstruction framework-based classifier and remaining three are boundary framework-based classifiers.

[4] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/

[5] https://www.tudelft.nl/ewi/over-de-faculteit/afdelingen/intelligent-systems/pattern-recognition-bioinformatics/pattern-recognition-laboratory/data-and-software/dd-tools/

[6] https://github.com/Chandan-IITI/One-Class-Kernel-ELM

[7] http://www.heikohoffmann.de/kpca.html

Table 3.1: Dataset description

S. No.	Name	Abbreviated Name	#Targets	#Anomalies	#Features	#Samples
1	Australia Credit(1)	Aust(1)	307	383	14	690
2	Australia Credit(2)	Aust(2)	383	307	14	690
3	Bupa(1)	Bupa(1)	145	200	6	345
4	Bupa(2)	Bupa(2)	200	145	6	345
5	Ecoli(1)	Ecoli(1)	143	193	7	336
6	Ecoli(2)	Ecoli(2)	193	143	7	336
7	German Credit(1)	Germ(1)	700	300	24	1000
8	German Credit(2)	Germ(2)	300	700	24	1000
9	Glass(1)	Glass(1)	76	138	9	214
10	Glass(2)	Glass(2)	138	76	9	214
11	Heart(1)	Heart(1)	160	137	13	297
12	Heart(2)	Heart(2)	137	160	13	297
13	Ionosphere(1)	Iono(1)	225	126	34	351
14	Ionosphere(2)	Iono(2)	126	225	34	351
15	Iris(1)	Iris(1)	50	100	4	150
16	Iris(2)	Iris(2)	50	100	4	150
17	Iris(3)	Iris(3)	50	100	4	150
18	Japan Credit(1)	Jap(1)	294	357	15	651
19	Japan Credit(2)	Jap(2)	357	294	15	651
20	Parkinson(1)	Park(1)	520	520	28	1040
21	Parkinson(2)	Park(2)	520	520	28	1040
22	Pima Indians Diabetes(1)	Pima(1)	500	268	8	768
23	Pima Indians Diabetes(2)	Pima(2)	268	500	8	768

3.2.2 Range of the Parameters for the Proposed and Existing Methods

All used methods in this thesis are kernel-based only. These methods employ a Radial Basis Function (RBF) kernel, which is computed for data points $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ as follows:

$$\mathbb{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = exp\left(-\frac{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|_2^2}{2\sigma^2}\right), \tag{3.16}$$

where σ is calculated as the mean Euclidean distance between training vectors in the corresponding feature space. For the KOC and AEKOC methods, regularization parameter (C) is selected from the range $\{2^{-5}, \ldots, 2^5\}$. For SVDD, the regularization parameter (C) is set as $\frac{1}{\nu * N}$. For KPCA based OCC, the percentage of the preserved variance is selected from the range $[85, 90, 95]$. The fraction of rejection (ν) of outliers during threshold selection is set equal to 0.05 for all methods presented in this thesis.

3.2.3 Performance Evaluation

Detailed results of 5 one-class classifiers in terms of η_g are presented in Table 3.2. After analyzing this table, it is observed that AEKOC yields the best results for 6 out of 23 datasets. Table 3.3 shows that number of datasets with the best η_g value for each classifier. In some cases, two or more than two classifiers yield identical and best results for the same dataset. It can be observed from Table 3.2 and 3.3 that OCSVM and SVDD yield identical and best results for 3 datasets viz., Glass(1), Heart(1), and Iono(1). We further analyze that SVDD and AEKOC yield the best results for $\approx 30\%$ and $\approx 26\%$ of datasets, respectively. Hence, we can not declare any one classifier as the best classifier just by looking at the best η_g value in Table 3.2. The same observation has been found by Fernandez et al. [167] when they tested 179 classifiers on 121 datasets. By following their work [167], we compute three criteria: (i) η_g and an average of η_g (η_m) (ii) Friedman statistical testing (iii) Friedman rank (η_f). We have analyzed the performance of the existing and proposed classifiers based on these three criteria as follows:

Table 3.2: Performance in terms of η_g for 23 datasets

Datasets	KPCA	OCSVM	SVDD	KOC	AEKOC
Aust(1)	63.69	66.08	65.55	65.07	**72.88**
Aust(2)	73.06	76.59	76.78	74.21	**77.96**
Bupa(1)	**62.91**	60.64	60.64	57.09	56.19
Bupa(2)	**74.28**	69.78	69.75	68.81	68.31
Ecoli(1)	65.79	89.43	**89.49**	89.38	89.00
Ecoli(2)	72.30	79.42	78.87	**82.32**	79.16
Germ(1)	80.77	80.34	**81.10**	73.17	74.04
Germ(2)	49.75	52.80	52.77	**53.41**	51.57
Glass(1)	57.69	**59.61**	**59.61**	58.91	59.29
Glass(2)	**77.65**	73.32	72.89	73.08	73.22
Heart(1)	70.42	**72.91**	**72.91**	65.03	67.99
Heart(2)	63.50	64.90	64.90	**66.39**	61.15
Iono(1)	76.54	**93.13**	**93.13**	92.69	92.95
Iono(2)	**57.10**	44.63	44.63	53.40	41.04
Iris(1)	**96.44**	85.06	84.12	92.35	92.79
Iris(2)	72.99	81.70	81.70	85.59	**88.37**
Iris(3)	69.29	83.18	82.67	83.40	**83.74**
Jap(1)	64.09	71.45	70.15	67.33	**76.23**
Jap(2)	72.29	75.78	76.58	73.48	**78.27**
Park(1)	70.20	95.71	**97.07**	96.15	96.67
Park(2)	67.79	81.82	79.77	**90.74**	83.78
Pima(1)	77.98	79.18	**79.21**	79.04	78.66
Pima(2)	**57.05**	56.59	56.71	54.78	54.01
Mean of all η_g (η_m)	69.28	73.65	73.52	73.73	**73.79**

Table 3.3: Number of datasets for which each one-class classifier yields best η_g

One-class Classifiers	Number of datasets with best η_g value
SVDD	7
AEKOC	6
KPCA	6
KOC	4
OCSVM	3

Table 3.4: Friedman Rank (η_f) and mean of η_g (η_m)

3.4(a) In decreasing order of η_m

	η_f	η_m
AEKOC	3.00	**73.79**
KOC	3.13	73.73
OCSVM	**2.54**	73.65
SVDD	2.63	73.52
KPCA	3.70	69.28

3.4(b) In increasing order of η_f

	η_f	η_m
OCSVM	**2.54**	73.65
SVDD	2.63	73.52
AEKOC	3.00	**73.79**
KOC	3.13	73.73
KPCA	3.70	69.28

(i) We compute η_m for analyzing the combined performance of the classifier. η_m value of each classifier is available in the last row of Table 3.2. It is also provided in Table 3.4(a) in decreasing order of η_m. When we compare η_m values, then it is observed that AEKOC performs better than all 4 existing classifiers in spite of obtaining the best η_g for 6 datasets only. One more interesting fact is observed that OCSVM shows better performance compared to KPCA and SVDD in terms of η_m, while it yields the best η_g for only three datasets.

(ii) Although AEKOC has yielded best η_m among 5 classifiers; however, we need to verify the outcomes of the proposed and existing methods statistically. For this purpose, we conduct a non-parametric Friedman test [166]. Friedman test mainly computes three components viz., p-value, F-score, and critical value. If the computed F-score is higher than the critical value, and the p-value is lesser than the tolerance level $\alpha = 0.1$, then the null hypothesis can be rejected with 90% of a confidence level. The computed p-value, F-score value, and critical value are 0.09, 7.78 and 7.77, respectively. Since the computed F-score value is

higher than the critical value, and the p-value is lesser than 0.1, we reject the null hypothesis. Therefore, it is concluded that the outcomes presented in this chapter are statistically significant.

(iii) Further, we compute Friedman Rank (η_f) [166] for each classifier as discussed in Section 2.8 of Chapter 2. We present η_f along with η_m of each classifier in Table 3.4(b) in increasing order of η_f. We consider η_f as the final decision criterion to decide the rank of any classifiers. Here, we observe that in spite of best η_m value obtained by AEKOC, it slipped to 3^{rd} position as per η_f criterion. According to η_f values, OCSVM emerges as the best classifier.

Further, we compute the training and testing time for all 5 classifiers. The total time taken in training and testing has been presented collectively in Table 3.5 and the average time is plotted in Figure 3.2. KOC and AEKOC consume less average time compared to other classifiers, and this outcome is evident because both use a non-iterative approach to learning.

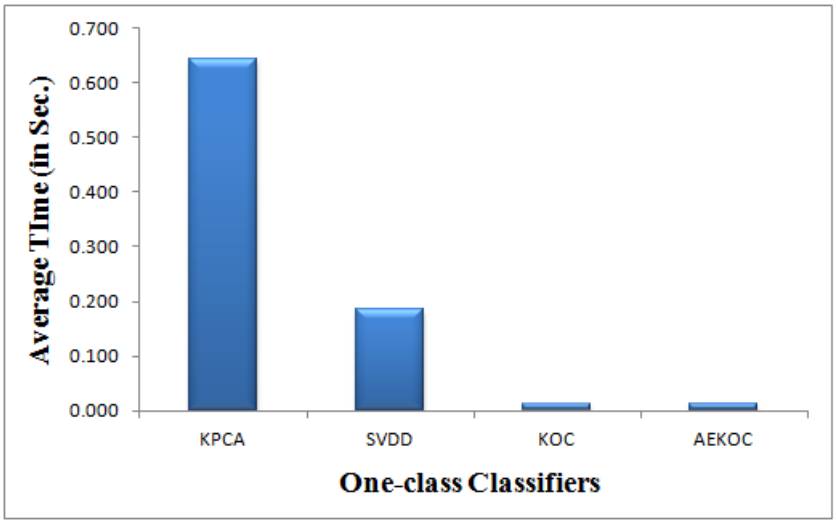


Figure 3.2: Consumed average time by one-class classifiers

Table 3.5: Total time (training time + testing time in seconds) consumed by existing and proposed one-class classifiers

Datasets	KPCA	OCSVM*	SVDD	KOC	AEKOC
Aust(1)	0.119	0.033	0.603	0.011	0.016
Aust(2)	0.037	0.019	0.155	0.003	0.003
Bupa(1)	0.038	0.017	0.124	0.003	0.003
Bupa(2)	0.214	0.019	0.127	0.006	0.006
Ecoli(1)	0.166	0.021	0.201	0.009	0.010
Ecoli(2)	1.343	0.032	0.217	0.019	0.020
Germ(1)	0.532	0.025	0.150	0.014	0.015
Germ(2)	0.138	0.019	0.143	0.006	0.006
Glass(1)	0.186	0.019	0.133	0.006	0.007
Glass(2)	0.612	0.023	0.162	0.011	0.011
Heart(1)	0.303	0.021	0.139	0.008	0.008
Heart(2)	3.075	0.048	0.319	0.035	0.035
Iono(1)	0.788	0.031	0.163	0.017	0.018
Iono(2)	0.580	0.023	0.154	0.010	0.012
Iris(1)	0.777	0.024	0.171	0.014	0.012
Iris(2)	0.612	0.025	0.153	0.011	0.010
Iris(3)	0.697	0.030	0.168	0.011	0.011
Jap(1)	0.139	0.020	0.133	0.006	0.008
Jap(2)	0.122	0.023	0.136	0.006	0.006
Park(1)	2.123	0.039	0.227	0.026	0.028
Park(2)	1.859	0.041	0.231	0.029	0.028
Pima(1)	0.147	0.020	0.130	0.006	0.007
Pima(2)	0.190	0.021	0.140	0.007	0.007

* Here, training times of all classifiers, except OCSVM, are computed on the MATLAB platform. OCSVM has used Mex C++ -compiler in MATLAB. Therefore, it consumes lesser time compared to SVDD. Generally, OCSVM and SVDD consume a similar time on the same platform.

3.3 Summary

This chapter has proposed a reconstruction framework-based one-class classifier using a non-iterative approach of learning. This is a single hidden layer-based architecture. Since the proposed classifier has used a non-iterative approach of learning, it consumes less average time compared to the iterative learning-based state-of-the-art one-class classifiers. We have tested on 23 benchmark datasets from various disciplines, and compared outcomes with 3 boundary and 1 reconstruction framework-based one-class classifiers. AEKOC has outperformed reconstruction framework-based one-class classifier (i.e., KPCA) by a significant amount of more than 4% in terms of η_m. However, AEKOC has yielded slightly better η_m value compared to 3 state-of-the-art boundary framework-based one-class classifiers. Further, when we compared KRR-based one-class classifiers i.e., KOC (boundary framework-based) and AEKOC (reconstruction framework-based), they have yielded best η_g for 13 and 10 datasets, respectively. Both have yielded similar η_m and η_f value. Therefore, it is very difficult to declare any one framework-based one-class classifier as the best classifier.

We can further improve the performance of the KRR-based one-class classifier by taking a cue from the above discussion. We can combine the concept of boundary and reconstruction framework in a single architecture, and this architecture can be obtained by developing a multi-layer architecture. This multi-layer architecture is explored for OCC in the next chapter.

Chapter 4

Multi-layer Kernel Ridge Regression for One-class Classification

In the previous chapters, KRR-based two one-class classifiers are discussed. One is boundary framework-based, and the other is a reconstruction framework-based classifier. The boundary framework is good in describing classification boundaries based on the structure of the dataset, and the reconstruction framework is good in representing the data in a better way. This chapter combines both frameworks (boundary and reconstruction) in sequential order and develop a multi-layer method. In this way, proposed multi-layer method gets benefited from both frameworks. It is discussed in detail in Section 4.1.

4.1 Multi-layer Kernel Ridge Regression for One-class Classification:MKOC

In this section, first, we discuss all preliminaries in Section 4.1.1, which is required to discuss the proposed method MKOC. Then, a proposed method MKOC is presented in Section 4.1.2. Further, two types of threshold criteria are discussed in Section 4.1.3.

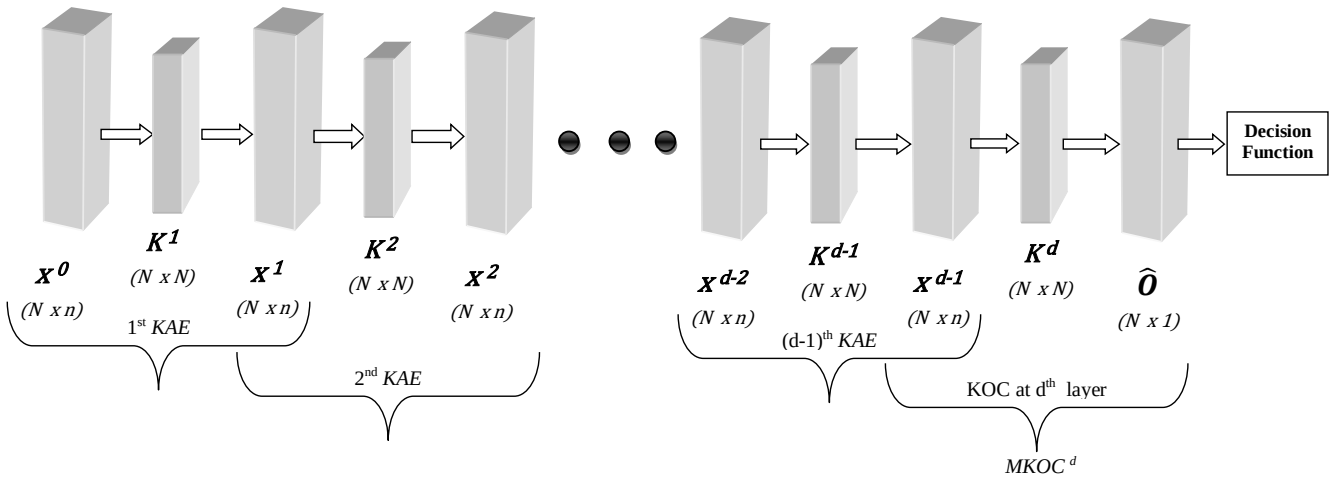


Figure 4.1: A schematic diagram of $\boldsymbol{MKOC}$

4.1.1 Preliminaries

In this section, we provide some notations based on the schematic diagram of MKOC, as shown in Figure 4.1. In MKOC, stacked kernelized Auto-Encoders (KAEs) are employed for defining the successive data representation. In the 1^{st} KAE of Figure 4.1, input training matrix is denoted by $\boldsymbol{X} = \boldsymbol{X^0} = \{\boldsymbol{x_i^0}\}$, where $\boldsymbol{x_i^0} = [x_{i1}^0, x_{i2}^0, ..., x_{in}^0]$, $i = 1, 2, ..., N$, is the n-dimensional input vector of the i^{th} training sample. Let us assume that there are d layers in the proposed architecture, i.e., $h = 1, 2, ..., d$. The output of the h^{th} layer is passed as input to the $(h+1)^{th}$ layer. Let us denote output at h^{th} layer of Auto-Encoder, $\boldsymbol{X^h} = \{\boldsymbol{x_i^h}\}$, where $\boldsymbol{x_i^h} = [x_{i1}^h, x_{i2}^h, ..., x_{in}^h]$, $i = 1, 2, ..., N$. $\boldsymbol{X^h}$ corresponds to the output of the h^{th} Auto-Encoder and the input of the $(h+1)^{th}$ Auto-Encoder. Each of the Auto-Encoders involves a data mapping using function $\phi(.)$, which maps $\boldsymbol{X^{h-1}}$ to non-linear feature space as $\boldsymbol{\Phi^h} = \boldsymbol{\phi(X^{h-1})}$. $\phi(.)$ corresponds to a non-linear feature mapping of $\boldsymbol{X^{h-1}}$ to the corresponding kernel space $\boldsymbol{K^h} = \boldsymbol{(\Phi^h)^T \Phi^h}$, where $\boldsymbol{\Phi^h} = \boldsymbol{[\phi_1^h, \phi_2^h, ..., \phi_N^h]}$. The data representation obtained by calculating the output of the $(d-1)^{th}$ Auto-Encoder in the architecture is passed to the d^{th} layer for OCC using KOC. Two types of training errors are generated by $MKOC$. One of them is generated by the Auto-Encoder until $d-1$ layers and denoted as an error matrix $\boldsymbol{E^h} = \boldsymbol{[e_1^h, e_2^h, \ldots, e_N^h]}$, where $h = 1, 2, ..., (d-1)$. Another one is generated by the one-class classifier at d^{th} layer which is denoted as an

error vector $\boldsymbol{E^d} = [e_1^d, e_2^d, \ldots, e_N^d]$. Based on the above notations, formulations of the proposed method MKOC is discussed in the following section.

4.1.2 Proposed Method MKOC

In this section, a **M**ulti-layer **K**RR-based architecture for **O**ne-class **C**lassification (**MKOC**) is proposed. This proposed multi-layer architecture is constructed by stacking various KRR-based Auto-Encoders (KAEs), followed by a KRR-based one-class classifier (KOC), as shown in Figure 4.1. Here, KAE and KOC belong to reconstruction and boundary framework-based method, respectively. The overall architecture of $MKOC$ is formed by **two processing steps** as follows:

In the **first step**, $(d-1)$ KAEs are trained. Each KAE defines a pair $(\boldsymbol{X^h}, \boldsymbol{\beta_a^h})$, and is stacked in a hierarchical manner. Here, $\boldsymbol{\beta_a^h}$ denotes weight matrix of the h^{th} Auto-Encoder. The h^{th} KAE minimizes the following criterion, which involves a non-linear[1] feature mapping $\boldsymbol{X^{h-1}} \rightarrow \boldsymbol{\Phi^h}$:

$$\begin{aligned} \underset{\boldsymbol{\beta_a^h}, \boldsymbol{e_i^h}}{\text{Minimize}} : \pounds_{KAE} &= \frac{1}{2} \left\| \boldsymbol{\beta_a^h} \right\|_F^2 + \frac{C}{2} \sum_{i=1}^{N} \left\| \boldsymbol{e_i^h} \right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{\beta_a^h})^{\boldsymbol{T}} \boldsymbol{\phi_i^h} = (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{e_i^h})^{\boldsymbol{T}}, \ i = 1, 2, ..., N, \end{aligned} \tag{4.1}$$

where C is a regularization parameter, and $\boldsymbol{e_i^h}$ is a training error vector corresponding to the i^{th} training sample at h^{th} layer. Based on the Representer Theorem [103], we express $\boldsymbol{\beta_a^h}$ as a linear combination of the training data representation $\boldsymbol{\Phi^h}$ and a reconstruction weight matrix $\boldsymbol{W_a^h}$:

$$\boldsymbol{\beta_a^h} = \boldsymbol{\Phi^h} \boldsymbol{W_a^h}. \tag{4.2}$$

[1]Linear case can be easily derived from (4.1) by substituting $\boldsymbol{\phi_i^h} \rightarrow \boldsymbol{x_i^{h-1}}$

By using (4.2), minimization criterion in (4.1) is reformulated as follows:

$$\begin{aligned} \underset{\boldsymbol{W_a^h}, \boldsymbol{e_i^h}}{\text{Minimize}} : \pounds_{KAE} &= \frac{1}{2} Tr\Big((\boldsymbol{W_a^h})^{\boldsymbol{T}} (\boldsymbol{\Phi^h})^{\boldsymbol{T}} \boldsymbol{\Phi^h} \boldsymbol{W_a^h}\Big) + \frac{C}{2} \sum_{i=1}^{N} \left\| \boldsymbol{e_i^h} \right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{W_a^h})^{\boldsymbol{T}} (\boldsymbol{\Phi^h})^{\boldsymbol{T}} \boldsymbol{\phi_i^h} = (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{e_i^h})^{\boldsymbol{T}}, \ i = 1, 2, ..., N. \end{aligned} \tag{4.3}$$

Here, 'Tr' represents trace of a matrix. Further we substitute $\boldsymbol{K^h} = (\boldsymbol{\Phi^h})^{\boldsymbol{T}} \boldsymbol{\Phi^h}$, and $\boldsymbol{k_i^h} = (\boldsymbol{\Phi^h})^{\boldsymbol{T}} \boldsymbol{\phi_i^h}$ (where the individual elements of $\boldsymbol{k_i^h}$ equal to $\boldsymbol{k_{ij}^h} = (\boldsymbol{\phi_i^h})^{\boldsymbol{T}} \boldsymbol{\phi_j^h}, j = 1, 2, \ldots, N$) in (4.3). Now, the optimization problem in (4.3) is written as:

$$\begin{aligned} \underset{\boldsymbol{W_a^h}, \boldsymbol{e_i^h}}{\text{Minimize}} : \pounds_{KAE} &= \frac{1}{2} Tr\Big((\boldsymbol{W_a^h})^{\boldsymbol{T}} \boldsymbol{K^h} \boldsymbol{W_a^h}\Big) + \frac{C}{2} \sum_{i=1}^{N} \left\| \boldsymbol{e_i^h} \right\|_2^2, \\ \text{Subject to} : \ & (\boldsymbol{W_a^h})^{\boldsymbol{T}} \boldsymbol{k_i^h} = (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{e_i^h})^{\boldsymbol{T}}, \ i = 1, 2, ..., N. \end{aligned} \tag{4.4}$$

The Lagrangian relaxation of (4.4) is given below:

$$\pounds_{KAE} = \frac{1}{2} Tr\Big((\boldsymbol{W_a^h})^{\boldsymbol{T}} \boldsymbol{K^h} \boldsymbol{W_a^h}\Big) + \frac{C}{2} \sum_{i=1}^{N} \left\| \boldsymbol{e_i^h} \right\|_2^2 - \sum_{i=1}^{N} \boldsymbol{\alpha_i^h} ((\boldsymbol{W_a^h})^{\boldsymbol{T}} \boldsymbol{k_i^h} - (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} + (\boldsymbol{e_i^h})^{\boldsymbol{T}}), \tag{4.5}$$

where $\boldsymbol{\alpha} = \boldsymbol{\{\alpha_i^h\}}, i = 1, 2 \ldots N$, is a Lagrangian multiplier. In order to optimize $\pounds_{KAE}$, we compute its derivatives as follows:

$$\frac{\partial \pounds_{KAE}}{\partial \boldsymbol{W_a^h}} = 0 \Rightarrow \boldsymbol{W_a^h} = \boldsymbol{\alpha}, \tag{4.6}$$

$$\frac{\partial \pounds_{KAE}}{\partial \boldsymbol{e_i^h}} = 0 \Rightarrow \boldsymbol{E^h} = \frac{1}{C} \boldsymbol{\alpha}, \tag{4.7}$$

$$\frac{\partial \pounds_{KAE}}{\partial \boldsymbol{\alpha_i^h}} = 0 \Rightarrow (\boldsymbol{W_a^h})^{\boldsymbol{T}} \boldsymbol{K^h} = (\boldsymbol{X^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{E^h})^{\boldsymbol{T}}. \tag{4.8}$$

The matrix $\boldsymbol{W_a^h}$ is obtained by substituting (4.7) and (4.8) into (4.6) as follows:

$$\boldsymbol{W_a^h} = \left(\boldsymbol{K^h} + \frac{\boldsymbol{I}}{C} \right)^{-1} \boldsymbol{X^{h-1}}. \tag{4.9}$$

Now, $\boldsymbol{\beta_a^h}$ is derived by substituting (4.9) into (4.2):

$$\boldsymbol{\beta_a^h} = \boldsymbol{\Phi^h}\left(\boldsymbol{K^h} + \frac{\boldsymbol{I}}{C}\right)^{-1}\boldsymbol{X^{h-1}}. \tag{4.10}$$

Hence, the transformed data $\boldsymbol{X^h}$ by the h^{th} KAE can be obtained as follows:

$$\boldsymbol{X^h} = (\boldsymbol{\Phi^h})^{\boldsymbol{T}}\boldsymbol{\beta_a^h} = (\boldsymbol{\Phi^h})^{\boldsymbol{T}}\boldsymbol{\Phi^h}\boldsymbol{W_a^h} = \boldsymbol{K^h}\boldsymbol{W_a^h}. \tag{4.11}$$

where $\boldsymbol{K^h} \in \mathbb{R}^{N\times N}$ is the kernel matrix for the h^{th} layer. After mapping the training data through the $(d-1)$ successive KAEs **in the first step**, the training data representations defined by the outputs of the $(d-1)^{th}$ KAE are used in order to train a KOC at d^{th} layer **in the second step**. The second step represents the last layer (i.e., d^{th} layer) of MKOC, i.e., MKOC^d. It involves a nonlinear feature mapping $\boldsymbol{X^{d-1}} \rightarrow \boldsymbol{\Phi^d}$ and is trained by solving the following optimization problem:

$$\begin{aligned} \underset{\boldsymbol{\beta_o^d}, e_i^d}{\text{Minimize}} : \mathcal{L}_{MKOC^d} &= \frac{1}{2}\left\|\boldsymbol{\beta_o^d}\right\|^2 + \frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2 \\ \text{Subject to} : \ (\boldsymbol{\beta_o^d})^{\boldsymbol{T}}\boldsymbol{\phi_i^d} &= r - e_i^d, \ i = 1, 2, ..., N, \end{aligned} \tag{4.12}$$

where e_i^d is training error corresponding to i^{th} training sample, $\boldsymbol{\beta_o^d}$ denotes weight vector at d^{th} layer, and r is any real number. We set r at equal to 1. By using Representer Theorem [103], $\boldsymbol{\beta_o^d}$ is expressed as a linear combination of the training data representation $\boldsymbol{\Phi^d}$ and reconstruction **weight vector** $\boldsymbol{W_o^d}$:

$$\boldsymbol{\beta_o^d} = \boldsymbol{\Phi^d}\boldsymbol{W_o^d}. \tag{4.13}$$

By using (4.13), the minimization criterion in (4.12) is reformulated as follows:

$$\begin{aligned} \underset{\boldsymbol{W_o^d}, e_i^d}{\text{Minimize}} : \mathcal{L}_{MKOC^d} &= \frac{1}{2}(\boldsymbol{W_o^d})^{\boldsymbol{T}}(\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\Phi^d}\boldsymbol{W_o^d} + \frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2 \\ \text{Subject to} : \ (\boldsymbol{W_o^d})^{\boldsymbol{T}}(\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\phi_i^d} &= r - e_i^d, \ i = 1, 2, ..., N. \end{aligned} \tag{4.14}$$

Further, we substitute $\boldsymbol{K^d} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\Phi^d}$, and $\boldsymbol{k_i^d} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\phi_i^d}$ (where the individual elements of $\boldsymbol{k_i^d}$ equal to $\boldsymbol{k_{ij}^d} = (\boldsymbol{\phi_i^d})^{\boldsymbol{T}}\boldsymbol{\phi_j^d}, j = 1, 2, \ldots, N$) in (4.14). Now, the optimization problem in (4.14) is reformulated as follows:

$$\begin{aligned} \underset{\boldsymbol{W_o^d}, e_i^d}{\text{Minimize}} : \pounds_{MKOC^d} &= \frac{1}{2}(\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{K^d}\boldsymbol{W_o^d} + \frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{k_i^d} = r - e_i^d, \ i = 1, 2, ..., N. \end{aligned} \tag{4.15}$$

The Lagrangian relaxation of (4.15) is given below:

$$\pounds_{MKOC^d} = \frac{1}{2}(\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{K^d}\boldsymbol{W_o^d} + \frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2 - \sum_{i=1}^{N}\boldsymbol{\alpha_i^d}((\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{k_i^d} - r + e_i^d), \tag{4.16}$$

where $\boldsymbol{\alpha} = \{\boldsymbol{\alpha_i^d}\}, i = 1, 2 \ldots N$, is a Lagrangian multiplier. In order to optimize (4.16), we compute its derivatives as follows:

$$\frac{\partial \pounds_{MKOC^d}}{\partial \boldsymbol{W_o^d}} = 0 \Rightarrow \boldsymbol{W_o^d} = \boldsymbol{\alpha}, \tag{4.17}$$

$$\frac{\partial \pounds_{MKOC^d}}{\partial \boldsymbol{e_i^d}} = 0 \Rightarrow \boldsymbol{E^d} = \frac{1}{C}\boldsymbol{\alpha}, \tag{4.18}$$

$$\frac{\partial \pounds_{MKOC^d}}{\partial \boldsymbol{\alpha_i^d}} = 0 \Rightarrow (\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{K^d} = \boldsymbol{r} - \boldsymbol{E^d}. \tag{4.19}$$

The weight vector $\boldsymbol{W_o^d}$ of d^{th} layer (i.e., last layer) is, thus, obtained by substituting (4.18) and (4.19) into (4.17), and is given by:

$$\boldsymbol{W_o^d} = \left(\boldsymbol{K^d} + \frac{\boldsymbol{I}}{C}\right)^{-1}\boldsymbol{r}. \tag{4.20}$$

$\boldsymbol{\beta_o^d}$ is derived by substituting (4.20) in (4.13):

$$\boldsymbol{\beta_o^d} = \boldsymbol{\Phi^d}\left(\boldsymbol{K^d} + \frac{\boldsymbol{I}}{C}\right)^{-1}\boldsymbol{r}, \tag{4.21}$$

where $\boldsymbol{r}$ is a vector having all elements equal to r. Since the value r can be arbitrary, we set it equal to $r = 1$.

The predicted output of the final layer (i.e., d^{th} layer) of MKOC (for training samples) can be calculated as follows:

$$\widehat{\boldsymbol{O}} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\beta_o^d} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\Phi^d}\boldsymbol{W_o^d} = \boldsymbol{K^d}\boldsymbol{W_o^d}, \tag{4.22}$$

where $\widehat{\boldsymbol{O}}$ is the predicted output of training data.

After completing the training process and getting the predicted output of the training data, a threshold is required to decide whether any sample is an outlier or not. Two types of threshold criteria ($\theta 1$ and $\theta 2$) are discussed in the next section.

4.1.3 Decision Function

Two types of thresholds namely, $\theta 1$ and $\theta 2$, are employed with the proposed method, which are determined as follows:

(i) **For $\theta 1$:**

Step I We calculate the distance between the predicted value of the i^{th} training sample and r, and store in a vector, $\boldsymbol{\Lambda} = \{\Lambda_i\}$ and $i = 1, 2, ..., N$, as follows:

$$\Lambda_i = \left|\widehat{O}_i - r\right|. \tag{4.23}$$

Step II After storing all distances in $\boldsymbol{\Lambda}$ as per (4.23), we sort these distances in decreasing order and denoted by a vector $\boldsymbol{\Lambda_{dec}}$. Further, we reject a few percents of training samples based on the deviation. Most deviated samples are rejected because they are most probably far from the distribution of the target data. The threshold is decided based on these deviations as follows:

$$\theta 1 = \Lambda_{dec}(\lfloor \nu * N \rfloor), \tag{4.24}$$

where $0 < \nu \leq 1$ is the fraction of rejection of training samples for deciding threshold value. N is the number of training samples and $\lfloor\ \rfloor$

denotes floor operation.

(ii) **For $\theta 2$:** We select threshold ($\theta 2$) as a small fraction of the mean of the predicted output:

$$\theta 2 = (\left\lfloor \nu * \text{mean}(\widehat{\boldsymbol{O}}) \right\rfloor), \tag{4.25}$$

where $0 < \nu \leq 1$ is the fraction of rejection for deciding threshold value.

The training process is completed after getting a threshold value by either using $\theta 1$ or $\theta 2$. Afterward, during testing, a test vector $\boldsymbol{x_p}$ is fed to the trained multi-layer architecture and its output $\widehat{O}_p$ is obtained. Further, compute $\widehat{\Lambda}_p$ for both types of threshold as follows:

For $\theta 1$, calculate the distance ($\widehat{\Lambda}_p$) between the predicted value $\widehat{O}_p$ of the p^{th} testing sample and r:

$$\widehat{\Lambda}_p = \left| \widehat{O}_p - r \right|. \tag{4.26}$$

For $\theta 2$, calculate the distance ($\widehat{\Lambda}_p$) between the predicted value $\widehat{O}_p$ of the p^{th} testing sample and mean of the predicted values obtained after training as follows:

$$\widehat{\Lambda}_p = \left| \widehat{O}_p - \text{mean}(\widehat{\boldsymbol{O}}) \right|. \tag{4.27}$$

Finally, $\boldsymbol{x_p}$ is classified based on the following rule:

$$\boldsymbol{x_p} \text{ belongs to } \begin{cases} \text{Target class}, & \text{If } \widehat{\Lambda}_p \leq \text{Threshold} \\ \text{Outlier}, & \text{otherwise.} \end{cases} \tag{4.28}$$

The complete training steps followed by $MKOC$ are described in Algorithm 4.1.

Overall, the proposed multi-layer OCC architecture creates two variants of MKOC using two types of threshold criteria (viz., $\theta 1$ and $\theta 2$), i.e., MKOC_$\theta 1$ and MKOC_$\theta 2$.

4.2 Experiments

In this chapter, we follow the same experimental setup and datasets, as mentioned in Section 3.2 of Chapter 3. In order to compare our proposed methods viz., AEKOC

Algorithm 4.1 Multi-layer KRR-based architecture for OCC: MKOC

Input: Training set $\boldsymbol{X}$, regularization parameter (C), non-linear feature map ($\boldsymbol{\Phi}$), number of layers (d)

Output: Target class or Outlier

Initially, $\boldsymbol{X^0 = X}$

for $h = 1$ to d **do**

  First Phase:

  if $h < d$ **then**

    Pass $\boldsymbol{X^{h-1}}$ as an input to h^{th} KAE.

    Train the KAE as per (4.4).

    Compute $(\boldsymbol{X^h, \beta_a^h})$ during training of h^{th} KAE.

    Transformed output $\boldsymbol{X^h}$ is computed from the current layer and pass it as the input to the next layer, i.e., $(h+1)^{th}$, in the hierarchy.

    Second Phase:

  else

    Train final layer, i.e., d^{th} layer, for one-class classification.

    Output of $(d-1)^{th}$ Auto-Encoder is passed as an input to the one-class classifier at d^{th} layer.

    Train the d^{th} layer by $MKOC^d$ as per (4.15).

  end if

end for

Compute a threshold value either by using $\theta 1$ (4.24) or $\theta 2$ (4.25).

At a final step, whether a new input is outlier or not, decides based on the rule discussed in (4.28).

(Chapter 3), and MKOC, we have conducted experimentation on various existing state-of-the-art kernel-based OCC methods, such as OCSVM [1], SVDD [1], KPCA [4], and KOC [7]. Overall, performance among 7 variants of one-class classifier is presented next.

Performance of 7 variants of one-class classifiers in terms of η_g are presented in Table 4.1. After analyzing this table, it is observed that MKOC_θ1 and MKOC_θ2, each yields the best results for 6 out of 23 datasets. Table 4.2 shows that number of datasets with the best η_g value for each classifier. Proposed classifiers MKOC_θ1 and MKOC_θ2 attain top positions in Table 4.2. Proposed classifier from the previous chapter (i.e., AEKOC) still yields the best result for 4 datasets. Overall, these three proposed classifiers collectively yield the best results for 16 out of 23 datasets. As per discussion in Chapter 3, we compute three performance criteria [167] for further analysis: (i) average of η_g (η_m) (ii) Friedman test (F-score and p-value) (iii) Friedman rank (η_f). We analyze the performance of 7 variants of different classifiers based on these three criteria as follows:

(i) We compute η_m for analyzing the combined performance of the classifier. η_m value of each classifier is available in the last row of Table 4.1. It is also provided in Table 4.3(a) in decreasing order of η_m. When we compare η_m values; then it is observed that MKOC[2] performs better than 4 existing and 1 proposed classifiers in spite of obtaining the best η_g for 12 datasets only. As discussed in Chapter 3, AEKOC only slightly yields better η_m compared to other existing classifiers. However, MKOC_θ1 yields at least 1.56% more η_m for all datasets compared to AEKOC and 4 state-of-the-art existing classifiers.

(ii) Although both MKOC[2] variants attain top positions among 7 variants of classifiers in Table 4.3(a); however, we need to verify the outcomes of the proposed and existing methods statistically. For this purpose, we conduct a non-parametric Friedman test [166] similar to discussed in Section 3.2 of Chapter 3. Friedman test mainly computes three components viz., p-value, F-score, and critical value.

[2] Wherever we mention the name MKOC, it collectively represents MKOC_θ1 and MKOC_θ2.

Table 4.1: Performance in terms of η_g for 23 datasets

Datasets	KPCA	OCSVM	SVDD	KOC	AEKOC	MKOC_θ1	MKOC_θ2
Aust(1)	63.69	66.08	65.55	65.07	**72.88**	72.12	72.43
Aust(2)	73.06	76.59	76.78	74.21	77.96	79.89	**80.31**
Bupa(1)	**62.91**	60.64	60.64	57.09	56.19	62.55	62.19
Bupa(2)	**74.28**	69.78	69.75	68.81	68.31	73.56	73.61
Ecoli(1)	65.79	89.43	**89.49**	89.38	89.00	82.51	80.36
Ecoli(2)	72.30	79.42	78.87	82.32	79.16	**84.42**	81.76
Germ(1)	80.77	80.34	81.10	73.17	74.04	74.10	**82.79**
Germ(2)	49.75	52.80	52.77	53.41	51.57	**54.46**	49.59
Glass(1)	57.69	59.61	59.61	58.91	59.29	**62.46**	59.23
Glass(2)	**77.65**	73.32	72.89	73.08	73.22	77.15	75.69
Heart(1)	70.42	72.91	72.91	65.03	67.99	71.72	**73.93**
Heart(2)	63.50	64.90	64.90	66.39	61.15	**68.89**	59.51
Iono(1)	76.54	**93.13**	**93.13**	92.69	92.95	89.54	88.77
Iono(2)	57.10	44.63	44.63	53.40	41.04	**67.70**	60.78
Iris(1)	96.44	85.06	84.12	92.35	92.79	**99.59**	93.87
Iris(2)	72.99	81.70	81.70	85.59	**88.37**	77.20	74.07
Iris(3)	69.29	83.18	82.67	83.40	**83.74**	71.29	69.82
Jap(1)	64.09	71.45	70.15	67.33	**76.23**	74.45	74.66
Jap(2)	72.29	75.78	76.58	73.48	78.27	80.14	**80.55**
Park(1)	70.20	95.71	**97.07**	96.15	96.67	91.48	91.70
Park(2)	67.79	81.82	79.77	**90.74**	83.78	80.97	79.65
Pima(1)	77.98	79.18	79.21	79.04	78.66	79.21	**80.50**
Pima(2)	57.05	56.59	56.71	54.78	54.01	57.70	**57.80**
Mean of all η_g (η_m)	69.28	73.65	73.52	73.73	73.79	**75.35**	**74.07**

Table 4.2: Number of datasets for which each one-class classifier yields the best η_g

One-class Classifiers	Number of datasets with the best η_g value
MKOC_θ1	6.00
MKOC_θ2	6.00
AEKOC	4.00
KPCA	3.00
SVDD	3.00
OCSVM	1.00
KOC	1.00

Table 4.3: Friedman Rank (η_f) and mean of η_g (η_m)

4.3(a) In decreasing order of η_m

	η_f	η_m
MKOC_θ1	**2.91**	**75.35**
MKOC_θ2	**3.48**	**74.07**
AEKOC	4.17	73.79
KOC	4.48	73.73
OCSVM	3.80	73.65
SVDD	3.89	73.52
KPCA	5.26	69.28

4.3(b) In increasing order of η_f

	η_f	η_m
MKOC_θ1	**2.91**	**75.35**
MKOC_θ2	**3.48**	**74.07**
OCSVM	3.80	73.65
SVDD	3.89	73.52
AEKOC	4.17	73.79
KOC	4.48	73.73
KPCA	5.26	69.28

Table 4.4: Total time (training time + testing time in seconds) consumed by existing and proposed one-class classifiers

Datasets	KPCA	OCSVM*	SVDD	KOC	AEKOC	MKOC
Aust(1)	0.119	0.033	0.603	0.011	0.016	0.006
Aust(2)	0.037	0.019	0.155	0.003	0.003	0.005
Bupa(1)	0.038	0.017	0.124	0.003	0.003	0.005
Bupa(2)	0.214	0.019	0.127	0.006	0.006	0.029
Ecoli(1)	0.166	0.021	0.201	0.009	0.010	0.013
Ecoli(2)	1.343	0.032	0.217	0.019	0.020	0.143
Germ(1)	0.532	0.025	0.150	0.014	0.015	0.041
Germ(2)	0.138	0.019	0.143	0.006	0.006	0.014
Glass(1)	0.186	0.019	0.133	0.006	0.007	0.022
Glass(2)	0.612	0.023	0.162	0.011	0.011	0.069
Heart(1)	0.303	0.021	0.139	0.008	0.008	0.027
Heart(2)	3.075	0.048	0.319	0.035	0.035	0.313
Iono(1)	0.788	0.031	0.163	0.017	0.018	0.058
Iono(2)	0.580	0.023	0.154	0.010	0.012	0.057
Iris(1)	0.777	0.024	0.171	0.014	0.012	0.083
Iris(2)	0.612	0.025	0.153	0.011	0.010	0.051
Iris(3)	0.697	0.030	0.168	0.011	0.011	0.076
Jap(1)	0.139	0.020	0.133	0.006	0.008	0.019
Jap(2)	0.122	0.023	0.136	0.006	0.006	0.016
Park(1)	2.123	0.039	0.227	0.026	0.028	0.159
Park(2)	1.859	0.041	0.231	0.029	0.028	0.163
Pima(1)	0.147	0.020	0.130	0.006	0.007	0.016
Pima(2)	0.190	0.021	0.140	0.007	0.007	0.021
Average time	0.643	0.026	0.186	0.012	0.012	0.061

* Here, training times of all classifiers, except OCSVM, are computed on MATLAB platform. OCSVM has used Mex C++ -compiler in MATLAB. Therefore, it consumes lesser time compared to SVDD. However, OCSVM and SVDD consumes similar amount of average time on the same platform.

The computed p-value, F-score value and critical value are 0.01, 16.52 and 12.59, respectively. Since the computed F-score value is higher than the critical value, and the p-value is lesser than the tolerance level 0.05, we reject the null hypothesis with 95% of confidence. Therefore, it is concluded that the outcomes presented in this chapter are statistically significant.

(iii) Further, we compute Friedman Rank (η_f) [166] for each classifier as discussed in Section 2.8 of Chapter 2. We consider η_f as the final decision criteria to decide the rank of any classifiers. In Table 4.3(b), we present η_f along with η_m of each classifier in increasing order of η_f. In Table 4.3(b), KRR-based single hidden layer-based methods (KOC and AEKOC) do not obtain better η_f in spite of obtaining better η_m compared to the OCSVM, SVDD, and KPCA. However, both multi-layer variants, MKOC_θ1 and MKOC_θ2, attain top position among 7 variants of classifiers in terms of both criteria i.e., η_f and η_m. Although, MKOC_θ1 and MKOC_θ2 yield the best results for the same number of datasets (see Table 4.2), MKOC_θ1 outperforms MKOC_θ2 in terms of η_m and η_f.

Further, we compute the training and testing time for these classifiers. The total time taken in training and testing has been presented collectively in Table 4.4. The average time is presented in the last row of this table. MKOC^2 consumes more average time compared to KRR-based single hidden layer-based methods (KOC and AEKOC). It is due to its multi-layer architecture. Despite the multi-layer architecture of MKOC, it consumes significantly less average time compared to KPCA and SVDD due to its non-iterative approach of learning. In last, we need to discuss one crucial aspect of multi-layer architecture, i.e., number of layers. We have experimented with 5 layers. However, we observe that the performance of the classifier either slightly improves sometimes or mostly degraded after 3^{rd} layer. Therefore, the number of layers is empirically decided, and all presented results in this and next chapter of this thesis use 3-layered architecture.

4.3 Summary

This chapter has proposed a multi-layer architecture for OCC, which follows a non-iterative approach of learning. It has combined the concept of two different frameworks i.e., boundary and reconstruction. We have used two types of threshold criteria and proposed two variants, namely MKOC_θ1 and MKOC_θ2. MKOC also consumes less average time compared to KPCA and SVDD in spite of multi-layer architecture. However, MKOC consumes more time compared to KOC and AEKOC due to its multi-layer architecture. MKOC variants have achieved better η_m and η_f compared to all mentioned one-class classifiers in this chapter. Conversely, proposed classifier AEKOC was unable to achieve the same in Chapter 3. MKOC_θ1 obtained better η_m and η_f compared to MKOC_θ2 and emerged as a top performer among all mentioned methods. We have also statistically verified that outcomes presented in this chapter are statistically significant with 95% of confidence.

We have observed that Auto-Encoder has helped the proposed methods (AEKOC and MKOC) in getting better performance by providing a better representation of the data. However, it does not consider structural information of the data. Therefore, structural information does not play any role in deciding the boundary of the proposed classifiers. The structural information has well-representation power, which can enhance the performance of the classifier. It is explored in the next chapter, where we combine the concept of structural information with MKOC.

Chapter 5

Graph-Embedded Multi-layer Kernel Ridge Regression for One-class Classification

In the previous chapter, a KRR-based multi-layer one-class classifier is proposed, which utilizes representation learning in its architecture. However, this classifier does not utilize the structural information of the data. Structural information has well-representation power and can be utilized with any machine learning methods using the Graph-Embedding approach [119, 121]. Structural information can be obtained by using a Laplacian graph, as discussed in Section 2.4 of Chapter 2. In the current chapter, the proposed classifier MKOC (discussed in Chapter 4) is extended by embedding structural information within it, and referred to as Graph-Embedded MKOC (GMKOC). The proposed GMKOC also combines two frameworks (boundary and reconstruction) in sequential order, and develops a Graph-Embedded multi-layer architecture. During the construction of GMKOC, we propose a novel Graph-Embedded KRR-based Auto-Encoder, which provides a better representation of data. It helps GMKOC to achieve better performance compared to existing KRR-based with and without Graph-Embedded methods. The proposed method is discussed in detail in the next section.

5.1 Graph-Embedding with Multi-layer Kernel Ridge Regression for One-class Classification:GMKOC

In this section, first, we discuss all preliminariess in Section 5.1.1, which is required to discuss the proposed method GMKOC. Then, a proposed method GMKOC is presented in Section 5.1.2. Further, two types of threshold criteria are discussed in Section 5.1.3.

5.1.1 Preliminaries

The proposed GMKOC is a multi-layer architecture, which is shown in Figure 5.1. In the 1^{st} Graph-Embedded KRR-based Auto-Encoder (GKAE) of this figure, input training matrix is denoted by $\boldsymbol{X = X^0 = \{x_i^0\}}$, where $\boldsymbol{x_i^0} = [x_{i1}^0, x_{i2}^0, ..., x_{in}^0]$, $i = 1, 2, ..., N$, is the n-dimensional input vector of the i^{th} training sample. Let us assume that there are d layers in the proposed architecture, i.e., $h = 1, 2, ..., d$. Output of the h^{th} layer is passed as input to the $(h+1)^{th}$ layer. Let us denote output at h^{th} layer of Auto-Encoder, $\boldsymbol{X^h} = \{\boldsymbol{x_i^h}\}$, where $\boldsymbol{x_i^h} = [x_{i1}^h, x_{i2}^h, ..., x_{in}^h]$, $i = 1, 2, ..., N$. $\boldsymbol{X^h}$ corresponds to the output of the h^{th} Auto-Encoder and the input of the $(h+1)^{th}$ Auto-Encoder. Each of the Auto-Encoders involves a data mapping using function $\phi(.)$, which maps $\boldsymbol{X^{h-1}}$ to non-linear feature space as $\boldsymbol{\Phi^h = \phi(X^{h-1})}$. $\phi(.)$ corresponds to a non-linear feature mapping of $\boldsymbol{X^{h-1}}$ to the corresponding kernel space $\boldsymbol{K^h = (\Phi^h)^T \Phi^h}$, where $\boldsymbol{\Phi^h} = \left[\boldsymbol{\phi_1^h, \phi_2^h, ..., \phi_N^h}\right]$. The data representation obtained by calculating the output of the $(d-1)^{th}$ Auto-Encoder in the architecture is passed to the d^{th} layer for OCC using Graph-Embedded KOC. The d^{th} layer (i.e., last layer) of GMKOC is denoted as GMKOC^d. In Figure 5.1, Graph-Embedding is performed by using a scattered matrix $\boldsymbol{S^h}$, which encodes the variance information with the kernel matrix. Here, $\boldsymbol{S^h}$ denotes scattered matrix of h^{th} layer. Two types of training errors and weight matrices are generated by GMKOC. The first type of training error matrix and weight matrix are generated by the h^{th} Auto-Encoder until $(d-1)$ layers and

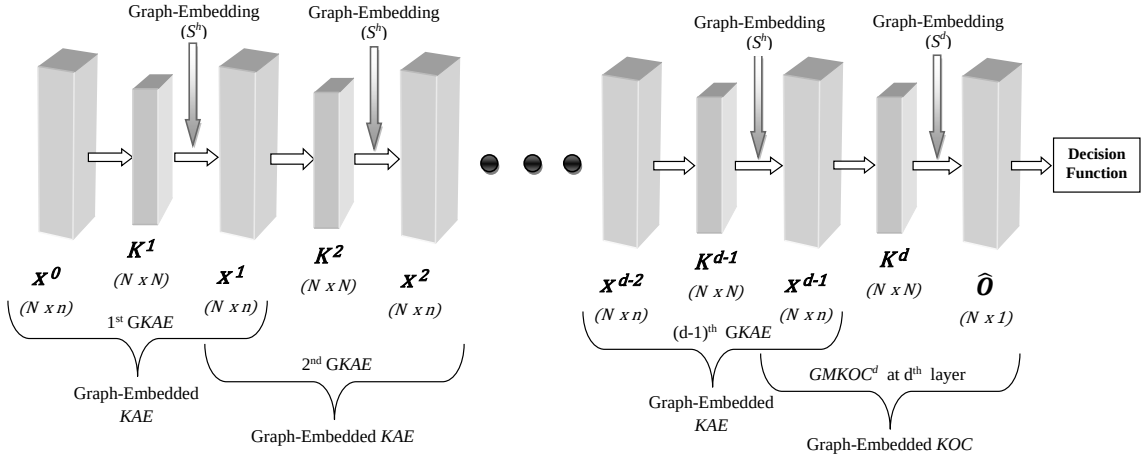


Figure 5.1: A schematic diagram of Graph-Embedded multi-layer KRR-based architecture for one-class classification

denoted as $\boldsymbol{E^h} = [\boldsymbol{e_1^h}, \boldsymbol{e_2^h}, \ldots, \boldsymbol{e_N^h}]$ and $\boldsymbol{\beta_a^h}$, where $h = 1, 2, ..., (d-1)$, respectively. Another type of training error vector and weight vector are generated by the one-class classifier at d^{th} layer and denoted as $\boldsymbol{E^d} = [e_1^d, e_2^d, \ldots, e_N^d]$ and $\boldsymbol{\beta_o^d}$, respectively. Based on the above notations, formulations of the proposed method GMKOC is discussed in the following section.

5.1.2 Proposed Method GMKOC

In this section, a Graph-Embedded multi-layer KRR-based method for OCC (**GMKOC**) is proposed. The multi-layer architecture of the proposed method is constructed by stacking various proposed Graph-Embedded KRR-based Auto-Encoders (GKAEs), followed by a Graph-Embedded KRR-based one-class classifier at the last layer (as shown in Figure 5.1). These stacked Auto-Encoders in GMKOC are employed for defining the successive data representation, and these layers are based on the reconstruction frameworks. The last layer in GMKOC is based on a boundary framework, which is a classification layer. Overall, GMKOC is a combination of reconstruction and boundary framework. It uses different types of Laplacian graph for obtaining structural information of the data. The structural information primarily provides variance information between the samples. The overall architecture of

GMKOC is formed by **two processing steps** as follows:

In the **first step**, $(d-1)$ GKAEs are trained. Each GKAE defines a triplet $(\boldsymbol{X^h}, \boldsymbol{\beta_a^h}, \boldsymbol{S^h})$, and is stacked in a hierarchical manner. GKAE involves non-linear mapping of the input data into kernel feature space, i.e., $\boldsymbol{X^{h-1}} \rightarrow \boldsymbol{\Phi^h}$. Let us first discuss the process of Graph-Embedding[1] in kernel feature space in the following lines; then, we provide the proposed optimization problem for GKAE.

Let us define a graph $\mathcal{G}^h = \left\{\boldsymbol{\Phi^h}, \boldsymbol{V^h}\right\}$ where $\boldsymbol{V^h} \in \mathbb{R}^{N \times N}$ is the weight matrix expressing similarities between the graph nodes $\boldsymbol{\phi_i^h} \in \Phi^h$. The Graph Laplacian matrix of the h^{th} GKAE is calculated by $\boldsymbol{\mathcal{L}^h = D^h - V^h}$, where $\boldsymbol{D^h}$ is a diagonal degree matrix in the h^{th} layer defined as [47]:

$$\boldsymbol{D_{ii}^h} = \sum_{j=1}^{N} \boldsymbol{V_{ij}^h}. \tag{5.1}$$

Any Laplacian Graph can be exploited with GKAE. In our experiments, we have used the fully connected and k-nearest neighbor graphs. These graphs compute the similarity between samples using the heat kernel function as follows:

$$v_{ij}^h = exp\left(-\frac{\left\|\boldsymbol{\phi_i^h} - \boldsymbol{\phi_j^h}\right\|_2^2}{2\sigma^2}\right), \tag{5.2}$$

where σ is a hyper-parameter scaling the square Euclidean distance between $\boldsymbol{\phi_i^h}$ and $\boldsymbol{\phi_j^h}$. In the case of k-nearest neighbor Graph, the weight matrix $\boldsymbol{V^h}$ is defined as follows:

$$V_{ij}^h = \begin{cases} v_{ij}^h, & \text{if } \boldsymbol{\phi_j^h} \in \mathcal{N}_i^h \\ 0, & \text{otherwise,} \end{cases} \tag{5.3}$$

where $\mathcal{N}_i^h$ denotes the neighborhood of $\boldsymbol{\phi_i^h}$. Using the above notations, the scatter matrix $\boldsymbol{S^h}$ encodes the variance information as follows:

$$\boldsymbol{S^h} = \boldsymbol{\Phi^h \mathcal{L}^h (\Phi^h)^T}. \tag{5.4}$$

[1]Graph-Embedding is discussed in detail in Section 2.4 of Chapter 2.

$\boldsymbol{S^h}$ can be expressed as dispersion (scattering/variance) of training data in kernel feature space from their mean [8, 47]:

$$\begin{aligned}\boldsymbol{S^h} &= \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{\phi_i^h} - \overline{\boldsymbol{\Phi^h}})(\boldsymbol{\phi_i^h} - \overline{\boldsymbol{\Phi^h}})^T \\ &= \frac{1}{N}\boldsymbol{\Phi^h}(\boldsymbol{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^T)(\boldsymbol{\Phi^h})^T \\ &= \boldsymbol{\Phi^h}(\frac{1}{N}\boldsymbol{I} - \frac{1}{N^2}\mathbf{1}\mathbf{1}^T)(\boldsymbol{\Phi^h})^T \\ &= \boldsymbol{\Phi^h}\boldsymbol{\mathcal{L}^h}(\boldsymbol{\Phi^h})^T,\end{aligned} \tag{5.5}$$

where $\overline{\boldsymbol{\Phi^h}}$ expresses the mean of training vector in the kernel feature space, $\mathbf{1} \in \Re^N$ is a vector of ones and $\boldsymbol{I} \in \Re^{N\times N}$ is an identity matrix. It is quite clear that $\boldsymbol{S^h}$ can be suppressed within the Graph Embedding Framework with the above expression, where matrices $\boldsymbol{D^h} = \frac{1}{N}\boldsymbol{I}$ and $\boldsymbol{V^h} = \frac{1}{N^2}\mathbf{1}\mathbf{1}^T$.

Based on the above discussion of the Graph-Embedding, we formulate the optimization problem of GKAE. Since the minimization criterion of GKAE is derived by using vanilla KRR-based Auto-Encoder (KAE), we provide formulation of KAE as follows:

$$\begin{aligned}\underset{\boldsymbol{\beta_a^h},\boldsymbol{e_i^h}}{\text{Minimize}} : \pounds_{KAE} &= \frac{1}{2}\left\|\boldsymbol{\beta_a^h}\right\|_F^2 + \frac{C}{2}\sum_{i=1}^{N}\left\|\boldsymbol{e_i^h}\right\|_2^2 \\ \text{Subject to} : \ &(\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h} = (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{e_i^h})^{\boldsymbol{T}},\ i = 1, 2, ..., N,\end{aligned} \tag{5.6}$$

where C is a regularization parameter, and $\boldsymbol{e_i^h}$ is a training error vector corresponding to the i^{th} training sample at h^{th} layer. Based on the minimization criterion in (5.6) and scatter matrix in (5.4), GKAE for h^{th} layer (where $h = 1, 2, \ldots, (d-1)$) is formulated as follows:

$$\begin{aligned}\underset{\boldsymbol{\beta_a^h},\boldsymbol{e_i^h}}{\text{Minimize}} : \pounds_{GKAE} &= \frac{1}{2}Tr\Big((\boldsymbol{\beta_a^h})^{\boldsymbol{T}}(\boldsymbol{S^h} + \lambda\boldsymbol{I})\boldsymbol{\beta_a^h}\Big) + \frac{C}{2}\sum_{i=1}^{N}\left\|\boldsymbol{e_i^h}\right\|_2^2 \\ \text{Subject to} : \ &(\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h} = (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{e_i^h})^{\boldsymbol{T}},\ i = 1, 2, ..., N,\end{aligned} \tag{5.7}$$

where λ represents graph regularization parameter. The optimization problem of

GKAE simultaneously minimizes the training error as well as class compactness. We express (5.7) using (5.5) as follows:

$$\begin{aligned}
\pounds_{GKAE} &= \frac{1}{2}Tr\Big((\boldsymbol{\beta_a^h})^{\boldsymbol{T}}(\boldsymbol{S^h}+\lambda\boldsymbol{I})\boldsymbol{\beta_a^h}\Big) + \frac{C}{2}\sum_{i=1}^{N}\big\|(\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h}\big\|_2^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\Big(((\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h} - (\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\overline{\boldsymbol{\Phi^h}})^T((\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h} - (\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\overline{\boldsymbol{\Phi^h}})\Big) \\
&+ \frac{C}{2}\sum_{i=1}^{N}\big\|(\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h}\big\|_2^2 + \frac{\lambda}{2}Tr\Big((\boldsymbol{\beta_a^h})^T\boldsymbol{\beta_a^h}\Big) \\
&= \frac{1}{N}\sum_{i=1}^{N}\big\|\boldsymbol{o_i^h} - \boldsymbol{o^h}\big\|_2^2 + \frac{C}{2}\sum_{i=1}^{N}\big\|(\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - \boldsymbol{o_i^h}\big\|_2^2 + \frac{\lambda}{2}Tr\Big((\boldsymbol{\beta_a^h})^T\boldsymbol{\beta_a^h}\Big),
\end{aligned} \tag{5.8}$$

where $\boldsymbol{o_i^h} = (\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h}$ and $\boldsymbol{o^h} = (\boldsymbol{\beta_a^h})^{\boldsymbol{T}}\overline{\boldsymbol{\Phi^h}}$. Here, the regularization parameter C provides the trade-off between the two objectives viz., minimizing the training error and class compactness.

Further, based on the Representer Theorem [103], we express $\boldsymbol{\beta_a^h}$ as a linear combination of the training data representation $\boldsymbol{\Phi^h}$ and a reconstruction weight matrix $\boldsymbol{W_a^h}$:

$$\boldsymbol{\beta_a^h} = \boldsymbol{\Phi^h}\boldsymbol{W_a^h}. \tag{5.9}$$

Hence, by using Representer Theorem [103], minimization criterion in (5.7) is reformulated as follows:

$$\begin{aligned}
\underset{\boldsymbol{W_a^h},\boldsymbol{e_i^h}}{\text{Minimize}} : \pounds_{GKAE} &= \frac{1}{2}Tr\Big((\boldsymbol{W_a^h})^{\boldsymbol{T}}(\boldsymbol{\Phi^h})^{\boldsymbol{T}}(\boldsymbol{\Phi^h}\boldsymbol{\mathcal{L}^h}(\boldsymbol{\Phi^h})^{\boldsymbol{T}} \\
&\qquad +\lambda\boldsymbol{I})\boldsymbol{\Phi^h}\boldsymbol{W_a^h}\Big) + \frac{C}{2}\sum_{i=1}^{N}\big\|\boldsymbol{e_i^h}\big\|_2^2, \\
\text{Subject to} : \ & (\boldsymbol{W_a^h})^{\boldsymbol{T}}(\boldsymbol{\Phi^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h} = (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{e_i^h})^{\boldsymbol{T}},\ i = 1, 2, ..., N.
\end{aligned} \tag{5.10}$$

Here, 'Tr' represents trace of a matrix. Further we substitute $\boldsymbol{K^h} = (\boldsymbol{\Phi^h})^{\boldsymbol{T}}\boldsymbol{\Phi^h}$, and $\boldsymbol{k_i^h} = (\boldsymbol{\Phi^h})^{\boldsymbol{T}}\boldsymbol{\phi_i^h}$ (where the individual elements of $\boldsymbol{k_i^h}$ equal to $\boldsymbol{k_{ij}^h} = (\boldsymbol{\phi_i^h})^{\boldsymbol{T}}\boldsymbol{\phi_j^h}, j = 1, 2, \ldots, N$) in (5.10). Now, the optimization problem in (5.10) is written as:

$$\underset{\boldsymbol{W_a^h},\boldsymbol{e_i^h}}{\text{Minimize}} : \pounds_{GKAE} = \frac{1}{2}Tr\Big((\boldsymbol{W_a^h})^{\boldsymbol{T}}(\boldsymbol{K^h}\boldsymbol{\mathcal{L}^h}\boldsymbol{K^h} + \lambda\boldsymbol{K^h})\boldsymbol{W_a^h}\Big) + \frac{C}{2}\sum_{i=1}^{N}\left\|\boldsymbol{e_i^h}\right\|_2^2, \tag{5.11}$$

$$\text{Subject to} : \ (\boldsymbol{W_a^h})^{\boldsymbol{T}}\boldsymbol{k_i^h} = (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{e_i^h})^{\boldsymbol{T}}, \ i = 1, 2, ..., N.$$

The Lagrangian relaxation of (5.11) is given below:

$$\begin{aligned}\pounds_{GKAE} = \frac{1}{2}Tr\Big((\boldsymbol{W_a^h})^{\boldsymbol{T}}(\boldsymbol{K^h}\boldsymbol{\mathcal{L}^h}\boldsymbol{K^h} + \lambda\boldsymbol{K^h})\boldsymbol{W_a^h}\Big) \\ + \frac{C}{2}\sum_{i=1}^{N}\left\|\boldsymbol{e_i^h}\right\|_2^2 - \sum_{i=1}^{N}\boldsymbol{\alpha_i^h}((\boldsymbol{W_a^h})^{\boldsymbol{T}}\boldsymbol{k_i^h} - (\boldsymbol{x_i^{h-1}})^{\boldsymbol{T}} + (\boldsymbol{e_i^h})^{\boldsymbol{T}}),\end{aligned} \tag{5.12}$$

where $\boldsymbol{\alpha^h} = \boldsymbol{\{\alpha_i^h\}}, i = 1, 2 \dots N$, is a Lagrangian multiplier. In order to optimize (5.12), we compute its derivatives as follows:

$$\frac{\partial \pounds_{GKAE}}{\partial \boldsymbol{W_a^h}} = 0 \Rightarrow \boldsymbol{W_a^h} = (\boldsymbol{\mathcal{L}^h}\boldsymbol{K^h} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\alpha^h}, \tag{5.13}$$

$$\frac{\partial \pounds_{GKAE}}{\partial \boldsymbol{e_i^h}} = 0 \Rightarrow \boldsymbol{E^h} = \frac{1}{C}\boldsymbol{\alpha^h}, \tag{5.14}$$

$$\frac{\partial \pounds_{GKAE}}{\partial \boldsymbol{\alpha_i^h}} = 0 \Rightarrow (\boldsymbol{W_a^h})^{\boldsymbol{T}}\boldsymbol{K^h} = (\boldsymbol{X^{h-1}})^{\boldsymbol{T}} - (\boldsymbol{E^h})^{\boldsymbol{T}}. \tag{5.15}$$

The matrix $\boldsymbol{W_a^h}$ is obtained by substituting (5.14) and (5.15) into (5.13) as follows:

$$\boldsymbol{W_a^h} = \left(\boldsymbol{K^h} + \frac{1}{C}\boldsymbol{\mathcal{L}^h}\boldsymbol{K^h} + \frac{\lambda}{C}\boldsymbol{I}\right)^{-1}\boldsymbol{X^{h-1}}. \tag{5.16}$$

Now, $\boldsymbol{\beta_a^h}$ is derived by substituting (5.16) into (5.9):

$$\boldsymbol{\beta_a^h} = \boldsymbol{\Phi^h}\left(\boldsymbol{K^h} + \frac{1}{C}\boldsymbol{\mathcal{L}^h}\boldsymbol{K^h} + \frac{\lambda}{C}\boldsymbol{I}\right)^{-1}\boldsymbol{X^{h-1}}. \tag{5.17}$$

After mapping the training data through the $(d-1)$ successive GKAEs in the **first step**, the training data representations defined by the outputs of the $(d-1)^{th}$ GKAE are used in order to train d^{th} layer (i.e., last layer) of GMKOC (GMKOC^d) in the **second step**. The GMKOC^d involves a nonlinear mapping $\boldsymbol{X^{d-1}} \rightarrow \boldsymbol{\Phi^d}$ and is

trained by solving the following optimization problem:

$$\begin{aligned} \underset{\boldsymbol{\beta_o^d}, e_i^d}{\text{Minimize}} : \mathcal{L}_{GMKOC^d} &= \frac{1}{2}(\boldsymbol{\beta_o^d})^{\boldsymbol{T}}(\boldsymbol{S^d} + \boldsymbol{\lambda I})\boldsymbol{\beta_o^d} + \frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{\beta_o^d})^{\boldsymbol{T}}\boldsymbol{\phi_i^d} = r - e_i^d, \ i = 1, 2, ..., N. \end{aligned} \tag{5.18}$$

By using Representer Theorem [103], $\boldsymbol{\beta_o^d}$ is expressed as a linear combination of the training data representation $\boldsymbol{\Phi^d}$ and reconstruction **weight vector** $\boldsymbol{W_o^d}$:

$$\boldsymbol{\beta_o^d} = \boldsymbol{\Phi^d W_o^d}. \tag{5.19}$$

The scatter matrix $\boldsymbol{S^d}$ encodes the variance information at d^{th} layer, and is given by:

$$\boldsymbol{S^d} = \boldsymbol{\Phi^d \mathcal{L}^d (\Phi^d)^T}. \tag{5.20}$$

Now, by using (5.19) and (5.20), the minimization criterion in (5.18) is reformulated as follows:

$$\begin{aligned} \underset{\boldsymbol{W_o^d}, e_i^d}{\text{Minimize}} : \mathcal{L}_{GMKOC^d} &= \frac{1}{2}(\boldsymbol{W_o^d})^{\boldsymbol{T}}(\boldsymbol{\Phi^d})^{\boldsymbol{T}}(\boldsymbol{\Phi^d \mathcal{L}^d (\Phi^d)^T} + \lambda \boldsymbol{I})\boldsymbol{\Phi^d W_o^d} + \frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2 \\ \text{Subject to} : \ & (\boldsymbol{W_o^d})^{\boldsymbol{T}}(\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\phi_i^d} = r - e_i^d, \ i = 1, 2, ..., N. \end{aligned} \tag{5.21}$$

Further, we substitute $\boldsymbol{K^d} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\Phi^d}$, and $\boldsymbol{k_i^d} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\phi_i^d}$ (where the individual elements of $\boldsymbol{k_i^d}$ equal to $\boldsymbol{k_{ij}^d} = (\boldsymbol{\phi_i^d})^{\boldsymbol{T}}\boldsymbol{\phi_j^d}, j = 1, 2, \ldots, N$) in (5.21). Now, the optimization problem in (5.21) can be reformulated as follows:

$$\begin{aligned} \underset{\boldsymbol{W_o^d}, e_i^d}{\text{Minimize}} : \mathcal{L}_{GMKOC^d} &= \frac{1}{2}(\boldsymbol{W_o^d})^{\boldsymbol{T}}(\boldsymbol{K^d \mathcal{L}^d K^d} + \lambda \boldsymbol{K^d})\boldsymbol{W_o^d} + \frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2, \\ \text{Subject to} : \ & (\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{k_i^d} = r - e_i^d, \ i = 1, 2, ..., N. \end{aligned} \tag{5.22}$$

The Lagrangian relaxation of (5.22) is given below:

$$\begin{aligned}\pounds_{GMKOC^d} &= \frac{1}{2}(\boldsymbol{W_o^d})^{\boldsymbol{T}}(\boldsymbol{K^d}\boldsymbol{\mathcal{L}^d}\boldsymbol{K^d} + \lambda \boldsymbol{K^d})\boldsymbol{W_o^d} \\ &+\frac{C}{2}\sum_{i=1}^{N}\left\|e_i^d\right\|_2^2 - \sum_{i=1}^{N}\boldsymbol{\alpha_i^d}((\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{k_i^d} - r + e_i^d),\end{aligned} \tag{5.23}$$

where $\boldsymbol{\alpha^d} = \boldsymbol{\{\alpha_i^d\}}, i = 1, 2 \ldots N$, is a Lagrangian multiplier. In order to optimize (5.23), we compute its derivatives as follows:

$$\frac{\partial \pounds_{GMKOC^d}}{\partial \boldsymbol{W_o^d}} = 0 \Rightarrow \boldsymbol{W_o^d} = (\boldsymbol{\mathcal{L}^d}\boldsymbol{K^d} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\alpha^h}, \tag{5.24}$$

$$\frac{\partial \pounds_{GMKOC^d}}{\partial \boldsymbol{e_i^d}} = 0 \Rightarrow \boldsymbol{E^d} = \frac{1}{C}\boldsymbol{\alpha^h}, \tag{5.25}$$

$$\frac{\partial \pounds_{GMKOC^d}}{\partial \boldsymbol{\alpha_i^d}} = 0 \Rightarrow (\boldsymbol{W_o^d})^{\boldsymbol{T}}\boldsymbol{K^d} = \boldsymbol{r} - \boldsymbol{E^h}. \tag{5.26}$$

The weight vector $\boldsymbol{W_o^d}$ is obtained by substituting (5.25) and (5.26) into (5.24), and is given by:

$$\boldsymbol{W_o^d} = \left(\boldsymbol{K^d} + \frac{1}{C}\boldsymbol{\mathcal{L}^d}\boldsymbol{K^d} + \frac{\lambda}{C}\boldsymbol{I}\right)^{-1}\boldsymbol{r}. \tag{5.27}$$

Now, $\boldsymbol{\beta_o^d}$ is derived by substituting (5.27) into (5.19):

$$\boldsymbol{\beta_o^d} = \boldsymbol{\Phi^d}\left(\boldsymbol{K^d} + \frac{1}{C}\boldsymbol{\mathcal{L}^d}\boldsymbol{K^d} + \frac{\lambda}{C}\boldsymbol{I}\right)^{-1}\boldsymbol{r}. \tag{5.28}$$

The predicted output of the final layer (i.e., d^{th} layer) of GMKOC for training samples is calculated as follows:

$$\boldsymbol{\widehat{O}} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\beta_o^d} = (\boldsymbol{\Phi^d})^{\boldsymbol{T}}\boldsymbol{\Phi^d}\boldsymbol{W_o^d} = \boldsymbol{K^d}\boldsymbol{W_o^d}, \tag{5.29}$$

where $\boldsymbol{\widehat{O}}$ is the predicted output for training data.

After completing the training process and getting the predicted output of the training data, a threshold is required to decide whether any sample is an outlier or not. Two types of threshold criteria ($\theta 1$ and $\theta 2$) are discussed in the next section.

5.1.3 Decision Function

Two types of thresholds namely, $\theta 1$ and $\theta 2$, are employed with the proposed methods, which are determined as follows:

(i) **For $\theta 1$:**

Step I We calculate the distance between the predicted value of the i^{th} training sample and r, and store in a vector, $\mathbf{\Lambda} = \{\Lambda_i\}$ and $i = 1, 2, ..., N$, as follows:

$$\Lambda_i = \left| \widehat{O}_i - r \right|. \tag{5.30}$$

Step II After storing all distances in $\mathbf{\Lambda}$ as per (5.30), we sort these distances in decreasing order and denoted by a vector $\boldsymbol{\Lambda_{dec}}$. Further, we reject few percent of training samples based on the deviation. Most deviated samples are rejected first because they are most probably far from the distribution of the target data. The threshold is decided based on these deviations as follows:

$$\theta 1 = \Lambda_{dec}(\lfloor \nu * N \rfloor), \tag{5.31}$$

where $0 < \nu \leq 1$ is the fraction of rejection of training samples for deciding threshold value. N is the number of training samples and $\lfloor\ \rfloor$ denotes the floor operation.

(ii) **For $\theta 2$:** We select threshold ($\theta 2$) as a small fraction of the mean of the predicted output:

$$\theta 2 = (\left\lfloor \nu * \text{mean}(\widehat{\boldsymbol{O}}) \right\rfloor), \tag{5.32}$$

where $0 < \nu \leq 1$ is the fraction of rejection for deciding threshold value.

Training process is completed after getting a threshold value by either using $\theta 1$ or $\theta 2$. Afterwards, during testing, a test vector $\boldsymbol{x_p}$ is fed to the trained multi-layer architecture and its output $\widehat{O}_p$ is obtained. Further, compute $\widehat{\Lambda}_p$ for any one types of threshold as follows:

For $\theta 1$, calculate the distance ($\widehat{\Lambda}_p$) between the predicted value $\widehat{O}_p$ of the p^{th} testing sample and r as follows:

$$\widehat{\Lambda}_p = \left| \widehat{O}_p - r \right|. \tag{5.33}$$

For $\theta 2$, calculate the distance ($\widehat{\Lambda}$) between the predicted value $\widehat{O}_p$ of the p^{th} testing sample and mean of the predicted values obtained after training as follows:

$$\widehat{\Lambda}_p = \left| \widehat{O}_p - \text{mean}(\widehat{O}) \right|. \tag{5.34}$$

Finally, $\boldsymbol{x_p}$ is classified based on the following rule:

$$\boldsymbol{x_p} \text{ belongs to } \begin{cases} \text{Target class}, & \text{If } \widehat{\Lambda}_p \leq \text{Threshold} \\ \text{Outlier}, & \text{otherwise.} \end{cases} \tag{5.35}$$

The complete training steps followed by GMKOC are described in Algorithm 5.1.

Overall, we propose 8 variants using 2 types of threshold criteria (viz., $\theta 1$ and $\theta 2$) and 4 types of Laplacian graphs (viz., Laplacian eigenmaps (LE) [127], locally linear embedding (LLE) [172], linear discriminant analysis (LDA) [129], clustering-based LDA (CDA) [131]). For generating the names of these 8 variants, we concatenate the name of the Laplacian graph and types of threshold criteria with the name of the proposed method GMKOC. Those 8 variants are GMKOC-LE_$\theta 1$, GMKOC-LE_$\theta 2$, GMKOC-LLE_$\theta 1$, GMKOC-LLE_$\theta 2$, GMKOC-LDA_$\theta 1$, GMKOC-LDA_$\theta 2$, GMKOC-CDA_$\theta 1$, and GMKOC-CDA_$\theta 2$.

5.2 Experiments

In this chapter, we follow the same experimental setup and datasets, as mentioned in Section 3.2 of Chapter 3. In order to compare our proposed methods (viz., AEKOC (Chapter 3), MKOC (Chapter 4), and GMKOC) with the existing methods, we have conducted experimentation on various existing state-of-the-art kernel-based OCC methods, such as (a) 4 variants of KRR-based Graph-Embedding method GKOC [8] viz., GKOC-LE, GKOC-LLE, GKOC-LDA, and GKOC-CDA, (b) 4 with-

Algorithm 5.1 Graph-Embedded Multi-layer KRR-based One-class classification: GMKOC

Input: Training set $\boldsymbol{X}$, regularization parameter (C), Graph regularization parameter (λ), non-linear feature map ($\boldsymbol{\Phi}$), number of layers (d)

Output: Target class or Outlier

Initially, $\boldsymbol{X^0 = X}$
for $h = 1$ to d **do**
    First Phase:
    if $h < d$ **then**
        Pass $\boldsymbol{X^{h-1}}$ as an input to h^{th} GKAE.
        Train the GKAE as per (5.11).
        Compute the triplet $(\boldsymbol{X^h, \beta_a^h, S^h})$ during training of h^{th} GKAE.
        Transformed output $\boldsymbol{X^h}$ is computed from the current layer and pass it as the input to the next layer, i.e., $(h+1)^{th}$, in the hierarchy.
        Second Phase:
    else
        Train final layer i.e. d^{th} layer for one-class classification.
        Output of $(d-1)^{th}$ Auto-Encoder is passed as an input to the one-class classifier at d^{th} layer.
        Train the d^{th} layer by GMKOC^d as per 5.22
    end if
end for
Compute a threshold either $\theta 1$ (5.31) or $\theta 2$ (5.32).
At final step, whether a new input is outlier or not, decides based on the rule discussed in 5.35.

out Graph-Embedding methods viz., OCSVM [1], SVDD [1], KPCA [4], and KOC [7]. For all existing methods, we have used threshold criterion $\theta 1$ as mentioned in their corresponding paper. The proposed methods MKOC_θ1, GMKOC-LE_θ1, GMKOC-LLE_θ1, GMKOC-LDA_θ1, and GMKOC-CDA_θ1 are multi-layer version of the existing single hidden layer-based classifiers KOC, GKOC-LE, GKOC-LLE, GKOC-LDA, and GKOC-CDA, respectively. Overall, performance among 19 variants of one-class classifier is presented next.

Performance of 19 variants of one-class classifiers in terms of η_g are presented in Table 5.1. The best η_g values are kept in bold in this table. After analyzing this table, it is observed that GMKOC-CDA_θ2 yields the best η_g for the maximum number of datasets, i.e., 4 out of 23 datasets. Table 5.2 shows that number of datasets with the best η_g value for each classifier. In this table, proposed multi-layer classifiers from this chapter (with Graph-Embedding) and previous chapter (without Graph-Embedding) collectively yield the best results for 20 out of 23 datasets. Out of 20 best results, the proposed Graph-Embedded multi-layer classifier's variants collectively yield the best results for 17 datasets. Existing classifiers yield the best results for only 3 out of 23 datasets. These existing classifiers are single hidden layer-based one-class classifiers. Therefore, it can be stated that multi-layer classifiers have clearly outperformed single hidden layer-based classifiers for most of the datasets. In Table 5.2, any one classifier yields the best η_g for maximum 17.39% of datasets. Hence, we can not declare any one classifier as the best classifier just by looking at the best η_g value in Table 5.1. We need some other performance criteria to decide the best performing classifier. As per discussion in Chapter 3, we compute three criteria [167] for further analysis: (i) average of η_g (η_m) (ii) Friedman test (F-score and p-value) (iii) Friedman rank (η_f). We analyze the performance of 19 classifiers based on these three criteria as follows:

(i) We compute η_m for analyzing the combined performance of the classifier. η_m value of each classifier is available in the last row of Table 5.1. It is also provided in Table 5.3(a) in decreasing order of η_m. When we compare η_m values in this table, it is observed that all multi-layer classifiers yield better results compared to their corresponding single hidden layer-based version. Moreover, all the pro-

Table 5.1: Performance in terms of η_g for 23 datasets

Datasets	KPCA	OCSVM	SVDD	KOC	GKOC-LDA	GKOC-CDA	GKOC-LE	GKOC-LLE	AEKOC	MKOC_θ1	MKOC_θ2	GMKOC-LDA_θ1	GMKOC-LDA_θ2	GMKOC-CDA_θ1	GMKOC-CDA_θ2	GMKOC-LE_θ1	GMKOC-LE_θ2	GMKOC-LLE_θ1	GMKOC-LLE_θ2
Aust(1)	63.69	66.08	65.55	65.07	64.98	67.48	65.09	62.95	72.88	72.12	72.43	72.24	72.62	77.55	79.68	72.33	72.61	77.81	**81.70**
Aust(2)	73.06	76.59	76.78	74.21	73.71	73.74	74.03	70.72	77.96	79.89	80.31	79.48	80.32	79.55	**80.49**	79.78	80.27	75.38	79.85
Bupa(1)	62.91	60.64	60.64	57.09	57.04	57.07	57.12	56.28	56.19	62.55	62.19	63.06	62.77	62.13	**63.07**	62.59	62.21	61.69	62.04
Bupa(2)	74.28	69.78	69.75	68.81	68.81	68.81	68.77	67.72	68.31	73.56	73.61	73.52	73.07	72.83	**75.31**	73.49	73.80	71.65	73.76
Ecoli(1)	65.79	89.43	89.49	89.38	88.79	89.48	89.31	87.42	89.00	82.51	80.36	92.22	89.70	92.28	90.06	**92.75**	88.71	89.13	88.66
Ecoli(2)	72.30	79.42	78.87	82.32	82.22	82.80	82.28	82.24	79.16	84.42	81.76	85.75	82.22	86.32	82.77	85.00	**87.24**	84.79	81.95
Germ(1)	80.77	80.34	81.10	73.17	72.53	72.60	72.75	70.86	74.04	74.10	**82.79**	73.58	82.76	73.49	81.65	75.45	82.70	74.04	82.19
Germ(2)	49.75	52.80	52.77	53.41	52.94	52.90	53.06	52.51	51.57	**54.46**	49.59	54.31	50.09	53.37	49.56	54.23	50.64	51.89	49.53
Glass(1)	57.69	59.61	59.61	58.91	58.55	58.49	59.01	59.23	59.29	62.46	59.23	62.52	59.97	**62.58**	61.29	62.07	60.45	59.66	61.45
Glass(2)	77.65	73.32	72.89	73.08	73.14	73.14	72.94	72.07	73.22	77.15	75.69	78.23	77.02	76.84	78.23	78.44	**79.46**	76.59	78.45
Heart(1)	70.42	72.91	72.91	65.03	64.44	64.39	64.19	59.00	67.99	71.72	73.93	71.40	**74.52**	71.15	74.25	72.06	74.28	67.53	70.28
Heart(2)	63.50	64.90	64.90	66.39	62.88	64.84	64.38	66.00	61.15	68.89	59.51	67.91	60.09	**69.86**	58.79	67.89	60.08	64.30	62.64
Iono(1)	76.54	**93.13**	**93.13**	92.69	92.06	92.06	92.26	89.18	92.95	89.54	88.77	88.85	88.41	88.81	89.81	88.77	88.32	88.55	90.73
Iono(2)	57.10	44.63	44.63	53.40	43.04	43.04	45.64	53.90	41.04	**67.70**	60.78	59.91	42.39	67.50	65.02	59.91	39.64	60.30	49.28
Iris(1)	96.44	85.06	84.12	92.35	89.67	89.67	90.35	92.35	92.79	**99.59**	93.87	98.33	94.91	98.33	95.71	97.09	95.80	**99.59**	95.36
Iris(2)	72.99	81.70	81.70	85.59	84.34	85.68	85.03	85.31	88.37	77.20	74.07	75.93	72.85	76.95	78.79	74.35	76.79	90.26	**90.90**
Iris(3)	69.29	83.18	82.67	83.40	83.58	83.34	84.05	80.83	83.74	71.29	69.82	70.24	71.32	71.39	70.88	64.40	71.30	**86.67**	84.96
Jap(1)	64.09	71.45	70.15	67.33	67.12	74.58	67.24	64.97	76.23	74.45	74.66	74.60	74.92	77.91	80.13	75.03	75.20	80.80	**83.72**
Jap(2)	72.29	75.78	76.58	73.48	72.73	72.75	73.15	69.79	78.27	80.14	80.55	79.91	80.54	79.52	**80.72**	80.08	80.53	74.63	78.58
Park(1)	70.20	95.71	**97.07**	96.15	96.07	96.07	96.05	96.15	96.67	91.48	91.70	91.36	91.87	90.93	92.86	90.59	91.68	91.40	92.40
Park(2)	67.79	81.82	79.77	**90.74**	90.50	90.48	90.66	90.16	83.78	80.97	79.65	81.35	79.72	81.35	79.72	81.06	79.43	80.95	79.57
Pima(1)	77.98	79.18	79.21	79.04	78.94	78.89	79.02	77.50	78.66	79.21	80.50	79.83	**80.60**	79.71	80.52	79.79	80.41	78.20	80.15
Pima(2)	57.05	56.59	56.71	54.78	54.57	54.50	54.58	52.02	54.01	57.70	57.80	58.00	57.96	**58.92**	58.44	57.69	57.95	56.21	57.85
Mean of all η_g (η_m)	69.28	73.65	73.52	73.73	72.72	73.34	73.09	72.14	73.79	75.35	74.07	75.33	73.94	76.06	75.99	74.99	74.33	75.74	**76.35**

Table 5.2: Number of datasets for which each one-class classifier yields the best η_g

One-class Classifiers	Number of datasets with the best η_g value
GMKOC-CDA_θ2	4
GMKOC-CDA_θ1	3
GMKOC-LLE_θ2	3
MKOC_θ1	3
GMKOC-LDA_θ2	2
GMKOC-LE_θ2	2
GMKOC-LLE_θ1	2
SVDD	2
GMKOC-LE_θ1	1
MKOC_θ2	1
KOC	1
OCSVM	1
GMKOC-LDA_θ1	0
KPCA	0
GKOC-LDA	0
GKOC-CDA	0
GKOC-LE	0
GKOC-LLE	0
AEKOC	0

Table 5.3: Friedman Rank (η_f) and mean of η_g (η_m)

5.3(a) In decreasing order of η_m

	η_f	η_m
GMKOC-LLE_θ2	8.04	76.35
GMKOC-CDA_θ1	7.13	76.06
GMKOC-CDA_θ2	6.54	75.99
GMKOC-LLE_θ1	9.50	75.74
MKOC_θ1	7.85	75.35
GMKOC-LDA_θ1	7.41	75.33
GMKOC-LE_θ1	7.93	74.99
GMKOC-LE_θ2	8.74	74.33
MKOC_θ2	10.13	74.07
GMKOC-LDA_θ2	8.89	73.94
AEKOC	11.13	73.79
KOC	10.78	73.73
OCSVM	10.41	73.65
SVDD	10.59	73.52
GKOC-CDA	11.93	73.34
GKOC-LE	11.87	73.09
GKOC-LDA	13.07	72.72
GKOC-LLE	14.04	72.14
KPCA	14.00	69.28

5.3(b) In increasing order of η_f

	η_f	η_m
GMKOC-CDA_θ2	6.54	75.99
GMKOC-CDA_θ1	7.13	76.06
GMKOC-LDA_θ1	7.41	75.33
MKOC_θ1	7.85	75.35
GMKOC-LE_θ1	7.93	74.99
GMKOC-LLE_θ2	8.04	76.35
GMKOC-LE_θ2	8.74	74.33
GMKOC-LDA_θ2	8.89	73.94
GMKOC-LLE_θ1	9.50	75.74
MKOC_θ2	10.13	74.07
OCSVM	10.41	73.65
SVDD	10.59	73.52
KOC	10.78	73.73
AEKOC	11.13	73.79
GKOC-LE	11.87	73.09
GKOC-CDA	11.93	73.34
GKOC-LDA	13.07	72.72
KPCA	14.00	69.28
GKOC-LLE	14.04	72.14

posed classifiers (Graph-Embedded and without Graph-Embedded) outperform all the existing classifiers (Graph-Embedded and without Graph-Embedded). We find an interesting observation that GMKOC-CDA_θ2 obtains 3^{rd} position in Table 5.2 despite yielding the best η_g for maximum number of datasets. Overall, GMKOC-LLE_θ2 attains a top position in the table and yields the best η_m among 19 classifiers.

(ii) Although the proposed Graph-Embedded classifier's variants attain top 4 positions among 19 variants of classifiers in Table 5.3(a); however, we need to verify the outcomes of the proposed and existing classifiers statistically. For this purpose, we conduct a non-parametric Friedman test [166] similar to discussed in Section 3.2 of Chapter 3. Friedman test mainly computes three components viz., p-value, F-score, and critical value. The computed p-value, F-score value and critical value are $8.6508e-08$, 68.33 and 28.87, respectively. Since the computed F-score value is higher than the critical value, and the p-value is significantly lesser than the tolerance level of 0.05, we reject the null hypothesis with 95% of confidence. Therefore, it is concluded that the outcomes presented in this chapter are statistically significant.

(iii) Further, we compute Friedman Rank (η_f) [166] for each classifier as discussed in Section 2.8 of Chapter 2. We consider η_f as the final decision criteria to decide the rank of any classifiers. In Table 5.3(b), we present η_f along with η_m of each classifier in increasing order of η_f. In this table, GMKOC-LLE_θ2 does not attain top position in spite of the best η_m in Table 5.3(a). η_f value of GMKOC-LLE_θ2 is significantly lesser (1.5) than the best η_f value. GMKOC-CDA_θ2 attains top position among 19 variants of classifiers. Overall, GMKOC variants have outperformed all existing classifiers in terms of both criteria i.e., η_f and η_m. Moreover, all multi-layer classifiers yield better η_f compared to their corresponding single hidden layer-based version. Among four types of used Laplacian graphs, CDA-based GMKOC variants (GMKOC-CDA_θ1 and GMKOC-CDA_θ2) perform better than the other variants of GMKOC.

5.3 Summary

This chapter has proposed a Graph-Embedded multi-layer method for OCC (i.e., GMKOC), which follows a non-iterative approach to learning. Similar to MKOC, it has combined the concept of two different frameworks, i.e., boundary and reconstruction, in sequential order. In order to construct the Graph-Embedded multi-layer architecture, we have proposed a GKAE. It presents a better representation of data compared to KAE, which has resulted in better generalization performance of GMKOC compared to MKOC. Overall, we have proposed eight variants using 2 types of threshold criteria and 4 types of Laplacian graphs. Three out of eight variants have obtained better η_m and η_f values compared to all previously (i.e., in previous chapters of this thesis) proposed and existing one-class classifiers. We have statistically verified that the outcomes presented in this chapter are statistically significant, with 95% of confidence. Moreover, the p-value is significantly lesser than the tolerance level of 0.05.

In this and previous chapters, we have combined various kernels in sequential order, and obtained better results compared to single hidden layer-based one-class classifier. In the later chapter, we combine multiple kernels simultaneous instead of sequentially with hyperplane and hypersphere-based two distinct optimization problems.

Chapter 6

Localized Multiple Kernel Learning for One-class Classification

In the previous chapter, we have stacked multiple kernels in sequential order and formed a multi-layer architecture for OCC. This chapter presents an optimization problem for data-driven anomaly detection in which a convex combination of kernels is used. Here, all kernels are optimized in a single optimization problem, and weights are assigned locally based on the local region of the data. We develop two types of MKL-based one-class classifiers by taking OCSVM and SVDD as base classifiers. OCSVM and SVDD-based proposed classifiers are referred to as localized multiple kernel anomaly detection (LMKAD) and localized multiple kernel support vector data description (LMSVDD), respectively. Both methods are based on the boundary framework. The proposed LMKAD and LMSVDD are discussed in Section 6.1 and 6.2, respectively.

6.1 Localized Multiple Kernel Learning for Anomaly Detection: LMKAD

In this section, we propose Localized Multiple Kernel Anomaly Detection (LMKAD). In LMKAD, weights are assigned to kernel, based on the locality present in the data. We intend to give more weights to those kernel functions, which best

match the underlying locality of the data in different regions of the input space. We modify the decision function of OCSVM (mentioned in (2.7) of Chapter 2) as follows:

$$f(\boldsymbol{x}) = \sum_{m=1}^{p} \eta_m(\boldsymbol{x}) \langle \boldsymbol{\omega_m}, \boldsymbol{\phi_m(x)} \rangle - \rho, \tag{6.1}$$

where $\boldsymbol{\omega}$ is the weight coefficients, $\phi(.)$ represents the mapping in the feature space, ρ denotes bias term, p denotes number of kernels, and $\eta_m(x)$ is the weight corresponding to each kernel. Here, $\eta_m(x)$ is called as gating function. The $\eta_m(x)$ is defined by the input $\boldsymbol{x}$, and parameters of the gating function. These parameters learn from the data during optimization, as shown later in this section. We rewrite the conventional OCSVM optimization problem with our new decision function and obtain the following primal optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{\omega_m}, \eta_m(\boldsymbol{x}), \boldsymbol{\xi}, \rho}{\text{Minimize}} : \frac{1}{2} \sum_{m=1}^{p} \boldsymbol{\omega_m^T \omega_m} - \rho + \frac{1}{\nu N} \sum_{i=1}^{N} \xi_i \\ &\text{Subject to} : \sum_{m=1}^{p} \eta_m(\boldsymbol{x}) \langle \boldsymbol{\omega_m}, \boldsymbol{\phi_m(x)} \rangle \geq \rho - \xi_i \quad \forall i, \\ &\xi_i \geq 0 \quad \forall i, \end{aligned} \tag{6.2}$$

where ν is the rate of rejection, N is the total number of training samples, and ξ_i is the slack variable as usual. We now need to solve this optimization problem for the above parameters. But, the minimization problem in (6.2) is not convex because non-linearity is introduced in the separation constraints due to the gating function. It might lead to suboptimal global solution [12]. Therefore, we do not solve this optimization problem directly, but use a two-step alternate optimization scheme inspired by Rakotomamonjy et al. [50] and Gonen et al. [12]. This optimization scheme finds the values of the parameters of the gating function ($\eta_m(\boldsymbol{x})$) and the parameters of the decision function.

Before starting the optimization procedure, we initialize the parameters of $\eta_m(\boldsymbol{x})$ by some random values. Then, **in the first step** of the procedure, we treat $\eta_m(\boldsymbol{x})$ as a

constant and solve the optimization problem (6.2) for $\boldsymbol{\omega}, \boldsymbol{\xi}$ and ρ. Note that if we treat $\eta_m(\boldsymbol{x})$ as a constant, this step is essentially the same as solving conventional OCSVM under certain conditions as we will explain later. Solving the conventional OCSVM returns the optimal value of the objective function and the Lagrange multipliers. **In the second step**, we update the values of the parameters of $\eta_m(\boldsymbol{x})$ using gradient descent on the objective function. The updated parameters define a new $\eta_m(\boldsymbol{x})$, which is used for the next iteration. The above two steps are repeated until convergence. For a fixed $\eta_m(\boldsymbol{x})$, the Lagrangian of the primal problem in (6.2) is written as:

$$L_D = \frac{1}{2}\sum_{m=1}^{p} \boldsymbol{\omega_m^T \omega_m} + \sum_{i=1}^{N}\Big(\frac{1}{\nu N} - \beta_i - \alpha_i\Big)\xi_i - \rho - \sum_{i=1}^{N}\alpha_i\Big(\sum_{m=1}^{p}\eta_m(\boldsymbol{x_i})\langle \boldsymbol{\omega_m}, \boldsymbol{\phi_m(x_i)}\rangle - \rho\Big), \tag{6.3}$$

where $\alpha_i \geq 0$ and $\beta_i \geq 0$ are the Lagrange multipliers.. Further, we compute the derivatives of the Lagrangian L_D with respect to the variables in (6.2) as follows:

$$\frac{\partial L_D}{\partial \boldsymbol{\omega_m}} = 0 \Rightarrow \boldsymbol{\omega_m} = \sum_{i=1}^{N}\alpha_i\eta_m(\boldsymbol{x_i})\boldsymbol{\phi_m(x_i)} \quad \forall m, \tag{6.4}$$

$$\frac{\partial L_D}{\partial \rho} = 0 \Rightarrow \sum_{i=1}^{N}\alpha_i = 1, \tag{6.5}$$

$$\frac{\partial L_D}{\partial \xi_i} = 0 \Rightarrow \frac{1}{\nu N} = \beta_i + \alpha_i. \tag{6.6}$$

By substituting (6.4), (6.5), and (6.6) into (6.3), we obtain the dual problem of (6.2) as follows:

$$\begin{aligned} &\underset{\boldsymbol{\alpha}}{\text{Maximize}} : J(\eta) = -\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha} \\ &\text{Subject to :}\ 0 \leq \alpha_i \leq 1 \quad i = 0, \ldots, N, \\ &\text{Subject to :}\ \sum_{i}^{N}\alpha_i = 1, \end{aligned} \tag{6.7}$$

where $k_{ij} = \mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$ is a weighted kernel function, and $\boldsymbol{K}$ is the kernel matrix where $k_{ij} \in \boldsymbol{K}$ generates an element of $\boldsymbol{K}$ between i^{th} and j^{th} sample. Here, the weighted kernel function is defined as:

$$\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \sum_{m=1}^{p}\eta_m(\boldsymbol{x_i})\langle \boldsymbol{\phi_m(x_i)}, \boldsymbol{\phi_m(x_j)}\rangle\eta_m(\boldsymbol{x_j}). \tag{6.8}$$

The objective function of the dual is termed as a function of $J(\eta)$. The dual formulation is exactly the same as the conventional OCSVM formulation with kernel function $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$. In (6.8), multiplying the kernel matrix with a non-negative value will still give a positive definite matrix [173]. Note that the locally combined kernel function will, therefore, satisfy the Mercer's condition [174] if the gating function is non-negative for both the input instances. This can be easily ensured by picking a non-negative $\eta_m(\boldsymbol{x})$. In order to assign the weight to the different kernels based on the training data, a gating function is used.

In the above formulations of LMKAD, the objective value of the dual formulation (6.7) is equal to the objective value of the primal (6.2). By using the dual formulation in (6.7), gating function is trained and $\eta_m(\boldsymbol{x})$ is computed. For training it, the gradient of the objective function of the dual (i.e., $J(\eta)$) is computed with respect to the parameters of the gating function. In this way, we obtain new parameter values of the gating function. Now substitute the new value of $\eta_m(\boldsymbol{x})$ in (6.8), which gives us the new $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$. This new $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$ is used to solve the dual formulation (6.7). We again update the value of the gating function parameters and repeat until convergence. The convergence of the algorithm is determined by observing the change in the objective function in (6.7). When the change in the objective of the current and previous iteration is less than some predefined very small number; then, it converges. Step size for every iteration is computed using the line search approach [175]. The entire algorithm is summarized in Algorithm 6.1.

Algorithm 6.1 LMKAD algorithm

1: Initialize the values of gating function parameters $\boldsymbol{v_m}$ and v_{m0} by random values for each m^{th} kernel, where $m = 1, 2, \ldots, p$
2: **do**
3: Calculate η_m using $\boldsymbol{v_m^t}$ and v_{m0}^t
4: Calculate $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$ using the gating function
5: Solve conventional OCSVM with $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$
6: $\boldsymbol{v_m^{(t+1)}} \Leftarrow \boldsymbol{v_m^{(t)}} - \mu^{(t)} \frac{\partial J(\eta)}{\partial \boldsymbol{v_m}} \quad \forall m$
7: $v_{m0}^{(t+1)} \Leftarrow v_{m0}^{(t)} - \mu^{(t)} \frac{\partial J(\eta)}{\partial v_{m0}} \quad \forall m$
8: **while** Not converged

Once the algorithm converges, and the final $\eta_m(\boldsymbol{x})$ and the Lagrange multipliers are obtained, the decision function can be rewritten as:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \sum_{m=1}^{p} \alpha_i \eta_m(\boldsymbol{x}) \mathbb{K}_{\boldsymbol{m}}(\boldsymbol{x}, \boldsymbol{x_i}) \eta_m(\boldsymbol{x_i}) - \rho. \tag{6.9}$$

The sign of this decision function tells us whether the given input is target or outlier. We also compute the average error by computing the difference between the target and predicted value on the training data. Then we set the bias term to this mean value.

In the above discussion, the procedure of weight assignment using a gating function is discussed in detail. We have discussed in the algorithm of LMKAD that the derivative of the gating function needs to be computed for the local assignment of the weight to each kernel. Three types of gating functions [12, 146] are used in this chapter for our experiments, namely Softmax, Sigmoid, and Radial Basis Function. These gating functions and their derivatives are provided in the following sections.

6.1.1 Softmax Function

$$\eta_m(\boldsymbol{x}) = \frac{exp(\langle \boldsymbol{v_m}, \boldsymbol{x} \rangle + v_{m0})}{\sum_{\wp=1}^{p} exp(\langle \boldsymbol{v_\wp}, \boldsymbol{x} \rangle + v_{\wp 0})}. \tag{6.10}$$

This function is characterized by the parameters v_{m0} and $\boldsymbol{v_m}$ for m^{th} kernel, where v_{m0} is bias and $\boldsymbol{v_m}$ is weighting vector. In above equation, $exp(\langle \boldsymbol{v_m}, \boldsymbol{x} \rangle + v_{m0})$ represents simply $e^{(\boldsymbol{v_m x} + v_{m0})}$ (the same notation is followed for the Sigmoid function). The above function is called as the Softmax function and ensures that $\eta_m(\boldsymbol{x})$ is non-negative. Note that if we use a constant gating function, the algorithm reduces to that of $MKAD$ [11], and assigns fixed weights over the entire input space. As per the above discussion, gradient of the objective function $J(\eta)$ needs to be calculated with respect to the parameters of the Softmax gating function (v_{m0} and $\boldsymbol{v_m}$), which is mentioned as follows:

$$\frac{\partial J(\eta)}{\partial v_{m0}} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\wp=1}^{p}\alpha_i\alpha_j\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_\wp(\boldsymbol{x_j})(\delta_m^\wp - \eta_m(\boldsymbol{x_i}) + \delta_m^\wp - \eta_m(\boldsymbol{x_j})), \tag{6.11}$$

$$\frac{\partial J(\eta)}{\partial \boldsymbol{v_m}} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\wp=1}^{p}\alpha_i\alpha_j\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_\wp(\boldsymbol{x_j})(\boldsymbol{x_i}(\delta_m^\wp - \eta_m(\boldsymbol{x_i})) + \boldsymbol{x_j}(\delta_m^\wp - \eta_m(\boldsymbol{x_j}))), \tag{6.12}$$

$$\text{where} \quad \delta_m^\wp = \begin{cases} 1, \text{ if } m = \wp\ , \\ 0, \text{ otherwise.} \end{cases}$$

Once we obtain the updated values of the parameters v_{m0} and $\boldsymbol{v_m}$, we calculate the new value of $\eta_m(x)$ using the gating function.

6.1.2 Sigmoid Function

$$\eta_m(\boldsymbol{x}) = \frac{1}{1 + exp(-\langle \boldsymbol{v_m}, \boldsymbol{x}\rangle - v_{m0})}. \tag{6.13}$$

Again the parameters v_{m0} and $\boldsymbol{v_m}$ characterize the above gating function for m^{th} kernel. Here, The gradients of the objective function $J(\eta)$ with respect to the parameters of the sigmoid gating function (v_{m0} and $\boldsymbol{v_m}$) are:

$$\frac{\partial J(\eta)}{\partial v_{m0}} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\eta_m(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{m}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_m(\boldsymbol{x_j})(1 - \eta_m(\boldsymbol{x_i}) + 1 - \eta_m(\boldsymbol{x_j})), \tag{6.14}$$

$$\frac{\partial J(\eta)}{\partial \boldsymbol{v_m}} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j\eta_m(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{m}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_m(\boldsymbol{x_j})(\boldsymbol{x_i}(1 - \eta_m(\boldsymbol{x_i})) + \boldsymbol{x_j}(1 - \eta_m(\boldsymbol{x_j}))). \tag{6.15}$$

6.1.3 Radial Basis Function (RBF)

$$\eta_m(\boldsymbol{x}) = \frac{e^{-\frac{\|\boldsymbol{x}-\boldsymbol{\mu_m}\|_2^2}{\sigma_m^2}}}{\sum_{\wp=1}^{p} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{\mu_\wp}\|_2^2}{\sigma_\wp^2}}}, \tag{6.16}$$

where $\boldsymbol{\mu_m}$ is the center and σ_m gives the spread of the local region. Similar as above, the gradients of the objective function $J(\eta)$ with respect to the parameters of the RBF gating function ($\boldsymbol{\mu_m}$ and σ_m) are:

$$\begin{aligned}\frac{\partial J(\eta)}{\partial \boldsymbol{\mu_m}} = &-\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\wp=1}^{p}\alpha_i\alpha_j\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_\wp(\boldsymbol{x_j})((\boldsymbol{x_i}-\boldsymbol{\mu_m})(\delta_m^\wp-\eta_m(\boldsymbol{x_i}))+\\ &(\boldsymbol{x_j}-\boldsymbol{\mu_m})(\delta_m^\wp-\eta_m(\boldsymbol{x_j})))/\sigma_m^2,\end{aligned} \tag{6.17}$$

$$\begin{aligned}\frac{\partial J(\eta)}{\partial \sigma_m} = &-\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\wp=1}^{p}\alpha_i\alpha_j\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_\wp(\boldsymbol{x_j})(\|\boldsymbol{x_i}-\boldsymbol{\mu_m}\|_2^2(\delta_m^\wp-\eta_m(\boldsymbol{x_i}))+\\ &\|\boldsymbol{x_j}-\boldsymbol{\mu_m}\|_2^2(\delta_m^\wp-\eta_m(\boldsymbol{x_j})))/\sigma_m^3,\end{aligned} \tag{6.18}$$

$$\text{where} \quad \delta_m^\wp = \begin{cases} 1, \text{ if } m = \wp\ , \\ 0, \text{ otherwise.}\end{cases}$$

6.2 Localized Multiple Kernel Support Vector Data Description: LMSVDD

In this section, LMSVDD is proposed, which assigns weights based on the underlying locality of the data in different regions of the input space. The intuition behind these weights assignment is the same as discussed for LMKAD. That is if a kernel identifies the underlying locality of the data for some specific region; then, more weight is assigned to that kernel for that specific region of the input space. It is just different from LMKAD in the way of classification. LMKAD classifies by constructing

a hyperplane; however, LMSVDD constructs a hypersphere for classification. The decision function of SVDD (mentioned in (2.10) of Chapter 2) is modified for achieving localization as follows:

$$f(\boldsymbol{x}) = \sum_{m=1}^{p} \|\eta_m(\boldsymbol{x})\boldsymbol{\phi_m}(\boldsymbol{x}) - \boldsymbol{a}\|^2 - R^2, \tag{6.19}$$

where $\eta_m(x)$ is the weight corresponding to each kernel, $\phi(.)$ is a function which is mapping data in the higher dimensional feature space, $\boldsymbol{a}$ is a center, and R is a radius of the hypersphere. This weight is assigned by using a gating function. The $\eta_m(x)$ is function of the input $\boldsymbol{x}$ and is defined in terms of the gating function parameters. Learning of these parameters are performed through an optimization process, which is described further in this section. By using newly defined decision function in (6.19), SVDD primal optimization problem in (2.8) can be rewritten for LMSVDD as follows:

$$\begin{aligned} &\underset{R,\boldsymbol{a},\boldsymbol{\xi},\eta_m(\boldsymbol{x_i})}{\text{Minimize}} : R^2 + C\sum_{i=1}^{N}\xi_i \\ &\text{Subject to} : \sum_{m=1}^{p} \|\eta_m(\boldsymbol{x_i})\boldsymbol{\phi_m}(\boldsymbol{x_i}) - \boldsymbol{a}\|^2 \leq R^2 + \xi_i \quad , \\ &\xi_i \geq 0, \quad \forall i \in 1, \ldots, N, \end{aligned} \tag{6.20}$$

where N is the total number of training samples, and ξ_i are the slack variables as discussed above for SVDD. Now, primal optimization problem of LMSVDD is solved for the parameters $R, \boldsymbol{a}, \boldsymbol{\xi}$, and $\eta_m(\boldsymbol{x_i})$. However, this optimization problem is not solved directly. For solving this, we use a two-steps alternate optimization scheme as discussed in Rakotomamonjy et al. [50]. By using this alternate optimization scheme, the parameter(s) of the gating function ($\eta_m(\boldsymbol{x})$) and the parameter(s) of the decision function are computed.

Before going for two-steps optimization procedure, $\eta_m(\boldsymbol{x})$ is initialized for some random value. After this, **in the first step** of the optimization procedure, the primal optimization problem in (6.20) is solved for $R, \boldsymbol{a}, \boldsymbol{\xi}$, and $\eta_m(\boldsymbol{x_i})$ by treating $\eta_m(\boldsymbol{x})$ as a constant. So, if $\eta_m(\boldsymbol{x})$ is constant; then, this primal optimization problem can

be solved similar as vanilla SVDD. When we solve vanilla SVDD; then, it mainly yields two optimal values viz., the Objective function and the Lagrange multiplier. **In the second step of this optimization,** the parameters of $\eta_m(\boldsymbol{x})$ are updated by employing gradient descent on the objective function with respect to the parameters of the gating function $\eta_m(\boldsymbol{x})$. Then, we compute a new $\eta_m(\boldsymbol{x})$ by using the updated parameters and this is used in the next iteration. These two steps are repeated until convergence. For a fixed $\eta_m(\boldsymbol{x})$, the Lagrangian of the primal problem in (6.20) can be written as:

$$\begin{aligned} L_D(R, \boldsymbol{a}, \boldsymbol{\xi}, \eta_m(\boldsymbol{x_i}), \alpha_i, \beta_i) = R^2 + C\sum_{i=1}^{N}\xi_i + \\ \sum_{i=1}^{N}\alpha_i\Bigg(\sum_{m=1}^{p}\|\eta_m(\boldsymbol{x_i})\boldsymbol{\phi_m}(\boldsymbol{x_i}) - \boldsymbol{a}\|^2 - R^2 - \xi_i\Bigg) - \sum_{i=1}^{N}\beta_i\xi_i, \end{aligned} \tag{6.21}$$

where $\alpha_i \geq 0$ and $\beta_i \geq 0$ are the Lagrange multipliers. Now, the derivatives of the L_D is computed with respect to the variables in (6.20) as follows:

$$\frac{\partial L_D}{\partial \boldsymbol{a}} = 0 \Rightarrow \boldsymbol{a} = \sum_{i=1}^{N}\alpha_i\eta_m(\boldsymbol{x_i})\boldsymbol{\phi_m}(\boldsymbol{x_i}) \quad \forall m, \tag{6.22}$$

$$\frac{\partial L_D}{\partial R} = 0 \Rightarrow \sum_{i=1}^{N}\alpha_i = 1, \tag{6.23}$$

$$\frac{\partial L_D}{\partial \xi_i} = 0 \Rightarrow C = \alpha_i + \beta_i. \tag{6.24}$$

Substituting, (6.22), (6.23), and (6.24) into (6.21), following dual problem is obtained:

$$\begin{aligned} &\underset{\boldsymbol{\alpha}}{\text{Maximize}} : J(\eta) = \sum_{i=1}^{N}\alpha_i k_{ii} - \boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha} \\ &\text{Subject to} : \ 0 \leq \alpha_i \leq C \quad i = 0, \ldots, N \\ &\text{Subject to} : \ \sum_{i}^{N}\alpha_i = 1, \end{aligned} \tag{6.25}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]^T$, and $k_{ij} = \mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$ is a weighted kernel function, and $\boldsymbol{K}$ is the kernel matrix where $k_{ij} \in \boldsymbol{K}$ generates an element of $\boldsymbol{K}$ between i^{th} and j^{th}

sample. Here, the weighted kernel function is defined as:

$$\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j}) = \sum_{m=1}^{p} \eta_m(\boldsymbol{x_i}) \langle \boldsymbol{\phi_m}(\boldsymbol{x_i}), \boldsymbol{\phi_m}(\boldsymbol{x_j}) \rangle \eta_m(\boldsymbol{x_j}). \tag{6.26}$$

The dual objective function in (6.25) is written as a function of $J(\eta)$. Now, it can be easily seen from (6.25) and (2.9) that the dual formulation of SVDD and LMSVDD is identical except the kernel function $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$ formulation. This function always yields non-negative value because the multiplication of a kernel matrix with some non-negative value always yields a positive definite matrix. It can be analyzed from (6.26) that the local combination of the kernel is constructed by multiplying two input instances in terms of gating function. If the gating function $\eta_m(\boldsymbol{x})$ yields non-negative values for both input instances; then, the Mercer's condition will be satisfied. We have ensured this by always selecting a non-negative value of $\eta_m(\boldsymbol{x})$. Overall, by using a gating function $\eta_m(\boldsymbol{x})$, we assign a set of weights to various kernels on the training data.

In the above formulations of LMSVDD, objective value of the primal and dual formulation is equal as strong duality holds between primal and dual formulation[176]. Therefore, the dual formulation in (6.25) is used instead of primal formulation to train a gating function $\eta_m(\boldsymbol{x})$, and value of $\eta_m(\boldsymbol{x})$ is computed. Now the computed new value of $\eta_m(\boldsymbol{x})$ is substituted in (6.26) and obtain a updated value of $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$. This $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$ is used to solve the dual formulation in (6.25). The values of the $\eta_m(\boldsymbol{x})$ parameters are updated, and we repeat this procedure until the convergence. The convergence of the algorithm is decided based on the change in the value of the objective function in (6.25). Above discussed procedure is summarized in Algorithm 6.2. After the convergence of the algorithm, the final $\eta_m(\boldsymbol{x})$ and the $\boldsymbol{\alpha}$ are obtained.

Algorithm 6.2 LMSVDD algorithm

1: Initialize the values of $\boldsymbol{v_m}$ and v_{m0} by random values for each m^{th} kernel, where $m = 1, 2, \ldots, p$
2: **do**
3: Calculate η_m using $\boldsymbol{v_m^t}$ and v_{m0}^t
4: Calculate $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$ using the gating function
5: Solve conventional SVDD with $\mathbb{K}_{\boldsymbol{\eta}}(\boldsymbol{x_i}, \boldsymbol{x_j})$
6: $\boldsymbol{v_m}^{(t+1)} \Leftarrow \boldsymbol{v_m}^{(t)} - \mu^{(t)} \frac{\partial J(\eta)}{\partial \boldsymbol{v_m}} \quad \forall m$
7: $v_{m0}^{(t+1)} \Leftarrow v_{m0}^{(t)} - \mu^{(t)} \frac{\partial J(\eta)}{\partial v_{m0}} \quad \forall m$
8: **while** Not converged

The decision function of SVDD can be modified for LMSVDD as follows:

$$\begin{aligned} f(\boldsymbol{x}) &= \sum_{m=1}^{p} \|\eta_m(\boldsymbol{x})\boldsymbol{\phi_m}(\boldsymbol{x}) - \boldsymbol{a}\|^2 - R^2 \\ &= \sum_{m=1}^{p} \eta_m(x)(\boldsymbol{\phi_m}(\boldsymbol{x}))^{\boldsymbol{T}}\boldsymbol{\phi_m}(\boldsymbol{x})\eta_m(x) + \boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha} - \\ & 2\sum_{i=1}^{N}\sum_{m=1}^{p} \alpha_i\eta_m(\boldsymbol{x})\mathbb{K}_m(x, x_i)\eta_m(\boldsymbol{x_i}) - R^2. \end{aligned} \tag{6.27}$$

The sign of this decision function tells us whether the given input is target or outlier. If $\boldsymbol{f(x)} \geq \boldsymbol{0}$; then, x is an outlier, otherwise, a normal sample. Here, R^2 is a radius of the sphere and calculated as follows:

$$\begin{aligned} R^2 = &\sum_{m=1}^{p} \eta_m(\boldsymbol{x_\alpha})(\boldsymbol{\phi_m}(\boldsymbol{x_\alpha}))^{\boldsymbol{T}}\boldsymbol{\phi_m}(\boldsymbol{x_\alpha})\eta_m(\boldsymbol{x_\alpha}) + \boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha} - \\ & 2\sum_{i=1}^{N}\sum_{m=1}^{p} \alpha_i\eta_m(\boldsymbol{x_i})\mathbb{K}_m(\boldsymbol{x_i}, \boldsymbol{x_\alpha})\eta_m(\boldsymbol{x_\alpha}). \end{aligned} \tag{6.28}$$

where $\boldsymbol{x_\alpha}$ is support vectors.

In the above discussion, the procedure of weight assignment using a gating function is discussed in detail. We have discussed in the algorithm of LMSVDD that the derivative of the gating function needs to be computed for the local assignment of the weight to each kernel. Three types of gating functions [12, 146] are used in this chapter for our experiments, namely Softmax, Sigmoid, and Radial Basis Function.

Expressions of these gating functions are already discussed in Section 6.1. Derivatives of these gating functions for LMSVDD are provided in the following sections.

6.2.1 Softmax Function

Softmax function is expressed in (6.10) during discussion of LMKAD. This function is characterized by the parameters v_{m0} and $\boldsymbol{v_m}$. This function ensures that $\eta_m(\boldsymbol{x})$ is non-negative. As we have seen in the above discussion that the gradient of the objective function $J(\eta)$ needs to be calculated with respect to the parameters of the Softmax gating function (v_{m0} and $\boldsymbol{v_m}$). These derivatives are mentioned as follows:

$$\begin{aligned}\frac{\partial J(\eta)}{\partial v_{m0}} &= \sum_{i=1}^{N}\sum_{\wp=1}^{p}\alpha_i\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_i})\eta_\wp(\boldsymbol{x_i})\times(\delta_m^\wp-\eta_m(\boldsymbol{x_i})+\delta_m^\wp-\eta_m(\boldsymbol{x_i})) \\ &-\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\wp=1}^{p}\alpha_i\alpha_j\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_\wp(\boldsymbol{x_j})(\delta_m^\wp-\eta_m(\boldsymbol{x_i})+\delta_m^\wp-\eta_m(\boldsymbol{x_j})),\end{aligned} \tag{6.29}$$

$$\begin{aligned}\frac{\partial J(\eta)}{\partial \boldsymbol{v_m}} &= \sum_{i=1}^{N}\sum_{\wp=1}^{p}\alpha_i\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_i})\eta_\wp(\boldsymbol{x_i})\times(\boldsymbol{x_i}(\delta_m^\wp-\eta_m(\boldsymbol{x_i}))+\boldsymbol{x_i}(\delta_m^\wp-\eta_m(\boldsymbol{x_i}))) \\ &-\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{\wp=1}^{p}\alpha_i\alpha_j\eta_\wp(\boldsymbol{x_i})\mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i},\boldsymbol{x_j})\eta_\wp(\boldsymbol{x_j})(\boldsymbol{x_i}(\delta_m^\wp-\eta_m(\boldsymbol{x_i}))+\boldsymbol{x_j}(\delta_m^\wp-\eta_m(\boldsymbol{x_j}))),\end{aligned} \tag{6.30}$$

$$\text{where}\quad \delta_m^\wp=\begin{cases}1, \text{ if } m=\wp\\ 0, \text{ otherwise.}\end{cases}$$

Once we obtain the updated values of the parameters v_{m0} and $\boldsymbol{v_m}$, we calculate the new value of $\eta_m(x)$ using the gating function.

6.2.2 Sigmoid Function

Sigmoid function is expressed in (6.13) during discussion of LMKAD. The parameters v_{m0} and $\boldsymbol{v_m}$ characterize this gating function. The gradients of the objective

function $J(\eta)$ with respect to the parameters of this gating function are:

$$\begin{aligned}\frac{\partial J(\eta)}{\partial v_{m0}} &= \sum_{i=1}^{N} \alpha_i \eta_m(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{m}}(\boldsymbol{x_i}, \boldsymbol{x_i}) \eta_m(\boldsymbol{x_i}) \times (1 - \eta_m(\boldsymbol{x_i}) + 1 - \eta_m(\boldsymbol{x_i})) \\ &- \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \eta_m(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{m}}(\boldsymbol{x_i}, \boldsymbol{x_j}) \eta_m(\boldsymbol{x_j}) (1 - \eta_m(\boldsymbol{x_i}) + 1 - \eta_m(\boldsymbol{x_j})),\end{aligned} \tag{6.31}$$

$$\begin{aligned}\frac{\partial J(\eta)}{\partial \boldsymbol{v_m}} &= \sum_{i=1}^{N} \alpha_i \eta_m(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{m}}(\boldsymbol{x_i}, \boldsymbol{x_i}) \eta_m(\boldsymbol{x_i}) \times (\boldsymbol{x_i}(1 - \eta_m(\boldsymbol{x_i})) + \boldsymbol{x_i}(1 - \eta_m(\boldsymbol{x_i}))) \\ &- \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j \eta_m(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{m}}(\boldsymbol{x_i}, \boldsymbol{x_j}) \eta_m(\boldsymbol{x_j}) (\boldsymbol{x_i}(1 - \eta_m(\boldsymbol{x_i})) + \boldsymbol{x_j}(1 - \eta_m(\boldsymbol{x_j}))).\end{aligned} \tag{6.32}$$

6.2.3 Radial Basis Function (RBF)

RBF function is expressed in (6.16) during discussion of LMKAD. This function is characterized by the parameters $\boldsymbol{\mu_m}$ and σ_m. Here, $\boldsymbol{\mu_m}$ is the center and σ_m gives the spread of the local region. Similar as above, the gradients of the objective function $J(\eta)$ with respect to the parameters of the gating function are:

$$\begin{aligned}\frac{\partial J(\eta)}{\partial \boldsymbol{\mu_m}} =& 2\Bigg(\sum_{i=1}^{N} \sum_{\wp=1}^{p} \alpha_i \eta_\wp(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i}, \boldsymbol{x_i}) \eta_\wp(\boldsymbol{x_i}) ((\boldsymbol{x_i} - \boldsymbol{\mu_m}) \times (\delta_m^\wp - \eta_m(\boldsymbol{x_i})) + \\ & (\boldsymbol{x_i} - \boldsymbol{\mu_m})(\delta_m^\wp - \eta_m(\boldsymbol{x_i})))/\sigma_m^2 - \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\wp=1}^{p} \alpha_i \alpha_j \eta_\wp(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i}, \boldsymbol{x_j}) \eta_\wp(\boldsymbol{x_j}) ((\boldsymbol{x_i} - \\ & \boldsymbol{\mu_m}) \times (\delta_m^\wp - \eta_m(\boldsymbol{x_i})) + (\boldsymbol{x_j} - \boldsymbol{\mu_m})(\delta_m^\wp - \eta_m(\boldsymbol{x_j})))/\sigma_m^2 \Bigg),\end{aligned} \tag{6.33}$$

$$\begin{aligned}\frac{\partial J(\eta)}{\partial \sigma_m} =& 2\Bigg(\sum_{i=1}^{N} \sum_{\wp=1}^{p} \alpha_i \eta_\wp(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i}, \boldsymbol{x_i}) \eta_\wp(\boldsymbol{x_i}) (\|\boldsymbol{x_i} - \boldsymbol{\mu_m}\|_2^2 \times (\delta_m^\wp - \eta_m(\boldsymbol{x_i})) + \\ & \|\boldsymbol{x_i} - \boldsymbol{\mu_m}\|_2^2 (\delta_m^\wp - \eta_m(\boldsymbol{x_i})))/\sigma_m^3 - \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{\wp=1}^{p} \alpha_i \alpha_j \eta_\wp(\boldsymbol{x_i}) \mathbb{K}_{\boldsymbol{\wp}}(\boldsymbol{x_i}, \boldsymbol{x_j}) \eta_\wp(\boldsymbol{x_j}) (\|\boldsymbol{x_i} - \\ & \boldsymbol{\mu_m}\|_2^2 \times (\delta_m^\wp - \eta_m(\boldsymbol{x_i})) + \|\boldsymbol{x_j} - \boldsymbol{\mu_m}\|_2^2 (\delta_m^\wp - \eta_m(\boldsymbol{x_j})))/\sigma_m^3 \Bigg),\end{aligned} \tag{6.34}$$

$$\text{where} \quad \delta_m^{\wp} = \begin{cases} 1, \text{ if } m = \wp \\ 0, \text{ otherwise.} \end{cases}$$

Overall, we propose 12 variants (6 variants of LMKAD, and 6 variants of LMSVDD) using different types of kernels and gating functions. Three types of commonly used kernels are used, namely linear (l), polynomial (p), and Gaussian (g) kernel, . Many combinations are possible with these kernels, such as ppp, ggg, ppl, gpl, and gpp. In our discussion, we use only those two kernel combinations (gpl and gpp), which exhibit better performance. In a multi-kernel-based classifier, these kernel combinations can be tied up with any of the above three discussed gating functions, such as Softmax (So), Sigmoid (S), and RBF (R). By following these combinations, we generate 12 variants of the proposed classifiers LMKAD and LMSVDD. The naming convention for the proposed variants is as follows: we combine the kernel name with the gating function name, and mention in a bracket with the proposed method name. By following this naming convention, 6 variants of LMKAD, and 6 variants of LMSVDD are generated, namely LMKAD(S_gpl), LMKAD(S_gpp), LMKAD(So_gpl), LMKAD(So_gpp), LMKAD(R_gpl), LMKAD(R_gpp), LMSVDD(S_gpl), LMSVDD(S_gpp), LMSVDD(So_gpl), LMSVDD(So_gpp), LMSVDD(R_gpl), and LMSVDD(R_gpp).

6.3 Experiments

In this chapter, we follow the same experimental setup and datasets, as mentioned in Section 3.2 of Chapter 3. Until this point in this thesis, we have proposed 23 variants from 5 proposed classifiers (viz., AEKOC (Chapter 3), MKOC (Chapter 4), GMKOC (Chapter 5), LMKAD, and LMSVDD). In order to compare our proposed methods with the existing methods, we have conducted experimentation on various existing state-of-the-art kernel-based OCC methods, such as (a) 8 variants of single kernel learning methods viz., OCSVM [1], SVDD [1], KPCA [4], KOC [7], GKOC-LE [8], GKOC-LLE [8], GKOC-LDA [8], and GKOC-CDA [8]. (b) 2 variants of multi-kernel learning method MKAD viz., MKAD(gpl), MKAD(gpp). Overall, we perform comparison among 33 kernel-based OCC methods, which is presented next.

As we are comparing 33 variants of one-class classifiers, hence, it is not possible to fit all results on a single page. Therefore, we are presenting results in two tables (Table 6.1 and Table 6.2). The best η_g values are kept in bold in both tables. Table 6.1 contains the results of 12 variants of the 2 proposed methods in this chapter, 4 vanilla single kernel learning-based classifiers, and 2 variants of vanilla MKL-based classifiers. After analyzing this table, it is observed that LMSVDD(So_gpp) yields the best η_g for the maximum number of datasets, i.e., 10 out of 23 datasets. Table 6.3 shows that no. of datasets with the best η_g value for each classifier. This table shows that the proposed 12 variants of LMKAD and LMSVDD collectively yield the best results for 20 out of 23 datasets. Out of 20 best results, LMKAD-based and LMSVDD-based variants yield the best results for 6 and 14 datasets, respectively. Existing classifiers yield the best results for only 3 out of 23 datasets. Out of 3 datasets, MKAD-based variants yield the best results for 2 datasets. Only one single kernel learning-based classifier manages to yield the best results for one dataset. The above discussion shows that the clear dominance of MKL-based classifiers. Overall, it can be stated that the proposed localized MKL-based classifiers outperform all single and multi-kernel learning-based existing classifiers for most of the datasets. Furthermore, when we compare the results of various kernel combinations for the proposed methods in Table 6.3, gpp kernel combination yields the best outcome among all combinations for both LMKAD and LMSVDD. The gpp kernel combination attains a top position with Sigmoid and Softmax gating function for LMKAD and LMSVDD, respectively.

Table 6.2 presents the results of the all proposed methods discussed so far in this thesis. In this table, AEKOC is a single hidden layer-based classifier, and remaining all methods either belong to multi-layer or multi-kernel learning. We observe in this table that LMSVDD(So_gpp) is still a top performer and yields the best η_g for the maximum number of datasets, i.e., 6 out of 23 datasets. Table 6.3 shows that no. of datasets with the best η_g value for each classifier. MKL-based proposed methods collectively yield the best results for 17 datasets. It again shows the clear dominance of MKL-based methods.

In Table 6.2, the best performing classifier LMSVDD(So_gpp) yields the best η_g

Table 6.1: Performance comparison among existing (vanilla single and multi-kernel-based) and proposed classifiers of this chapter in terms of η_g for 23 datasets

Datasets	KPCA	OCSVM	SVDD	KOC	MKAD (gpl)	MKAD (gpp)	LMKAD (S_gpl)	LMKAD (S_gpp)	LMKAD (So_gpl)	LMKAD (So_gpp)	LMKAD (R_gpl)	LMKAD (R_gpp)	LMSVDD (S_gpl)	LMSVDD (S_gpp)	LMSVDD (So_gpl)	LMSVDD (So_gpp)	LMSVDD (R_gpl)	LMSVDD (R_gpp)
Aust(1)	63.69	66.08	65.55	65.07	66.57	66.57	66.43	66.16	66.44	66.67	66.69	66.64	66.61	66.58	78.53	**78.91**	68.56	71.88
Aust(2)	73.06	76.59	76.78	74.21	75.55	75.80	76.48	77.13	76.67	76.66	76.56	76.64	75.25	74.94	75.06	**77.54**	75.68	73.40
Bupa(1)	62.91	60.64	60.64	57.09	64.83	64.87	64.97	**65.12**	64.79	64.81	64.97	65.00	64.78	64.50	61.77	63.10	61.89	61.12
Bupa(2)	74.28	69.78	69.75	68.81	**76.14**	**76.14**	75.67	75.68	75.49	75.67	75.80	75.87	75.74	75.78	75.98	74.19	73.84	75.70
Ecoli(1)	65.79	89.43	89.49	89.38	66.41	66.57	83.11	84.96	78.47	80.34	76.75	78.61	66.30	67.26	89.58	**89.77**	85.67	88.38
Ecoli(2)	72.30	79.42	78.87	82.32	75.79	75.79	76.23	76.65	75.83	75.86	75.83	75.83	76.04	75.58	80.87	85.14	83.90	**85.22**
Germ(1)	80.77	80.34	81.10	73.17	**83.67**	83.64	83.13	82.76	83.21	83.24	83.37	83.25	83.55	**83.63**	83.05	83.54	82.74	80.16
Germ(2)	49.75	52.80	52.77	53.41	54.67	54.67	54.50	54.27	55.09	54.74	54.72	54.76	54.63	**55.11**	53.73	51.02	49.23	44.98
Glass(1)	57.69	59.61	59.61	58.91	59.59	59.62	60.49	61.06	59.90	60.01	59.53	59.88	60.50	62.04	61.01	**62.65**	59.80	57.60
Glass(2)	77.65	73.32	72.89	73.08	80.30	80.30	80.43	80.83	80.20	80.48	80.29	**80.33**	80.57	79.49	77.57	78.35	78.57	79.42
Heart(1)	70.42	72.91	72.91	65.03	73.40	73.40	73.32	73.60	73.30	73.34	73.36	73.36	72.89	74.58	74.01	**76.63**	64.17	66.97
Heart(2)	63.50	64.90	64.90	66.39	67.91	67.91	67.68	67.78	67.20	67.12	67.91	67.91	67.89	68.04	71.18	**73.87**	53.74	54.50
Iono(1)	76.54	93.13	93.13	92.69	83.12	86.20	93.99	**95.08**	89.64	89.49	88.68	89.04	84.09	85.98	88.19	89.54	79.70	90.85
Iono(2)	57.10	44.63	44.63	53.40	59.91	59.91	60.03	59.91	**60.10**	59.84	59.92	59.83	58.71	57.75	57.26	55.29	55.85	56.45
Iris(1)	96.44	85.06	84.12	92.35	57.74	57.74	99.81	99.81	**100.00**	99.19	**100.00**	99.61	92.51	92.75	93.01	99.14	86.19	93.24
Iris(2)	72.99	81.70	81.70	**85.59**	57.74	57.74	76.76	78.07	69.96	70.99	62.33	63.14	69.89	70.62	74.99	79.04	81.22	80.81
Iris(3)	69.29	83.18	82.67	83.40	57.74	57.74	80.09	78.97	82.11	81.09	68.29	68.33	78.11	82.12	88.51	**90.49**	84.85	85.53
Jap(1)	64.09	71.45	70.15	67.33	66.99	66.95	66.96	67.31	67.07	67.02	67.09	67.10	67.39	68.28	82.09	**82.45**	70.28	68.74
Jap(2)	72.29	75.78	76.58	73.48	74.87	75.13	75.83	76.56	76.19	76.23	76.22	76.22	74.67	74.79	80.60	**81.29**	76.69	74.19
Park(1)	70.20	95.71	97.07	96.15	94.27	96.72	**99.46**	99.26	94.39	94.10	93.78	94.13	71.34	81.88	89.55	94.60	90.83	98.43
Park(2)	67.79	81.82	79.77	90.74	70.88	71.53	82.51	85.27	71.74	71.67	71.28	71.32	93.42	**95.00**	74.98	77.58	78.29	82.18
Pima(1)	77.98	79.18	79.21	79.04	80.59	80.61	80.69	80.85	80.84	80.69	80.88	80.89	80.57	80.63	**81.27**	80.23	80.51	80.89
Pima(2)	57.05	56.59	56.71	54.78	59.07	58.96	58.44	59.75	59.02	59.63	58.60	58.52	59.05	59.30	61.17	**64.06**	58.85	59.37
Mean of all η_g (η_m)	69.28	73.65	73.52	73.73	69.90	70.20	75.52	75.95	74.25	74.30	73.17	73.31	72.80	73.77	76.26	**77.76**	73.09	74.35

Table 6.2: Performance comparison among all proposed classifiers until this chapter in terms of η_g for 23 datasets

Datasets	AEKOC	MKOC_#1	MKOC_#2	GMKOC-LDA_θ1	GMKOC-LDA_θ2	GMKOC-CDA_θ1	GMKOC-CDA_θ2	GMKOC-LE_θ1	GMKOC-LE_θ2	GMKOC-LLE_θ1	GMKOC-LLE_θ2	LMKAD (S_gpl)	LMKAD (S_gpp)	LMKAD (So_gpl)	LMKAD (So_gpp)	LMKAD (R_gpl)	LMKAD (R_gpp)	LMSVDD (S_gpl)	LMSVDD (S_gpp)	LMSVDD (So_gpl)	LMSVDD (So_gpp)	LMSVDD (R_gpl)	LMSVDD (R_gpp)
Aust(1)	72.88	72.12	72.43	72.24	72.62	77.55	79.68	72.33	72.61	77.81	**81.70**	66.43	66.16	66.44	66.67	66.69	66.64	66.61	66.58	78.53	78.91	68.56	71.88
Aust(2)	77.96	79.89	80.31	79.48	80.32	79.55	**80.49**	79.78	80.27	75.38	79.85	76.48	77.13	76.67	76.66	76.56	76.64	75.25	74.94	75.06	77.54	75.68	73.40
Bupa(1)	56.19	62.55	62.19	63.06	62.77	62.13	63.07	62.59	62.21	61.69	62.04	64.97	**65.12**	64.79	64.81	64.97	65.00	64.78	64.50	61.77	63.10	61.80	61.12
Bupa(2)	68.31	73.56	73.61	73.52	73.07	72.83	75.31	73.49	73.80	71.65	73.76	75.67	75.68	75.49	75.67	75.80	75.87	75.74	75.78	**75.98**	74.19	73.84	75.70
Ecoli(1)	89.00	82.51	80.36	92.22	89.70	92.28	90.06	**92.75**	88.71	89.13	88.66	83.11	84.96	78.47	80.34	76.75	78.61	66.30	67.26	89.58	89.77	85.67	88.38
Ecoli(2)	79.16	84.42	81.76	85.75	82.22	86.32	82.77	85.00	**87.24**	84.79	81.95	76.23	76.65	75.83	75.86	75.83	75.83	76.04	75.58	80.87	85.14	83.90	85.22
Germ(1)	74.04	74.10	82.79	73.58	82.76	73.49	81.65	75.45	82.70	74.04	82.19	83.13	82.76	83.21	83.24	83.37	83.25	83.55	**83.63**	83.05	83.54	82.74	80.16
Germ(2)	51.57	54.46	49.59	54.31	50.09	53.37	49.56	54.23	50.64	51.89	49.53	54.50	54.27	55.09	54.74	54.72	54.76	54.63	**55.11**	53.73	51.02	49.23	44.98
Glass(1)	59.29	62.46	59.23	62.52	59.97	62.58	61.29	62.07	60.45	59.66	61.45	60.49	61.06	59.90	60.01	59.53	59.88	60.50	62.04	61.01	**62.65**	59.80	57.60
Glass(2)	73.22	77.15	75.69	78.23	77.02	76.84	78.23	78.44	79.46	76.59	78.45	80.43	**80.83**	80.20	80.48	80.29	80.33	80.57	79.49	77.57	78.35	78.57	79.42
Heart(1)	67.99	71.72	73.93	71.40	74.52	71.15	74.25	72.06	74.28	67.53	70.28	73.32	73.60	73.30	73.34	73.36	73.36	72.89	74.58	74.01	**76.63**	64.17	66.97
Heart(2)	61.15	68.89	59.51	67.91	60.09	69.86	58.79	67.89	60.08	64.30	62.64	67.68	67.78	67.20	67.12	67.91	67.91	67.89	68.04	71.18	**73.87**	53.74	54.50
Iono(1)	92.95	89.54	88.77	88.85	88.41	88.81	89.81	88.77	88.32	88.55	90.73	93.99	**95.08**	89.64	89.49	88.68	89.04	84.09	85.98	88.19	89.54	79.70	90.85
Iono(2)	41.04	**67.70**	60.78	59.91	42.39	67.50	65.02	59.91	39.64	60.30	49.28	60.03	59.91	60.10	59.84	59.92	59.83	58.71	57.75	57.26	55.29	55.85	56.45
Iris(1)	92.79	99.59	93.87	98.33	94.91	98.33	95.71	97.09	95.80	99.59	95.36	99.81	99.81	**100.00**	99.19	**100.00**	99.61	92.51	92.75	93.01	99.14	86.19	93.24
Iris(2)	88.37	77.20	74.07	75.93	72.85	76.95	78.79	74.35	76.79	90.26	**90.90**	76.76	78.07	69.96	70.99	62.33	63.14	69.89	70.62	74.99	79.04	81.22	80.81
Iris(3)	83.74	71.29	69.82	70.24	71.32	71.39	70.88	64.40	71.30	86.67	84.96	80.09	78.97	82.11	81.09	68.29	68.33	78.11	82.12	88.51	**90.49**	84.85	85.53
Jap(1)	76.23	74.45	74.66	74.60	74.92	77.91	80.13	75.03	75.20	80.80	**83.72**	66.96	67.31	67.07	67.02	67.09	67.10	67.39	68.28	82.09	82.45	70.28	68.74
Jap(2)	78.27	80.14	80.55	79.91	80.54	79.52	80.72	80.08	80.53	74.63	78.58	75.83	76.56	76.19	76.23	76.22	76.22	74.67	74.79	80.60	**81.29**	76.69	74.19
Park(1)	96.67	91.48	91.70	91.36	91.87	90.93	92.86	90.59	91.68	91.40	92.40	**99.46**	99.26	94.39	94.10	93.78	94.13	71.34	81.88	89.55	94.60	90.83	98.43
Park(2)	83.78	80.97	79.65	81.35	79.72	81.35	79.72	81.06	79.43	80.95	79.57	82.51	85.27	71.74	71.67	71.28	71.32	93.42	**95.00**	74.98	77.58	78.29	82.18
Pima(1)	78.66	79.21	80.50	79.83	80.60	79.71	80.52	79.79	80.41	78.20	80.15	80.69	80.85	80.84	80.69	80.88	80.89	80.57	80.63	**81.27**	80.23	80.51	80.89
Pima(2)	54.01	57.70	57.80	58.00	57.96	58.92	58.44	57.69	57.95	56.21	57.85	58.44	59.75	59.02	59.63	58.60	58.52	59.05	59.30	61.17	**64.06**	58.85	59.37
Mean of all η_g (η_m)	73.79	75.35	74.07	75.33	73.94	76.06	75.99	74.99	74.33	75.74	**76.35**	75.52	75.95	74.25	74.30	73.17	73.31	72.80	73.77	76.26	**77.76**	73.09	74.35

Table 6.3: Number of datasets for which each one-class classifier yields the best η_g

6.3(a) Among existing (vanilla single and multi-kernel-based) and MKL-based one-class classifiers

One-class Classifiers	No. of datasets with the best η_g value
LMSVDD(So_gpp)	10
LMKAD(S_gpp)	3
LMKAD(So_gpl)	2
LMSVDD(S_gpp)	2
MKAD(gpl)	2
LMKAD(S_gpl)	1
LMKAD(R_gpl)	1
LMSVDD(So_gpl)	1
LMSVDD(R_gpp)	1
MKAD(gpp)	1
KOC	1
LMKAD(So_gpp)	0
LMKAD(R_gpp)	0
LMSVDD(S_gpl)	0
LMSVDD(R_gpl)	0
KPCA	0
OCSVM	0
SVDD	0

6.3(b) Among all proposed one-class classifiers of this thesis so far

One-class Classifiers	No. of datasets with best η_g value
LMSVDD (So_gpp)	6
LMKAD (S_gpp)	3
LMSVDD (S_gpp)	3
GMKOC-LLE_θ2	3
LMSVDD (So_gpl)	2
LMKAD (S_gpl)	1
LMKAD (So_gpl)	1
LMKAD (R_gpl)	1
GMKOC-CDA_θ2	1
GMKOC-LE_θ1	1
GMKOC-LE_θ2	1
MKOC_θ1	1
MKOC_θ2	0
AEKOC	0
GMKOC-LDA_θ1	0
GMKOC-LDA_θ2	0
GMKOC-CDA_θ1	0
GMKOC-LLE_θ1	0
LMKAD (So_gpp)	0
LMKAD (R_gpp)	0
LMSVDD (S_gpl)	0
LMSVDD (R_gpl)	0
LMSVDD (R_gpp)	0

for only 26.09% of datasets. Therefore, we can't decide the best performer just by looking at the values of η_g in Table 6.1 and Table 6.2. We need some other performance criteria to decide the best performing classifier. As per discussion in Chapter 3, we compute three criteria [167] for further analysis: (i) average of η_g (η_m) (ii) Friedman test (F-score and p-value) (iii) Friedman rank (η_f). Based on these three criteria, we analyze the performance of all 33 variants of the classifiers. The analysis is provided in the following points:

(i) We compute η_m for analyzing the combined performance of the classifier. η_m value of each classifier is available in the last row of Table 6.1. It is also provided in Table 6.4(a) in decreasing order of η_m. When we compare η_m values in this table, it is observed that LMSVDD(So_gpp) attains the top position and yields a maximum η_m value among all proposed and existing classifiers. The second position is held by GMKOC-LLE_θ2. The difference between the η_m values of the first and second positions is 1.41, which is quite significant. However, difference between the η_m values of 2^{nd} and 3^{rd} position of classifier is only 0.09. None of the LMKAD or MKOC variants or any existing classifiers get a position in the top 5 classifiers in Table 6.4(a). Similar to the above discussion, gpp kernel combination with Sigmoid and Softmax gating function again yields the best results for both proposed classifiers LMKAD and LMSVDD, respectively.

(ii) Although proposed localized multi-kernel based classifiers attain a top position among all classifiers in Table 6.4(a); however, we need to verify the outcomes of the proposed and existing classifiers statistically. For this purpose, we conduct a non-parametric Friedman test [166] similar as discussed in Section 3.2 of Chapter 3. Friedman test mainly computes three components viz., p-value, F-score, and critical value. The computed p-value, F-score value and critical value are $3.8725e-10$, 107.7905 and 46.1943, respectively. Since the computed F-score value is higher than the critical value, and the p-value is significantly lesser than the tolerance level 0.05, we reject the null hypothesis, with 95% of confidence. Therefore, we conclude that the outcomes presented in this chapter are statistically significant.

(iii) Further, we compute Friedman Rank (η_f) [166] for each classifier as discussed in Section 2.8 of Chapter 2. We consider η_f as the final decision criteria to decide the rank of any classifiers. In Table 6.4(b), we present η_f along with η_m of each classifier in increasing order of η_f. In this table, similar as in Table 6.4(a), LMSVDD(So_gpp) again attains top position among all classifiers. LMSVDD(R_gpl) is the least performer among all multi-layer and multi-kernel based one-class classifiers. According to η_m criteria, 3 multi-layer classifiers are in top 5; however, only one multi-layer classifier get position in top 5 as per η_f criteria.

Overall, the localized mkl-based classifier performed very well according to various tested criteria. Proposed methods achieve this performance by combining the Gaussian kernel with less complex kernel like polynomial and linear kernel. It is to be noted that different types of kernels work in a different part of the datasets. If some part of the datasets is non-linearly separable; then, the Gaussian kernel will have more weight in that area compared to the linear kernel. The linear kernel will get more importance in that area where the dataset is linearly separable. Even some of the kernels might contain zero weights for a dataset if that is not contributing to classification. Moreover, localization has helped the classifier in achieving better performance.

6.4 Summary

In this chapter, two OCC methods, LMKAD and LMSVDD, are proposed by taking OCSVM and SVDD as a base classifier, respectively. These methods are extensions of MKAD. Proposed methods provide a localized formulation for the multi-kernel learning method by local assignment of weights to each kernel. Local assignments of weights are achieved by the training of a gating function and a conventional OCSVM/SVDD with the combined kernel in tandem. The derived formulation is also shown to be analogous to conventional OCSVM/SVDD and is solved in a similar fashion using a LIBSVM solver. Overall, we have proposed 12 variants (6 variants of LMKAD, and 6 variants of LMSVDD) using different types of kernels and gating functions. The

Table 6.4: Friedman Rank (η_f) and mean of η_g (η_m) among all discussed proposed and existing one-class classifiers in this thesis until this chapter

6.4(a) In decreasing order of η_m

	η_f	η_m
LMSVDD (So_gpp)	9.48	77.76
GMKOC-LLE_θ2	15.22	76.35
LMSVDD (So_gpl)	12.91	76.26
GMKOC-CDA_θ1	13.74	76.06
GMKOC-CDA_θ2	12.89	75.99
LMKAD (S_gpp)	11.02	75.95
GMKOC-LLE_θ1	17.72	75.74
LMKAD (S_gpl)	13.15	75.52
MKOC_θ1	14.72	75.35
GMKOC-LDA_θ1	14.43	75.33
GMKOC-LE_θ1	15.35	74.99
LMSVDD (R_gpp)	16.96	74.35
GMKOC-LE_θ2	16.26	74.33
LMKAD (So_gpp)	14.74	74.30
LMKAD (So_gpl)	14.98	74.25
MKOC_θ2	17.83	74.07
GMKOC-LDA_θ2	16.59	73.94
AEKOC	18.96	73.79
LMSVDD (S_gpp)	15.22	73.77
KOC	20.43	73.73
OCSVM	19.33	73.65
SVDD	19.11	73.52
GKOC-CDA	20.98	73.34
LMKAD (R_gpp)	15.17	73.31
LMKAD (R_gpl)	15.85	73.17
LMSVDD (R_gpl)	19.70	73.09
GKOC-LE	21.57	73.09
LMSVDD (S_gpl)	16.91	72.80
GKOC-LDA	22.76	72.72
GKOC-LLE	24.48	72.14
MKAD (gpp)	17.63	70.20
MKAD (gpl)	18.50	69.90
KPCA	26.43	69.28

6.4(b) In increasing order of η_f

	η_f	η_m
LMSVDD (So_gpp)	9.48	77.76
LMKAD (S_gpp)	11.02	75.95
GMKOC-CDA_θ2	12.89	75.99
LMSVDD (So_gpl)	12.91	76.26
LMKAD (S_gpl)	13.15	75.52
GMKOC-CDA_θ1	13.74	76.06
GMKOC-LDA_θ1	14.43	75.33
MKOC_θ1	14.72	75.35
LMKAD (So_gpp)	14.74	74.30
LMKAD (So_gpl)	14.98	74.25
LMKAD (R_gpp)	15.17	73.31
GMKOC-LLE_θ2	15.22	76.35
LMSVDD (S_gpp)	15.22	73.77
GMKOC-LE_θ1	15.35	74.99
LMKAD (R_gpl)	15.85	73.17
GMKOC-LE_θ2	16.26	74.33
GMKOC-LDA_θ2	16.59	73.94
LMSVDD (S_gpl)	16.91	72.80
LMSVDD (R_gpp)	16.96	74.35
MKAD (gpp)	17.63	70.20
GMKOC-LLE_θ1	17.72	75.74
MKOC_θ2	17.83	74.07
MKAD (gpl)	18.50	69.90
AEKOC	18.96	73.79
SVDD	19.11	73.52
OCSVM	19.33	73.65
LMSVDD (R_gpl)	19.70	73.09
KOC	20.43	73.73
GKOC-CDA	20.98	73.34
GKOC-LE	21.57	73.09
GKOC-LDA	22.76	72.72
GKOC-LLE	24.48	72.14
KPCA	26.43	69.28

proposed LMSVDD-based variant has achieved the best η_m and η_f among all mentioned one-class classifiers in this thesis so far. We have statistically verified that the outcomes presented in this chapter are statistically significant, with 95% of confidence. Moreover, the p-value is significantly lesser than the tolerance level of 0.05.

At this point in this thesis, we have developed mainly 5 one-class classifiers with their 23 variants. However, all proposed methods in this thesis so far are unable to handle two types of data (or information): (i) Privileged information, (ii) Non-stationary data. Our next two chapters are focused on handling these two types of data by taking KOC and AEKOC as base classifiers.

Chapter 7

Learning Using Privileged Information Framework with Kernel Ridge Regression for One-class Classification

In the real world, additional/privileged information generally exists with training samples [13]. However, all proposed classifiers so far in this thesis are unable to utilize this information because they follow the traditional classification setting, which does not utilize the additional/privileged information in building the classification model. Vapnik and Vashist [13] addressed this issue by proposing a novel framework, i.e., learning using privileged information (LUPI). This framework enables classifiers to utilize privileged information, which is generally ignored by any traditional machine learning setting-based classifiers but usually present in human learning. LUPI is briefly discussed in Section 2.6 of Chapter 2. In this chapter, we develop two types of LUPI framework-based one-class classifiers by taking KOC [7] and AEKOC as base classifiers. Both proposed methods are referred to as KOC+ and AEKOC+, and one is the boundary and the other is reconstruction framework-based one-class classifiers, respectively. Both proposed methods are discussed in the subsequent sections.

7.1 LUPI framework with KOC: KOC+

In this section, the optimization problem of KOC [7] is modified according to the LUPI framework, and we develop a minimization problem for KOC+. First, we provide the minimization problem of KOC because we are using this optimization problem for developing the proposed classifier.

Let us assume the input training matrix of size $N \times n$ is $\boldsymbol{X} = \{\boldsymbol{x_i}\}$, where $\boldsymbol{x_i} = [x_{i1}, x_{i2}, ..., x_{in}]$, $i = 1, 2, ..., N$, is the n-dimensional input vector of the i^{th} training sample. The minimization function of KOC [7] is as follows:

$$\begin{aligned} \underset{\boldsymbol{\beta}, e_i}{\text{Minimize}} : \pounds_{KOC} &= \frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\left\|e_i\right\|_2^2 \\ \text{Subject to} : \ & \boldsymbol{\phi_i} \cdot \boldsymbol{\beta} = r - e_i, \ i = 1, 2, ..., N, \end{aligned} \tag{7.1}$$

where $\boldsymbol{\beta}$ denotes weight matrix, $\phi(.)$ denotes kernel feature mapping function, $\boldsymbol{\phi_i} = \boldsymbol{\phi(x_i)}$, and $\boldsymbol{\Phi} = \boldsymbol{\Phi(X)} = [\boldsymbol{\phi_1}, \boldsymbol{\phi_2}, ..., \boldsymbol{\phi_N}]$. $\boldsymbol{E}$ is an error vector where $\boldsymbol{E} = \{e_i\}$, where $i = 1, 2, ..., N$. Here, $\boldsymbol{r}$ is a vector having all elements equal to r, and r is a real number, which is set at equal to 1.

Before discussing the minimization problem of KOC+, we make some assumptions regarding the LUPI framework. We assume that privileged information $\boldsymbol{X^*} = \{\boldsymbol{x_i^*}\}$, where $i = 1, 2, ..., N$, is available with feature space of X as $(\boldsymbol{x_i}, \boldsymbol{x_i^*})$ during training. However, this information is not available during testing. As suggested in [13], privileged information is incorporated to the optimization problem by modeling the slack variable e_i as so called correction function:

$$e_i = \boldsymbol{e_i(x_i^*)} = \boldsymbol{\phi^*(x_i^*).\beta^*} = \boldsymbol{\phi_i^*.\beta^*}, \tag{7.2}$$

where $\boldsymbol{\phi^*(.)}$ is a feature mapping in the privileged space and $\boldsymbol{\beta^*}$ is a correction weight. Now, we substitute (7.2) into (7.1) and modify the optimization problem for KOC+

as follows:

$$\underset{\boldsymbol{\beta},\boldsymbol{\beta^*},e_i}{\text{Minimize}} : \pounds_{KOC+} = \frac{1}{2}\left\|\boldsymbol{\beta}\right\|^2 + \mu\frac{1}{2}\left\|\boldsymbol{\beta^*}\right\|^2 + C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\phi_i^*}\cdot\boldsymbol{\beta^*})^2$$
$$\text{Subject to :}\ \ \boldsymbol{\phi_i}\cdot\boldsymbol{\beta} = r - \boldsymbol{\phi_i^*}\cdot\boldsymbol{\beta^*},\ \ i = 1, 2, ..., N, \tag{7.3}$$

where $\left\|\boldsymbol{\beta}\right\|^2$ reflects the capacity of the decision function and $\left\|\boldsymbol{\beta^*}\right\|^2$ reflects the capacity of the correction function. Here, μ controls the capacity of these two functions i.e. controls the relative weight of these two capacities.

By using Representer Theorem [103], $\boldsymbol{\beta}$ can be expressed as a linear combination of the training data representation in non-linear feature space $\boldsymbol{\Phi}$ and reconstruction weight vector $\boldsymbol{W}$:

$$\boldsymbol{\beta} = \boldsymbol{\Phi}\cdot\boldsymbol{W} \quad \text{and} \quad \boldsymbol{\beta*} = \boldsymbol{\Phi^*}\cdot\boldsymbol{W^*}. \tag{7.4}$$

By substituting the (7.4) into (7.3), following minimization problem is obtained:

$$\underset{\boldsymbol{W},\boldsymbol{W^*},e_i}{\text{Minimize}} : \pounds_{KOC+} = \frac{1}{2}\boldsymbol{\Phi}\cdot\boldsymbol{\Phi}\cdot\left\|\boldsymbol{W}\right\|^2 + \mu\frac{1}{2}\boldsymbol{\Phi^*}\cdot\boldsymbol{\Phi^*}\cdot\left\|\boldsymbol{W^*}\right\|^2 + C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\phi_i^*}\cdot\boldsymbol{\Phi_i^*}\cdot\boldsymbol{W^*})^2$$
$$\text{Subject to :}\ \ \boldsymbol{\phi_i}\cdot\boldsymbol{\Phi_i}\cdot\boldsymbol{W} = r - \boldsymbol{\phi_i^*}\cdot\boldsymbol{\Phi_i^*}\cdot\boldsymbol{W^*},\ \ i = 1, 2, ..., N. \tag{7.5}$$

Further, we substitute $\boldsymbol{K} = \boldsymbol{\Phi}\cdot\boldsymbol{\Phi}$, $\boldsymbol{K^*} = \boldsymbol{\Phi^*}\cdot\boldsymbol{\Phi^*}$, $\boldsymbol{k_i} = \boldsymbol{\phi_i}\cdot\boldsymbol{\Phi_i}$, and $\boldsymbol{k_i^*} = \boldsymbol{\phi_i^*}\cdot\boldsymbol{\Phi_i^*}$ in (7.5). Now, (7.5) is reformulated as follows:

$$\underset{\boldsymbol{W},\boldsymbol{W^*},e_i}{\text{Minimize}} : \pounds_{KOC+} = \frac{1}{2}\boldsymbol{K}\cdot\left\|\boldsymbol{W}\right\|^2 + \mu\frac{1}{2}\boldsymbol{K^*}\cdot\left\|\boldsymbol{W^*}\right\|^2 + C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{k_i^*}\cdot\boldsymbol{W^*})^2$$
$$\text{Subject to :}\ \ \boldsymbol{k_i}\cdot\boldsymbol{W} = r - \boldsymbol{k_i^*}\cdot\boldsymbol{W^*},\ \ i = 1, 2, ..., N, \tag{7.6}$$

where $\boldsymbol{k_i} \subseteq \boldsymbol{K}$ and $\boldsymbol{k_i^*} \subseteq \boldsymbol{K^*}$.

The Lagrangian relaxation of (7.6) is written as follows:

$$\underset{\alpha_i}{\text{Maximize}} \quad \underset{\boldsymbol{W},\boldsymbol{W^*},e_i}{\text{Minimize}} : \pounds_{KOC+} = \frac{1}{2}\boldsymbol{K} \cdot \|\boldsymbol{W}\|^2 + \mu\frac{1}{2}\boldsymbol{K^*} \cdot \|\boldsymbol{W^*}\|^2 +$$
$$C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{k_i^*} \cdot \boldsymbol{W^*})^2 - \sum_{i=1}^{N}\boldsymbol{\alpha_i}(\boldsymbol{k_i} \cdot \boldsymbol{W} - r + \boldsymbol{k_i^*} \cdot \boldsymbol{W^*}), \tag{7.7}$$
$$i = 1, 2, ..., N,$$

where $\boldsymbol{\alpha} = \{\boldsymbol{\alpha_i}\}, i = 1, 2 \ldots N$, is a Lagrangian multiplier. In order to optimize (7.7), we compute the partial derivatives of (7.7) as follows:

$$\frac{\partial \pounds_{KOC+}}{\partial \boldsymbol{W}} = 0 \Rightarrow \boldsymbol{W} = \boldsymbol{\alpha}, \tag{7.8}$$

$$\frac{\partial \pounds_{KOC+}}{\partial \boldsymbol{W^*}} = 0 \Rightarrow \mu\boldsymbol{K^*} \cdot \boldsymbol{W^*} + C\boldsymbol{K^*} \cdot \boldsymbol{W^*} \cdot \boldsymbol{K^*} - \boldsymbol{\alpha} \cdot \boldsymbol{K^*}, \tag{7.9}$$

$$\frac{\partial \pounds_{KOC+}}{\partial \boldsymbol{\alpha}} = 0 \Rightarrow \boldsymbol{K} \cdot \boldsymbol{W} + \boldsymbol{K^*} \cdot \boldsymbol{W^*} = \boldsymbol{r}. \tag{7.10}$$

Now, we substitute (7.8) and (7.10) into (7.9):

$$\mu(\boldsymbol{r} - \boldsymbol{K} \cdot \boldsymbol{W}) + C(\boldsymbol{r} - \boldsymbol{K} \cdot \boldsymbol{W}) \cdot \boldsymbol{K^*} - \boldsymbol{W} \cdot \boldsymbol{K^*} = 0. \tag{7.11}$$

After solving the (7.11), W is obtained as follows:

$$\boldsymbol{W} = (\mu\boldsymbol{K} + C\boldsymbol{K} \cdot \boldsymbol{K^*} + \boldsymbol{K^*})^{-1} \cdot (\mu\boldsymbol{I} + C\boldsymbol{K^*}) \cdot \boldsymbol{r}. \tag{7.12}$$

The weight $\boldsymbol{\beta}$ is obtained by substituting (7.12) into (7.4) as follows:

$$\boldsymbol{\beta} = \boldsymbol{\Phi} \cdot (\mu\boldsymbol{K} + C\boldsymbol{K} \cdot \boldsymbol{K^*} + \boldsymbol{K^*})^{-1} \cdot (\mu\boldsymbol{I} + C\boldsymbol{K^*}) \cdot \boldsymbol{r}. \tag{7.13}$$

The predicted output of KOC+ for training samples can be calculated as, $\widehat{\boldsymbol{O}} = \boldsymbol{\Phi}.\boldsymbol{\beta} = \boldsymbol{\Phi}.\boldsymbol{\Phi}.\boldsymbol{W} = \boldsymbol{K}.\boldsymbol{W}$, where $\widehat{\boldsymbol{O}}$ is the predicted output for training data. After obtaining the predicted output value, we compute a threshold value based on the predicted value at the output layer. This threshold value helps in deciding whether a

sample is an outlier or not. It is discussed next. A threshold ($\theta 1$) is employed with the proposed method, which is determined as follows:

(i) We calculate the distance between the predicted value of the i^{th} training sample and r, and store in a vector, $\mathbf{\Lambda} = \{\Lambda_i\}$ and $i = 1, 2, ..., N$, as follows:

$$\Lambda_i = \left| \widehat{O}_i - r \right|. \tag{7.14}$$

(ii) After storing all distances in $\mathbf{\Lambda}$ as per (7.14), we sort these distances in decreasing order and denoted by a vector $\boldsymbol{\Lambda_{dec}}$. Further, we reject a few percents of training samples based on the deviation. Most deviated samples are rejected first because they are most probably far from the distribution of the target data. The threshold is decided based on these deviations as follows:

$$\theta 1 = \Lambda_{dec}(\lfloor \nu * N \rfloor), \tag{7.15}$$

where $0 < \nu \leq 1$ is the fraction of rejection of training samples for deciding threshold value. N is the number of training samples and $\lfloor\ \rfloor$ denotes floor operation.

After determining a threshold value by the above procedure, during testing, a test vector $\boldsymbol{x_p}$ is fed to the trained architecture and its output $\widehat{O}_p$ is obtained. Further, compute the distance ($\widehat{\Lambda}_p$), for $\boldsymbol{x_p}$, between the predicted value $\widehat{O}_p$ of the p^{th} testing sample and r:

$$\widehat{\Lambda}_p = \left| \widehat{O}_p - r \right|. \tag{7.16}$$

Finally, $\boldsymbol{x_p}$ is classified based on the following rule:

$$\begin{aligned} &\text{If } \widehat{\Lambda}_p \leq \theta 1, \quad \boldsymbol{x_p} \text{ belongs to normal class,} \\ &\text{Otherwise,} \qquad\qquad\quad \boldsymbol{x_p} \text{ is an outlier.} \end{aligned} \tag{7.17}$$

7.2 LUPI framework with AEKOC: AEKOC+

In this section, the optimization problem of AEKOC is modified according to the LUPI framework, and we develop a minimization problem for AEKOC+. We propose AEKOC in Chapter 3 of this thesis. First, we provide the minimization problem of AEKOC because we are using this optimization problem for developing the proposed classifier.

Let us assume the input training matrix of size $N \times n$ is $\boldsymbol{X} = \{\boldsymbol{x_i}\}$, where $\boldsymbol{x_i} = [x_{i1}, x_{i2}, ..., x_{in}]$, $i = 1, 2, ..., N$, is the n-dimensional input vector of the i^{th} training sample. The minimization function of $AEKOC$ is as follows:

$$\begin{aligned} \underset{\boldsymbol{\beta_a}, \boldsymbol{e_i}}{\text{Minimize}} : \pounds_{AEKOC} &= \frac{1}{2} \|\boldsymbol{\beta_a}\|_F^2 + C\frac{1}{2}\sum_{i=1}^{N} \|\boldsymbol{e_i}\|_2^2 \\ \text{Subject to} : \ & \boldsymbol{\phi_i} \cdot \boldsymbol{\beta_a} = \boldsymbol{x_i} - \boldsymbol{e_i}, \ i = 1, 2, ..., N, \end{aligned} \tag{7.18}$$

where $\boldsymbol{\beta_a}$ denotes weight matrix, $\phi(.)$ denotes kernel feature mapping function, $\boldsymbol{\phi_i} = \boldsymbol{\phi(x_i)}$, and $\boldsymbol{\Phi} = \boldsymbol{\Phi(X)} = [\boldsymbol{\phi_1}, \boldsymbol{\phi_2}, ..., \boldsymbol{\phi_N}]$. $\boldsymbol{E}$ is an error matrix where $\boldsymbol{E} = \{\boldsymbol{e_i}\}$, where $i = 1, 2, ..., N$.

Let us do some assumption as per $LUPI$ framework. We assume that privileged information $\boldsymbol{X^* = \{x_i^*\}}$, where $i = 1, 2, ..., N$, is available with feature space of X as $(\boldsymbol{x_i}, \boldsymbol{x_i^*})$ during training. However, this information is not available during testing. As suggested in [13], privileged information is incorporated into the optimization problem by modeling the slack variable e_i as so-called correction function:

$$e_i = \boldsymbol{e_i(x_i^*) = \phi^*(x_i^*).\beta_a^* = \phi_i^*.\beta_a^*}, \tag{7.19}$$

. where $\boldsymbol{\phi^*(.)}$ is a feature mapping in the privileged space and $\boldsymbol{\beta_a^*}$ is a correction weight. Now, we substitute (7.19) into (7.18) and modify the optimization problem

for AEKOC+ as follows:

$$\underset{\boldsymbol{\beta_a},\boldsymbol{\beta_a^*},\boldsymbol{e_i}}{\text{Minimize}} : \pounds_{AEKOC+} = \frac{1}{2}\left\|\boldsymbol{\beta_a}\right\|_F^2 + \mu\frac{1}{2}\left\|\boldsymbol{\beta_a^*}\right\|_F^2 + C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\phi_i^*}\cdot\boldsymbol{\beta_a^*})^2$$
$$\text{Subject to :}\ \ \boldsymbol{\phi_i}\cdot\boldsymbol{\beta_a} = \boldsymbol{x_i} - \boldsymbol{\phi_i^*}\cdot\boldsymbol{\beta_a^*},\ i = 1, 2, ..., N, \tag{7.20}$$

where $\left\|\boldsymbol{\beta_a}\right\|^2$ reflects the capacity of the decision function and $\left\|\boldsymbol{\beta_a^*}\right\|^2$ reflects the capacity of the correction function. Here, μ controls the capacity of these two functions i.e. controls the relative weight of these two capacities.

By using Representer Theorem [103], $\boldsymbol{\beta_a}$ can be expressed as a linear combination of the training data representation in non-linear feature space $\boldsymbol{\Phi}$ and reconstruction weight matrix $\boldsymbol{W_a}$:

$$\boldsymbol{\beta_a} = \boldsymbol{\Phi}\cdot\boldsymbol{W_a} \quad \text{and} \quad \boldsymbol{\beta_a}* = \boldsymbol{\Phi^*}\cdot\boldsymbol{W_a^*}. \tag{7.21}$$

By substituting the (7.21) into (7.20), following minimization problem is obtained:

$$\underset{\boldsymbol{W_a},\boldsymbol{W_a^*},\boldsymbol{e_i}}{\text{Minimize}} : \pounds_{AEKOC+} = \frac{1}{2}\boldsymbol{\Phi}\cdot\boldsymbol{\Phi}\cdot\left\|\boldsymbol{W_a}\right\|_F^2 + \mu\frac{1}{2}\boldsymbol{\Phi^*}\cdot\boldsymbol{\Phi^*}\cdot\left\|\boldsymbol{W_a^*}\right\|_F^2 +$$
$$C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\phi_i^*}\cdot\boldsymbol{\phi_i^*}\cdot\boldsymbol{W_a^*})^2 \tag{7.22}$$
$$\text{Subject to :}\ \ \boldsymbol{\phi_i}\cdot\boldsymbol{\phi_i}\cdot\boldsymbol{W_a} = \boldsymbol{x_i} - \boldsymbol{\phi_i^*}\cdot\boldsymbol{\phi_i^*}\cdot\boldsymbol{W_a^*},\ i = 1, 2, ..., N.$$

Further, we substitute $\boldsymbol{K} = \boldsymbol{\Phi}\cdot\boldsymbol{\Phi}$, $\boldsymbol{K^*} = \boldsymbol{\Phi^*}\cdot\boldsymbol{\Phi^*}$, $\boldsymbol{k_i} = \boldsymbol{\phi_i}\cdot\boldsymbol{\Phi_i}$, and $\boldsymbol{k_i^*} = \boldsymbol{\phi_i^*}\cdot\boldsymbol{\Phi_i^*}$ in (7.22). Now, (7.22) is reformulated as follows:

$$\underset{\boldsymbol{W_a},\boldsymbol{W_a^*},\boldsymbol{e_i}}{\text{Minimize}} : \pounds_{AEKOC+} = \frac{1}{2}\boldsymbol{K}\cdot\left\|\boldsymbol{W_a}\right\|_F^2 + \mu\frac{1}{2}\boldsymbol{K^*}\cdot\left\|\boldsymbol{W_a^*}\right\|_F^2 + C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{k_i^*}\cdot\boldsymbol{W_a^*})^2$$
$$\text{Subject to :}\ \ \boldsymbol{k_i}\cdot\boldsymbol{W_a} = \boldsymbol{x_i} - \boldsymbol{k_i^*}\cdot\boldsymbol{W_a^*},\ i = 1, 2, ..., N, \tag{7.23}$$

where $\boldsymbol{k_i} \subseteq \boldsymbol{K}$ and $\boldsymbol{k_i^*} \subseteq \boldsymbol{K^*}$.

The Lagrangian relaxation of (7.23) can be written as follows:

$$\underset{\alpha_i}{\text{Maximize}} \quad \underset{\boldsymbol{W_a},\boldsymbol{W_a^*},e_i}{\text{Minimize}} : \pounds_{AEKOC+} = \frac{1}{2}\boldsymbol{K} \cdot \|\boldsymbol{W_a}\|_F^2 + \mu\frac{1}{2}\boldsymbol{K^*} \cdot \|\boldsymbol{W_a^*}\|_F^2 +$$
$$C\frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{k_i^*} \cdot \boldsymbol{W_a^*})^2 - \sum_{i=1}^{N}\boldsymbol{\alpha_i}(\boldsymbol{k_i} \cdot \boldsymbol{W_a} - \boldsymbol{x_i} + \boldsymbol{k_i^*} \cdot \boldsymbol{W_a^*}), \tag{7.24}$$
$$i = 1, 2, ..., N,$$

where $\boldsymbol{\alpha} = \{\boldsymbol{\alpha_i}\}, i = 1, 2 \ldots N$, is a Lagrangian multiplier. In order to optimize (7.24), we compute the partial derivatives of (7.24) as follows:

$$\frac{\partial \pounds_{AEKOC+}}{\partial \boldsymbol{W_a}} = 0 \Rightarrow \boldsymbol{W_a} = \boldsymbol{\alpha}, \tag{7.25}$$

$$\frac{\partial \pounds_{AEKOC+}}{\partial \boldsymbol{W_a^*}} = 0 \Rightarrow \mu\boldsymbol{K^*} \cdot \boldsymbol{W_a^*} + C\boldsymbol{K^*} \cdot \boldsymbol{W_a^*} \cdot \boldsymbol{K^*} - \boldsymbol{\alpha} \cdot \boldsymbol{K^*} = 0, \tag{7.26}$$

$$\frac{\partial \pounds_{AEKOC+}}{\partial \boldsymbol{\alpha}} = 0 \Rightarrow \boldsymbol{K} \cdot \boldsymbol{W_a} + \boldsymbol{K^*} \cdot \boldsymbol{W_a^*} = \boldsymbol{X}. \tag{7.27}$$

Now, substitute (7.25) and (7.27) into (7.26):

$$\mu(\boldsymbol{X} - \boldsymbol{K} \cdot \boldsymbol{W_a}) + C(\boldsymbol{X} - \boldsymbol{K} \cdot \boldsymbol{W_a}) \cdot \boldsymbol{K^*} - \boldsymbol{W_a} \cdot \boldsymbol{K^*} = 0. \tag{7.28}$$

After solving the (7.28), $\boldsymbol{W_a}$ is obtained as follows:

$$\boldsymbol{W_a} = (\mu\boldsymbol{K} + C\boldsymbol{K} \cdot \boldsymbol{K^*} + \boldsymbol{K^*})^{-1} \cdot (\mu\boldsymbol{I} + C\boldsymbol{K^*}) \cdot \boldsymbol{X}. \tag{7.29}$$

The weight $\boldsymbol{\beta_a}$ is obtained by substituting (7.29) into (7.21) as follows:

$$\boldsymbol{\beta_a} = \boldsymbol{\Phi} \cdot (\mu\boldsymbol{K} + C\boldsymbol{K} \cdot \boldsymbol{K^*} + \boldsymbol{K^*})^{-1} \cdot (\mu\boldsymbol{I} + C\boldsymbol{K^*}) \cdot \boldsymbol{X}. \tag{7.30}$$

The predicted output of AEKOC+ for training samples can be calculated as follows:

$$\widehat{\boldsymbol{O}}_{\boldsymbol{a}} = \boldsymbol{\Phi}.\boldsymbol{\beta_a} = \boldsymbol{\Phi}.\boldsymbol{\Phi}.\boldsymbol{W_a} = \boldsymbol{K}.\boldsymbol{W_a}, \tag{7.31}$$

where $\widehat{\boldsymbol{O}}_{\boldsymbol{a}} = \left\{(\widehat{\boldsymbol{O}}_{\boldsymbol{a}})\boldsymbol{i}\right\}$, and $i = 1, 2, ..., N$, is the predicted output for training data. After obtaining the predicted output value, we compute a threshold value based on the predicted value at the output layer. This threshold value helps in deciding whether a sample is an outlier or not. A threshold ($\theta 1$) is employed with the proposed method, which is determined as follows:

(i) We calculate the sum of square error as reconstruction error between the predicted value of the i^{th} training sample and $\boldsymbol{x_i}$, and store the distance in a vector, $\boldsymbol{\Lambda} = \{\Lambda_i\}$ and $i = 1, 2, ..., N$, as follows:

$$\Lambda_i = \sum_{j=1}^{n}((\widehat{O}_a)_{ij} - x_{ij})^2. \tag{7.32}$$

(ii) After storing all distances in $\boldsymbol{\Lambda}$ as per (7.32), we sort these distances in decreasing order and denoted by a vector $\boldsymbol{\Lambda_{dec}}$. Further, we reject a few percents of training samples based on the deviation. Most deviated samples are rejected first because they are most probably far from the distribution of the target data. The threshold is decided based on these deviations as follows:

$$\theta 1 = \Lambda_{dec}(\lfloor \nu * N \rfloor), \tag{7.33}$$

where $0 < \nu \leq 1$ is the fraction of rejection of training samples for deciding threshold value. N is the number of training samples and $\lfloor\ \rfloor$ denotes floor operation.

After determining a threshold value by the above procedure, during testing, a test vector $\boldsymbol{x_p}$ is fed to the trained architecture and its output $\boldsymbol{(\widehat{O}_a)_p}$ is obtained. Further, compute the distance ($\widehat{\Lambda}_p$), for $\boldsymbol{x_p}$, between the predicted value $\boldsymbol{(\widehat{O}_a)_p}$ of the p^{th} testing sample and $\boldsymbol{x_p}$:

$$\widehat{\Lambda}_p = \sum_{j=1}^{n}((\widehat{O}_a)_{pj} - x_{pj})^2. \tag{7.34}$$

Finally, $\boldsymbol{x_p}$ is classified based on the following rule:

$$\begin{aligned} &\text{If } \widehat{\Lambda}_p \leq \theta 1, \quad \boldsymbol{x_p} \text{ belongs to normal class,} \\ &\text{Otherwise,} \qquad\qquad\quad \boldsymbol{x_p} \text{ is an outlier.} \end{aligned} \tag{7.35}$$

7.3 Experiments

The proposed KOC+ and AEKOC+ classifiers are experimented on 11 datasets. We select the same datasets which are used by Zhu and Zhong [37]. We generate these 11 one-class datasets from 5 multi-class datasets. For a multi-class dataset with c classes, we alternately select any one class among c classes and treat all samples from this class as a target positive class. All samples of the remaining $(c-1)$ classes are outliers. In this way, if a multi-class dataset had c classes; then, we generate c one-class datasets. This procedure of generating one-class datasets from multi-class datasets is the same as discussed in [6]. They made various one-class datasets available on [169]. One more critical factor is the group attribute [37]. The group attribute is privileged information of the dataset. We select any one attribute at a time from the dataset as a group attribute. After that, we convert the group attribute into a categorical variable by imposing some criteria, as discussed in the further sections for each dataset. The group attributes' name of each dataset is mentioned in Table 7.1. Suppose, if there are 2 group attributes with a dataset; then, we perform the experiment corresponding to each group attribute separately. It is discussed in more detail with each dataset in the further sections. By following the work of [39], we compute the area under the precision-recall curve for comparing the performance of the classifiers [177, 178]. It is also called as average precision score [177, 178]. It summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight [177, 178]:

$$\text{Average precision score} = \sum_t ((\eta_{pr})_t - (\eta_{re})_{t-1})(\eta_{pr})_t, \tag{7.36}$$

Table 7.1: Dataset description

S. No.	Datasets	Group Attributes	# Total Samples	# Target	# Outlier	Name of Target Class
1	Abalone(1)	(a) Height (b) Length (c) Whole weight	4177	1407	2770	class 1-8
2	Abalone(2)			1323	2854	class 9-10
3	Abalone(3)			1447	2730	class 11-29
4	Haberman (1)	(a) Age of patient at time of operation (b) Number of positive axillary nodes detected	306	225	81	The patient survived 5 years or longer
5	Haberman (2)			81	225	The patient died within 5 year
6	Heart(1) (Statlog)	(a) Age (b) Electrocardiographic (c) Sex	270	150	120	Absence
7	Heart(2) (Statlog)			120	150	Presence
8	MNIST(1)	(a) Poetic Description	5981	2948	3033	5
9	MNIST(2)			3033	2948	8
10	Wisconsin Breast Cancer(1) (WBC(1))	(a) Clump Thickness (b) Uniformity of Cell Size (c) Uniformity of Cell Shape (d) Marginal Adhesion	683	239	444	Malignant
11	Wisconsin Breast Cancer(2) (WBC(2))			444	239	Benign

where $(\eta_{pr})_t$ and $(\eta_{re})_t$ are the precision and recall at the t^{th} threshold [177, 178]. We also compute mean of the average precision score over all datasets for each classifier, which is denoted as η_{mp}. In our experiments, we employ Gaussian kernel for all meth-

ods. Range of the Gaussian kernel parameter is $[10^{-10}, 10^{-9}, \cdots, 10^{19}, 10^{20}]$. Range of the remaining parameters (i.e., ν, λ, and γ) is $0.05 \leq \nu \leq 0.7$. Here, we select 10 values between them with an equal interval. We have conducted all experiments on Windows 7 (Intel Xeon 3 GHz processor, 64 GB RAM) with Python 2.7 environment.

7.3.1 Abalone Dataset

In the Abalone dataset, the goal of the classifier is to predict the age of abalone from physical measurements. It contains 29 classes. We first group the 29 classes into three classes [169]. This procedure is the same as discussed in [37], and [169]. Further, we generate 3 one-class datasets by using above-discussed procedure [6][169]. Description of these three datasets, namely, Abalone(1), Abalone(2), and Abalone(3), are available in Table 7.1. This dataset has 8 attributes. Further, we follow the two steps procedure [37] for creating privileged information. This procedure is the same as that in [37]. In the first step, we select 3 out of 8 attributes as a group attribute (i.e., privileged attribute). At a time, we take only one attribute as a privileged attribute and separate from the data. In the second step, we partition data into two or more than two groups by posing some criteria on the value of each group attribute. When we apply the criteria on the attribute, then that attribute is converted in the form of a categorical attribute. Due to this conversion, actual information of this attribute is hidden, and now, we can treat it as privileged information. A detailed description of these group partitions is available in Table 7.2. By using the group attributes of this table, we provide three comparisons of results for Abalone(1), Abalone(2), and Abalone(3) datasets in Table 7.3. In each comparison, we consider only one out of three group attributes as a group attribute, and remaining are available as general

Table 7.2: Partitioning of data as per group attribute

Datasets	Group Attribute	group-1	group-2
Abalone	Height	Height<0.15	Height≥ 0.15
	Length	Length<0.5	Length≥ 0.5
	Whole weight	Whole weight<0.8	Whole weight≥ 0.8

attributes. We perform five-fold cross-validation for computing these results. We follow the above-discussed procedure for all datasets used in this chapter. Results of Abalone(1), Abalone(2), and Abalone(3) datasets are available in Table 7.3. The last row contains an average of all results, i.e., average precision score (η_{mp}). It provides a clear idea regarding the overall performance of the classifiers. Proposed classifiers, KOC+ and AEKOC+, collectively outperforms all existing classifiers by a significant margin. However, both methods do not individually outperform existing classifiers in some cases (see in Table 7.3). In spite of this bad performance with some group attributes, KOC+ and AEKOC+ yield better results with a significant margin in terms of the mean of the average precision score (η_{mp}). It is available in the last row of the table. Minimum and maximum margin of η_{mp} between KOC+ and existing methods are 3.41% and 6.39%, respectively. Minimum and maximum margin of η_{mp} between AEKOC+ and existing methods are 4.47% and 7.45%, respectively. Overall, AEKOC+ yields the best η_{mp} compared to all classifiers.

Table 7.3: Average precision score after 5-fold CV for Abalone dataset

Datasets	Group Attribute	OCSVM+	SVDD+	AEKOC	AEKOC+	KOC	KOC+
Abalone(1)	Height	73.87	74.42	76.22	84.89	79.24	**85.15**
	Length	80.85	80.59	77.58	**85.03**	82.10	42.71
	Whole weight	78.20	77.91	75.76	**86.20**	79.72	63.49
Abalone(2)	Height	52.12	52.33	48.93	39.67	38.40	**85.66**
	Length	**54.45**	52.64	51.86	45.59	41.91	37.70
	Whole weight	56.34	56.70	46.71	44.05	43.53	**67.93**
Abalone(3)	Height	50.84	51.30	45.21	68.27	52.66	**85.46**
	Length	46.11	46.09	43.19	**65.23**	51.93	39.72
	Whole weight	48.53	47.26	49.06	62.65	49.39	**64.19**
mean of average precision score (η_{mp})		60.15	59.91	57.17	**64.62**	57.66	63.56

7.3.2 Haberman Dataset

This dataset contains two classes. We generate two one-class datasets from this, namely, Haberman(1), and Haberman(2). In this dataset, we have to predict the survival of breast cancer patients after surgery. We consider two attributes out of three attributes as group attributes, namely, 'Age of a patient at the time of operation,' and

'Number of positive axillary nodes detected.' We partition data into two groups based on the value of each group attributes by considering these as privileged information. A detailed description of theses group partitions is available in Table 7.4. We present results comparison between the proposed and existing methods in Table 7.5. The last row contains an average of all results, i.e., average precision score (η_{mp}). Proposed classifiers, KOC+ and AEKOC+, collectively outperform all existing classifiers by a significant margin. More specifically, AEKOC+ yields the best results among all cases except one. For Haberman(1), AEKOC+ outperforms AEKOC by a margin of 2.3% and 6.9% for two group attributes. It also outperforms for both group attributes. For Haberman(2), AEKOC+ outperforms AEKOC by a margin of 2.7% and 16.6% for two group attributes. Minimum and maximum margin of η_{mp} between AEKOC+ and existing methods are 1.21% and 7.14%, respectively. Overall, AEKOC+ yields the best η_{mp} compared to all classifiers.

Table 7.4: Partitioning of data as per group attribute

Datasets	Group Attribute	group-1	group-2
Haberman	Age of patient at time of operation	Age$\leq$ 50	Age$\geq$ 51
	Number of positive axillary nodes detected	Number = 0	Number$\geq$ 1

Table 7.5: Average precision score after 5-fold CV for Haberman dataset

Datasets	Group Attribute	OCSVM+	SVDD+	AEKOC	AEKOC+	KOC	KOC+
Haberman(1)	Age of patient at time of operation	81.37	80.37	80.89	83.22	**84.46**	80.46
	Number of positive axillary nodes detected	72.55	72.48	65.70	**72.61**	69.05	72.53
Haberman(2)	Age of patient at time of operation	45.46	44.56	29.21	**45.77**	37.93	36.58
	Number of positive axillary nodes detected	26.47	26.47	26.33	**29.08**	24.46	25.24
mean of average precision score (η_{mp})		56.46	55.97	50.53	**57.67**	53.98	53.70

Table 7.6: Partitioning of data as per group attribute

Datasets	Group Attribute	group-1	group-2	group-3
Statlog Heart	Age	Age$\leq$ 50	51 $\leq$Age$\leq$ 60	Age$\geq$ 61
	Sex	Sex= 0	Sex= 1	–
	Electrocardiographic	Electrocardiographic= 0 or 1	Electrocardiographic= 2	–

7.3.3 Heart (Statlog) Dataset

There are two classes in this dataset. We create two one-class datasets using the above-discussed procedure, namely, Heart(1) (Statlog) and Heart(2) (Statlog). Heart (Statlog) has a total of 13 attributes. We select 3 out of 13 attributes as a group attribute (i.e., privileged attribute). Those attributes are 'Age', 'Sex', and 'Electrocardiographic'. We partition data into two groups based on the value of each group attributes. We alternately treat each of the group attributes as privileged information. A detailed description of these group partitions is available in Table 7.6. By using the group attributes of this table, we provide three comparisons of results for Statlog Heart(1) and Statlog Heart(2) datasets in Table 7.7. Both KOC+ and AEKOC+ outperform their corresponding base method KOC and AEKOC, respectively. Among all comparisons in Table 7.7, KOC+ and AEKOC+ outperform all presented classifiers for all cases except one. More specifically, AEKOC+ only does not yield a better result in the case of Heart(2) for group attribute 'Age'. We also compute η_{mp} value and present in the last row of Table 7.7. Minimum and maximum margin of η_{mp} between KOC+ and existing methods are 3.69% and 7.55%, respectively. Minimum and maximum margin of η_{mp} between AEKOC+ and existing methods are 1.71% and 5.56%, respectively. Overall, KOC+ and AEKOC+ yield better η_{mp} compared to all classifiers.

Table 7.7: Average precision score after 5-fold CV for Heart dataset (Statlog)

Datasets	Group Attribute	OCSVM+	SVDD+	AEKOC	AEKOC+	KOC	KOC+
Heart(1)	Age	84.32	85.39	83.93	88.47	84.21	**90.10**
	Electrocardiographic	80.96	83.72	84.19	**85.80**	81.29	84.64
	Sex	83.82	83.88	83.53	84.65	84.03	**86.84**
Heart(2)	Age	81.25	81.59	74.41	78.91	75.02	**83.58**
	Electrocardiographic	79.95	79.98	73.30	82.14	75.20	**85.87**
	Sex	77.42	78.40	70.47	83.23	71.10	**84.08**
mean of average precision score (η_{mp})		81.29	82.16	78.31	83.87	78.48	**85.85**

7.3.4 MNIST Dataset

We picked this dataset from [13]. Vapnik and Vashist [13] performed binary classification on the subset of MNIST dataset. They had taken digit 5 and 8 as two classes. We renamed these digits as class 1 and 2 in this chapter. Like earlier, we create two one-class datasets, namely, MNIST(1) and MNIST(2), from these two classes. Original images in the MNIST database are the size of 28×28 pixels. We resized these images from 28×28 to 10×10, as shown in Figure 7.1. It make classification more difficult [13]. The description of the dataset is available in Table 7.1.

Figure 7.1: Digit 5 and 8. First row shows original image which is of 28×28 pixel size and second row shows resized image of 10×10 pixel size

Vapnik and Vashisht [13] generated the holistic (poetic) description of the image. These descriptions are treated as privileged information. The poetic description can be understood by the examples provided in Section 2.6 of Chapter 2, which were originally provided by Vapnik and Vashist [13]. They transformed the poetic description (see

the the poetic description in the example in Section 2.6 of Chapter 2) into 21 features like two-part-ness (0 - 5); tilting to the right (0 -3); aggressiveness (0 - 2); stability (0 - 3); uniformity(0 - 3), etc. Here, first digit 5 and 8 contain features $[2, 1, 2, 0, 1]$, and $[4, 1, 1, 0, 2]$, respectively. They have also provided preprocessed data with training, validation, and testing splits[1]. We also use the same split for our experiments as used in [13, 179]. Results for MNIST(1) and MNIST(2) datasets are available in Table 7.8. For MNIST(1), KOC+ and AEKOC+ outperform all available classifiers in the table. In case of MNIST(2) dataset, KOC+ and AEKOC+ outperform all three existing classifiers viz., OCSVM+, SVDD+, and KOC; however, do not outperform AEKOC. We also compute η_{mp} value and present in the last row of Table 7.7. Overall, AEKOC+ yields best η_{mp} compared to all classifiers. Minimum and maximum margin of η_{mp} between AEKOC+ and existing methods are 0.64% and 6.41%, respectively.

Table 7.8: Average precision score after 5-fold CV for MNIST dataset

Datasets	Group Attribute	OCSVM+	SVDD+	AEKOC	AEKOC+	KOC	KOC+
MNIST(1)	Poetic Description	63.63	63.64	71.98	**73.56**	63.25	72.35
MNIST(2)		79.22	79.21	**81.66**	81.35	78.85	79.63
mean of average precision score (η_{mp})		71.42	71.42	76.82	**77.45**	71.05	75.99

7.3.5 Wisconsin Breast Cancer (WBC) Dataset

This dataset contains two classes. We generate two one-class datasets from this, namely, WBC(1), and WBC(2). Here, the goal is to predict whether a sample belongs to a benign or malignant class. We removed 16 samples, which contained the missing values. The remaining number of samples is 683, as mentioned in Table 7.1. There is a total of nine attributes in the WBC dataset. We select four attributes as group attributes, namely, 'Clump Thickness,' 'Uniformity of Cell Size,' 'Uniformity of Cell Shape,' 'Marginal Adhesion.' We partition data into two groups. The division of these groups is based on the value of each group attributes. We alternately treat each of the group attributes as privileged information. A detailed description of theses group partitions is available in Table 7.9. KOC+ outperforms all available classifiers in the

[1] https://github.com/Chandan-IITI/svmplus_matlab/tree/master/data

table for WBC(1) and WBC(2) datasets. However, AEKOC+ does not yield better results for a few cases. We also compute η_{mp} value and present in the last row of Table 7.7. Minimum and maximum margin of η_{mp} between KOC+ and existing methods are 2.21% and 4.97%, respectively. Minimum and maximum margin of η_{mp} between AEKOC+ and existing methods are 0.09% and 2.85%, respectively. Overall, KOC+ and AEKOC+ yield better η_{mp} compared to all classifiers.

Table 7.9: Partitioning of data as per group attribute

Datasets	Group Attribute	group-1	group-2
WBC	Clump Thickness	1≤Thickness≤ 2	3 ≤Thickness≤ 5
	Uniformity of Cell Size	Cell Size= 1	2 ≤Cell Size≤ 10
	Uniformity of Cell Shape	Cell Shape= 1	2 ≤Cell Shape≤ 10
	Marginal Adhesion	Adheson= 1	2 ≤Adheson≤ 10

Table 7.10: Average precision score after 5-fold CV for WBC dataset

Datasets	Group Attribute	OCSVM+	SVDD+	AEKOC	AEKOC+	KOC	KOC+
WBC(1)	Clump Thickness	92.56	93.61	88.49	93.71	91.37	**97.10**
	Uniformity of Cell Size	93.35	93.30	87.81	94.12	90.40	**98.72**
	Uniformity of Cell Shape	94.45	95.45	87.92	96.20	92.62	**98.22**
	Marginal Adhesion	94.32	94.68	88.88	92.68	89.30	**98.33**
WBC(2)	Clump Thickness	99.42	99.18	99.45	99.52	99.42	**99.66**
	Uniformity of Cell Size	97.60	98.50	99.61	98.78	99.61	**99.71**
	Uniformity of Cell Shape	99.57	99.37	99.62	**99.65**	99.58	99.63
	Marginal Adhesion	99.50	99.30	99.54	99.48	99.51	**99.74**
mean of average precision score (η_{mp})		96.34	96.67	93.92	96.77	95.23	**98.89**

Based on the above discussion and available results, we observe that KOC+ and AEKOC+ collectively yield the best results for most cases. Here, one case means a comparison of results of one dataset for one group attribute. Total 29 comparisons are provided in Table 7.3, 7.5, 7.7, 7.8, and 7.10. KOC+ yields better results for 17 out of 29 cases. Table 7.11 shows that number of datasets with the best η_g value for each classifier. This table shows that proposed classifiers attain the top position in the table, and remaining classifiers yield the best results for only 3 datasets. As per

discussion in Chapter 3, we also compute three more criteria [167] for further analysis: (i) mean of average precision score (i.e., η_{mp}) over 29 comparisons (ii) Friedman test (F-score and p-value) (iii) Friedman rank (η_f). Based on these three criteria, we analyze the performance and provide in the following points:

(i) We compute η_{mp} for analyzing the combined performance of the classifier. η_{mp} value of each classifier is available in Table 7.12(a) in decreasing order of η_{mp}. In this table, we observe that KOC+ attains top position and yields maximum η_{mp} among all proposed and existing classifiers. Second position is also hold by AEKOC+. Difference between the η_{mp} values of first and second position is only 0.01. However, difference between the η_{mp} value of 1^{st} and 3^{rd} position of the classifier is 2.5. This difference is significant and we statistically verify the same in the next point.

(ii) Although proposed classifiers attain top position among all classifiers in Table 7.12(a). However, we need to verify the outcomes of the proposed and existing classifiers statistically. For this purpose, we conduct a non-parametric Friedman test [166] similar as discussed in Section 3.2 of Chapter 3. Friedman test mainly computes three components viz., p-value, F-score, and critical value. The computed p-value, F-score value and critical value are $3.1443e-07$, 38.3941 and 11.0705, respectively. Since the computed F-score value is higher than the critical value, and the p-value is significantly lesser than the tolerance level of 0.05, we reject the null hypothesis with 95% of confidence. Therefore, we conclude that the outcomes presented in this chapter are statistically significant.

(iii) Further, we compute Friedman Rank (η_f) [166] for each classifier as discussed in Section 2.8 of Chapter 2. We consider η_f as the final decision criteria to decide the rank of any classifiers. In Table 7.12(b), we present η_f along with η_{mp} of each classifier in increasing order of η_f. In this table, KOC+ does not attain top position in spite of the best η_{mp} in Table 7.12(a). Here, AEKOC+ attains top position as per η_f. ALL LUPI framework-based methods obtain better η_{mp} and η_f compared to tradition machine learning-based methods.

Table 7.11: Number of datasets for which each one-class classifier yields the best results

One-class Classifiers	Number of datasets with the best η_g value
KOC+	17.00
AEKOC+	9.00
OCSVM+	1.00
AEKOC	1.00
KOC	1.00
SVDD+	0.00

Table 7.12: Friedman Rank (η_f) and mean of average precision score (η_{mp})

7.12(a) In decreasing order of η_{mp}

	η_f	η_{mp}
KOC+	2.34	77.41
AEKOC+	2.28	77.40
SVDD+	3.84	74.91
OCSVM+	3.78	74.78
KOC	4.21	72.74
AEKOC	4.55	72.12

7.12(b) In increasing order of η_f

	η_f	η_{mp}
AEKOC+	2.28	77.40
KOC+	2.34	77.41
OCSVM+	3.78	74.78
SVDD+	3.84	74.91
KOC	4.21	72.74
AEKOC	4.55	72.12

7.4 Summary

In this chapter, we have investigated non-iterative learning-based one-class classifiers for the LUPI setting. We enabled KRR-based one-class classifiers for learning from privileged information by taking KOC and AEKOC as the base classifiers. KOC and AEKOC follow the traditional way of classification and ignore this additional information. However, the proposed one-class classifier, KOC+ and AEKOC+, utilized this information by introducing a correction function in the minimization problem of KOC and AEKOC. This correction function helped AEKOC+ in building a better classification model. Both proposed one-class classifiers have achieved better $\eta_m p$ and η_f compared to all other mentioned one-class classifiers in this chapter. We have statistically verified that the outcomes presented in this chapter are statistically significant, with 95% of confidence.

Although, the proposed KOC+ and AEKOC+ yielded better generalization performance compared to KOC and AEKOC, but they generated more complex model compared to KOC and AEKOC because proposed methods had to tune more parameters compared to KOC and AEKOC. In the real-time scenario, selection of appropriate group attribute will also be a challenging task, which needs to be investigated further. Also, the proposed methods can be further extended for different types of learnings, like online learning, one-shot learning, multi-task learning, etc.

Chapter 8

Adaptive Online Sequential Learning with Kernel Ridge Regression for One-class Classification

In all of the proposed works in the previous chapters, all methods belong to batch learning. These classifiers can efficiently handle stationary datasets; however, unable to handle non-stationary and streaming datasets. These types of data can be handled by online learning. Online learning has attracted researchers in recent years due to its capability to handle a high volume of streaming data with less computational and storage costs [164, 165, 180, 181]. In online learning, a model is built based on the currently available data and then continuously updates this model as the next samples arrive for training. Literature survey is briefly discussed in Section 2.7 of Chapter 2. In this chapter, KRR-based online sequential one-class classifiers are modeled for two types of frameworks viz., boundary, and reconstruction. We propose an online version of two one-class classifiers, viz., KOC, and AEKOC. Here KOC is boundary framework-based, and AEKOC is a reconstruction framework-based one-class classifier. Online sequential versions of KOC and AEKOC are named as OS-KOC and OS-AEKOC, respectively. Both proposed methods are discussed in the subsequent sections.

8.1 Boundary Framework-based Approach: OS-KOC

In Boundary framework-based method i.e. OS-KOC, model is trained by only target data $\boldsymbol{X}$ and approximates all data to a real number. Figure 8.1 shows online OCC with single output node architecture.

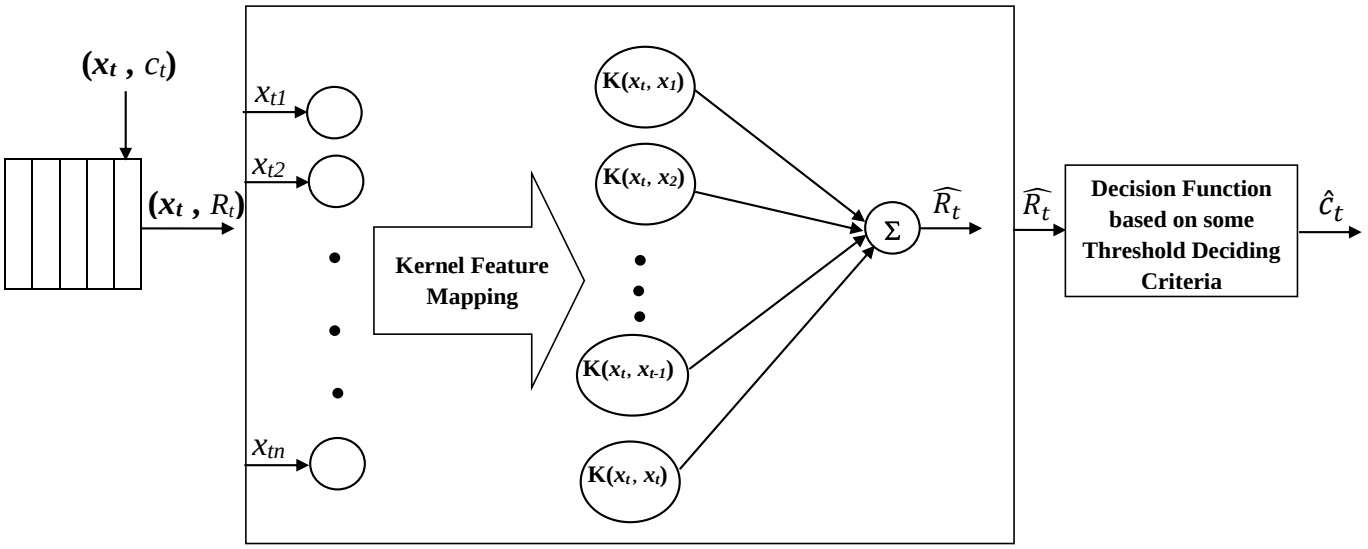


Figure 8.1: A schematic diagram of online sequential KRR-based single output node architecture for OCC: Boundary framework-based

Here, given a stream of training data $\boldsymbol{X}$: $\{(\boldsymbol{x}_1, c), (\boldsymbol{x}_2, c), ..., (\boldsymbol{x}_t, c), ...\}$, where $\boldsymbol{x}_t = [x_{t1}, x_{t2}, ..., x_{tn}] \in \Re^n$ is n-dimensional input of the t^{th} sample, and c is the class label of the target class. Input layer that takes data for t^{th} input sample is coded as $(\boldsymbol{x}_t, r)$ because model has to approximate all data to a real number r. Target output vector $\boldsymbol{R}$ is represented as an appropriate size of vector i.e. $[r, r....r, ...]$, where each element is r. Here, value of r is considered as 1 for all experiments. Further, kernel feature mapping has been employed between input and hidden layer by using a function $\boldsymbol{\phi(.)}$. Here, kernel matrix is represented as $\boldsymbol{K} = \boldsymbol{\Phi^T \Phi}$, where $\boldsymbol{\Phi}$ denotes feature mapping in higher dimension space. Note that $\boldsymbol{\Phi}$ can also be written in terms of $\boldsymbol{X}$, i.e. $\boldsymbol{\Phi} = \boldsymbol{\phi(X)}$, where $\boldsymbol{\phi(.)}$ denotes kernel feature mapping. Here, hidden layer output i.e. kernel matrix $\boldsymbol{K}$ is a square symmetric matrix of size $t \times t$.

Output weight vector $\boldsymbol{\beta}$ for any t samples in sequence of data is represented as $(\boldsymbol{\beta})_{\boldsymbol{t}}$. $\widehat{\boldsymbol{R}} = [\widehat{R}_1, \widehat{R}_2 ... \widehat{R}_t, ...]$ is the predicted output vector and $\widehat{R}_t$ is the predicted output for t^{th} sample. $\widehat{\boldsymbol{c}} = [\widehat{c_1}, \widehat{c_2} ... \widehat{c_t}, ...]$ is the predicted class vector, where $\widehat{c_t}$ is the predicted class for t^{th} sample. Based on these preliminaries, boundary framework-based online learning algorithm OS-KOC is discussed below in **four phases** i.e. **Initialization, Update, Forgetting** and **Decision**.

8.1.1 Initialization Phase

In this phase, first, basic KRR formulation is derived for kernel feature mapping by using a non-linear feature mapping $\boldsymbol{X} \rightarrow \boldsymbol{\Phi}$ and representer theorem[103]. Here, Linear case can be easily derived from the formulation of non-linear case by substituting $\boldsymbol{\Phi} \rightarrow \boldsymbol{X}$.

For the initial chunk $\{\boldsymbol{X_0}, \boldsymbol{R_0}\}$ of size N_0, the objective is to minimize both output weight vector $(\boldsymbol{\beta})_{\boldsymbol{0}}$ as well as a error vector $(\boldsymbol{e_0})$ between expected $(\boldsymbol{R_0})$ and predicted output $((\boldsymbol{\beta})_{\boldsymbol{0}}^{\boldsymbol{T}} \boldsymbol{\Phi_0})$. Minimization problem for non-linear feature mapping case can be written as follows:

$$\begin{aligned} \underset{(\boldsymbol{\beta})_{\boldsymbol{0}}, \boldsymbol{e_0}}{\text{Minimize}} : L &= \frac{1}{2} \left\| \boldsymbol{\beta_0} \right\|^2 + C \frac{1}{2} \left\| \boldsymbol{e_0} \right\|^2 \\ \text{Subject to} : \ & \boldsymbol{\beta_0^T \Phi_0 = R_0 - e_0}. \end{aligned} \tag{8.1}$$

Here, C is used as a regularization parameter.

Representer Theorem [103] is exploited in (8.2), which describes the weight vector $\boldsymbol{w}$ as a linear combination of the training data representation in non-linear space $(\boldsymbol{\Phi_0})$ and a reconstruction weight vector $(\boldsymbol{W_0})$ as follows:

$$(\boldsymbol{\beta})_{\boldsymbol{0}} = \boldsymbol{\Phi_0 W_0}. \tag{8.2}$$

Further, by using Representer theorem, minimization problem in (8.1) can be reformulated as follows:

$$\begin{aligned} \underset{(\boldsymbol{W})_{\boldsymbol{0}}, \boldsymbol{e_0}}{\text{Minimize}} : L &= \frac{1}{2} \boldsymbol{W_0^T \Phi_0^T \Phi_0 W_0} + C \frac{1}{2} \left\| \boldsymbol{e_0} \right\|^2 \\ \text{Subject to} : \ & \boldsymbol{W_0^T \Phi_0^T \Phi_0 = R_0 - e_0}. \end{aligned} \tag{8.3}$$

Now, substituting kernel matrix of initial training samples, $\boldsymbol{K_0} = \boldsymbol{(\Phi_0)^T \Phi_0} = \boldsymbol{K(X_0, X_0)}$, in the above equation and obtain the following minimization problem:

$$\begin{aligned} \underset{\boldsymbol{W_0}, \boldsymbol{e_0}}{\text{Minimize}} : L = \frac{1}{2}\boldsymbol{W_0^T K_0 W_0} + C\frac{1}{2}\left\|\boldsymbol{e_0}\right\|^2 \\ \text{Subject to : } \boldsymbol{W_0^T K_0 = R_0 - e_0}. \end{aligned} \tag{8.4}$$

By using Lagrangian relaxation [182], (8.4) can be written as:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{Maximize}} \quad \underset{\boldsymbol{W_0}, \boldsymbol{e_0}}{\text{Minimize}} : L = \frac{1}{2}\boldsymbol{W_0^T K_0 W_0} + C\frac{1}{2}\left\|\boldsymbol{e_0}\right\|^2 \\ -\boldsymbol{\alpha(W_0^T K_0 - R_0 + e_0)}, \end{aligned} \tag{8.5}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_{N_0}]$ are the Lagrange multipliers, which are non-negatives and employed to combine the constraint with the minimization problem. By using Karush-Kuhn-Tucker (KKT) theorem [183], take partial derivatives of (8.5) with respect to all variables $\boldsymbol{W_0}, \boldsymbol{e_0}$ and α. That is

$$\begin{aligned} &\frac{\partial L}{\partial \boldsymbol{W_0}} = 0 \Rightarrow \boldsymbol{W_0} = \boldsymbol{\alpha}, \\ &\frac{\partial L}{\partial \boldsymbol{e_0}} = 0 \Rightarrow C\boldsymbol{e_0} = \boldsymbol{\alpha}, \\ &\frac{\partial L}{\partial \boldsymbol{\alpha}} = 0 \Rightarrow \boldsymbol{W_0^T K_0 - R_0 + e_0} = 0. \end{aligned} \tag{8.6}$$

After solving the above set of equations in (8.6) for $\boldsymbol{W_0}$, it is obtained as follows:

$$\boldsymbol{W_0} = (\boldsymbol{K_0} + \frac{1}{C}\boldsymbol{I})^{-1}\boldsymbol{R_0}, \tag{8.7}$$

where $\boldsymbol{I}$ is an identity matrix.

Finally, $\boldsymbol{W_0}$ for initial training samples is obtained as follows:

$$\boldsymbol{W_0} = \boldsymbol{P_0 R_0}, \text{whrere } \boldsymbol{P_0} = \left(\boldsymbol{K_0} + \frac{\mathbf{1}}{\boldsymbol{C}}\boldsymbol{I}\right)^{-\mathbf{1}}. \tag{8.8}$$

$\boldsymbol{K_0}$ is defined based on Mercer's condition. That is, any kernel method which satisfies Mercer's condition can be adopted as the kernel for the classifier. Obtained $\boldsymbol{K_0}$ and

$\boldsymbol{P_0}$ are stored in different variables $\boldsymbol{K}$ and $\boldsymbol{P}$ respectively as these variables need to be updated when new training samples arrives. Update of $\boldsymbol{K}$ and $\boldsymbol{P}$ are discussed in the second phase. $\boldsymbol{K}$, $\boldsymbol{P}$, $\boldsymbol{K_0}$ and $\boldsymbol{P_0}$ are defined as follows:

$$\boldsymbol{K} = \left(\boldsymbol{K_0} + \frac{\boldsymbol{1}}{\boldsymbol{C}}\boldsymbol{I}\right) = \left(\begin{bmatrix} K_{11} & K_{12} & \dots & K_{1N_0} \\ K_{21} & K_{22} & \dots & K_{2N_0} \\ \vdots & \vdots & \ddots & \vdots \\ K_{N_01} & K_{N_02} & \dots & K_{N_0N_0} \end{bmatrix} + \frac{1}{C}\boldsymbol{I}\right), \tag{8.9}$$

$$\boldsymbol{P} = \boldsymbol{P_0} = \boldsymbol{K^{-1}} = \left(\begin{bmatrix} K_{11} & K_{12} & \dots & K_{1N_0} \\ K_{21} & K_{22} & \dots & K_{2N_0} \\ \vdots & \vdots & \ddots & \vdots \\ K_{N_01} & K_{N_02} & \dots & K_{N_0N_0} \end{bmatrix} + \frac{1}{C}\boldsymbol{I}\right)^{-1}. \tag{8.10}$$

8.1.2 Update of Kernel Matrix ($\boldsymbol{K}$) and Inverse of Kernel Matrix ($\boldsymbol{P}$)

Initially, $\boldsymbol{K}$ and $\boldsymbol{P}$ are calculated as per (8.9) and (8.10). Here, $\boldsymbol{K}$ represents kernel matrix and $\boldsymbol{P}$ represents inverse of this kernel matrix for all the arrived samples until now for training. $\boldsymbol{K}$ and $\boldsymbol{P}$ are updated continuously for any upcoming new samples $\boldsymbol{X^v}$ as per (8.11), where $\boldsymbol{X^v}$= $\{(\boldsymbol{x}_1^v, c_1), (\boldsymbol{x}_2^v, c_2), ..., (\boldsymbol{x}_s^v, c_s)\}$ and $\boldsymbol{X^v} \subset \boldsymbol{X}$. Current value of $\boldsymbol{K}$ is represented as $\boldsymbol{K_u}$, which is generated by using old samples $\boldsymbol{X^u} \subset \boldsymbol{X}$. Now, $\boldsymbol{K}$ is calculated after arrival of the new sample as follows:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K_u} & \boldsymbol{K_{u,v}} \\ (\boldsymbol{K_{u,v}})^{\boldsymbol{T}} & \boldsymbol{K_v} \end{bmatrix}, \tag{8.11}$$

where $\boldsymbol{K}$ is a combination of four block matrices. Block matrix $\boldsymbol{K_{u,v}}$ and $\boldsymbol{K_v}$ above are calculated as per (8.12), which is discussed next. Let the number of samples processed until now be b and number of samples in the current chunk be s. b is initially equal to N_0. Update b and s each time when calculation starts for new samples. The block

matrices $\boldsymbol{K_v}$ and $\boldsymbol{K_{u,v}}$ are defined as follows:

$$\boldsymbol{K_v} = \left(\begin{bmatrix} K(\boldsymbol{x}_1^v, \boldsymbol{x}_1^v) & \dots & K(\boldsymbol{x}_1^v, \boldsymbol{x}_s^v) \\ \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_s^v, \boldsymbol{x}_1^v) & \dots & K(\boldsymbol{x}_s^v, \mathbf{x}_s^v) \end{bmatrix} + \frac{1}{C}\boldsymbol{I} \right), \tag{8.12}$$

$$\boldsymbol{K_{u,v}} = \begin{bmatrix} K(\boldsymbol{x}_1^u, \boldsymbol{x}_1^v) & \dots & K(x_1^u, \boldsymbol{x_s^v}) \\ \vdots & \ddots & \vdots \\ K(\boldsymbol{x}_b^u, \boldsymbol{x}_1^v) & \dots & K(\boldsymbol{x}_b^u, \boldsymbol{x}_s^v) \end{bmatrix}. \tag{8.13}$$

$\boldsymbol{P}$, which is inverse of $\boldsymbol{K}$ given in (8.11), is defined as follows:

$$\boldsymbol{P} = \boldsymbol{K^{-1}} = \begin{bmatrix} \boldsymbol{K_u} & \boldsymbol{K_{u,v}} \\ \boldsymbol{(K_{u,v})^T} & \boldsymbol{K_v} \end{bmatrix}^{-1}. \tag{8.14}$$

Further, compute the inverse in (8.14) using block matrix inverse [184] given by:

$$\boldsymbol{S} = \boldsymbol{D^{-1}} = \begin{bmatrix} \boldsymbol{D_{11}} & \boldsymbol{D_{12}} \\ \boldsymbol{D_{21}} & \boldsymbol{D_{22}} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{S_{11}} & \boldsymbol{S_{12}} \\ \boldsymbol{S_{21}} & \boldsymbol{S_{22}} \end{bmatrix}. \tag{8.15}$$

$\boldsymbol{S}$ in (8.15) can be written as follows to obtain inverse:

$$\begin{bmatrix} \boldsymbol{D_{11}} & \boldsymbol{D_{12}} \\ \boldsymbol{D_{21}} & \boldsymbol{D_{22}} \end{bmatrix} \begin{bmatrix} \boldsymbol{S_{11}} & \boldsymbol{S_{12}} \\ \boldsymbol{S_{21}} & \boldsymbol{S_{22}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}, \tag{8.16}$$

where $\boldsymbol{S_{21}} = \boldsymbol{S_{12}^T}$ and $\boldsymbol{D_{21}} = \boldsymbol{D_{12}^T}$.

After solving (8.16), the following solution is obtained:

$$\begin{aligned} \boldsymbol{S_{11}} &= \boldsymbol{(D_{11} - D_{12}D_{22}^{-1}D_{21})^{-1}}, \\ \boldsymbol{S_{12}} &= \boldsymbol{-D_{11}^{-1}D_{12}(D_{22} - D_{21}D_{11}^{-1}D_{12})^{-1}}, \\ \boldsymbol{S_{21}} &= \boldsymbol{-D_{22}^{-1}D_{21}(D_{11} - D_{12}D_{22}^{-1}D_{21})^{-1}}, \\ \boldsymbol{S_{22}} &= \boldsymbol{(D_{22} - D_{21}D_{11}^{-1}D_{12})^{-1}}. \end{aligned}$$

Hence, (8.14) can be rewritten as follows:

$$\boldsymbol{P} = \boldsymbol{K}^{-1} = \begin{bmatrix} \boldsymbol{K_u} & \boldsymbol{K_{u,v}} \\ (\boldsymbol{K_{u,v}})^{\boldsymbol{T}} & \boldsymbol{K_v} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{P_{11}} & \boldsymbol{P_{12}} \\ \boldsymbol{P_{21}} & \boldsymbol{P_{22}} \end{bmatrix}. \tag{8.17}$$

Based on the above discussion, $\boldsymbol{P_{11}}$, $\boldsymbol{P_{12}}$, $\boldsymbol{P_{21}}$ and $\boldsymbol{P_{22}}$ in (8.17) are given as below:

$$\begin{aligned} \boldsymbol{P_{11}} &= (\boldsymbol{K_u} - \boldsymbol{K_{u,v}}\boldsymbol{K_v^{-1}}\boldsymbol{K_{u,v}^T})^{-1}, \\ \boldsymbol{P_{12}} &= -\boldsymbol{K_u^{-1}}\boldsymbol{K_{u,v}}\boldsymbol{P_{22}}, \\ \boldsymbol{P_{21}} &= -\boldsymbol{K_v^{-1}}\boldsymbol{K_{u,v}^T}\boldsymbol{P_{11}}, \\ \boldsymbol{P_{22}} &= (\boldsymbol{K_v} - \boldsymbol{K_{u,v}^T}\boldsymbol{K_u^{-1}}\boldsymbol{K_{u,v}})^{-1}. \end{aligned} \tag{8.18}$$

Further, $\boldsymbol{P_{11}}$ can be expanded by employing the Woodbury formula[185] as follows:

$$\begin{aligned} \boldsymbol{P_{11}} &= (\boldsymbol{K_u} - \boldsymbol{K_{u,v}}\boldsymbol{K_v^{-1}}\boldsymbol{K_{u,v}^T})^{-1} \\ &= \boldsymbol{K_u^{-1}} - \boldsymbol{K_u^{-1}}\boldsymbol{K_{u,v}}(\boldsymbol{K_{u,v}^T}\boldsymbol{K_u^{-1}}\boldsymbol{K_{u,v}} + \boldsymbol{K_v^{-1}})^{-1}\boldsymbol{K_{u,v}^T}\boldsymbol{K_u^{-1}}. \end{aligned} \tag{8.19}$$

Woodbury formula [185] is employed instead of computing the direct inverse because now there is a need to compute two inverses, i.e. K_u^{-1} and K_v^{-1} independently, where K_u^{-1} is already computed in the previous iteration. The advantage of this approach in terms of the time and storage complexity is described in Section 8.3.4. $\boldsymbol{P_{12}}$, $\boldsymbol{P_{21}}$ and $\boldsymbol{P_{22}}$ can be updated in a similar manner.

8.1.3 Adaptive Learning

During online learning, data increases continuously, and therefore, it creates two challenges: the algorithm needs to learn continuously in case (a) if the distribution of training samples changes, and (b) if the memory of system exhausts (as memory is not infinite). Both of these challenges can be addressed by a forgetting mechanism with a sliding window as shown in Figure 8.2. This mechanism helps to unlearn the old or irrelevant samples while unlearning of the trained model is required. Further,

relearning on new samples can be done in our proposed methods as discussed in the previous and next sections.

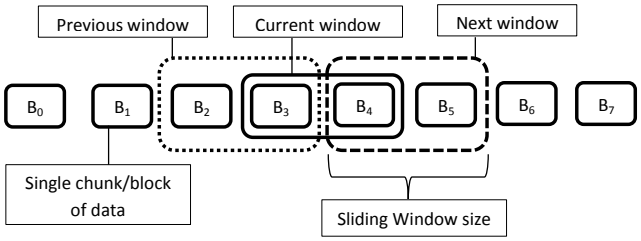


Figure 8.2: Illustration of sliding window for a given data stream

Forgetting mechanism for the proposed classifier:

Suppose, $\boldsymbol{P_{curr} = K_{curr}^{-1}} \in \boldsymbol{\Re^{s \times s}}$ is the inverse of the current kernel matrix $\boldsymbol{K_{curr}} \in \boldsymbol{\Re^{s \times s}}$. Now, the impact of learning of old f samples needs to be removed. In this mechanism, two things viz., kernel matrix and the inverse of that kernel matrix, need to be updated before moving to learn new samples. Modified kernel matrix $\boldsymbol{K_{new}}$ can be simply generated by removing rows and columns of the corresponding samples from the current kernel matrix $\boldsymbol{K_{curr}}$. Modified inverse matrix $\boldsymbol{P_{new}}$ is calculated after removing (forgetting) few samples from $\boldsymbol{K_{curr}}$ as follows:

$$\boldsymbol{P_{curr} = K_{curr}^{-1}} = \begin{bmatrix} \boldsymbol{F_{11}} & \boldsymbol{F_{12}} \\ \boldsymbol{F_{12}^T} & \boldsymbol{R_{22}} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{Fi_{11}} & \boldsymbol{Fi_{12}} \\ \boldsymbol{Fi_{12}^T} & \boldsymbol{Ri_{22}} \end{bmatrix}, \tag{8.20}$$

or

$$\begin{bmatrix} \boldsymbol{Fi_{11}} & \boldsymbol{Fi_{12}} \\ \boldsymbol{Fi_{12}^T} & \boldsymbol{Ri_{22}} \end{bmatrix} \begin{bmatrix} \boldsymbol{F_{11}} & \boldsymbol{F_{12}} \\ \boldsymbol{F_{12}^T} & \boldsymbol{R_{22}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}. \tag{8.21}$$

Suppose, if it is required to delete $\boldsymbol{F_{11}, F_{12}}$ and $\boldsymbol{F_{12}^T}$ from $\boldsymbol{K_{curr}}$ and to calculate the inverse of the remaining block $\boldsymbol{R_{22}}$ by reusing the currently available inverse $\boldsymbol{P_{curr}}$. Multiplying both sides of (8.21) with

$$\begin{bmatrix} \boldsymbol{I} & 0 \\ -\boldsymbol{Fi_{12}^T Fi_{11}^{-1}} & \boldsymbol{I} \end{bmatrix}, \tag{8.22}$$

and we get

$$\begin{bmatrix} \boldsymbol{Fi_{11}} & \boldsymbol{Fi_{12}} \\ \boldsymbol{0} & \boldsymbol{Ri_{22} - Fi_{12}^T Fi_{11}^{-1} Fi_{12}} \end{bmatrix} \begin{bmatrix} \boldsymbol{F_{11}} & \boldsymbol{F_{12}} \\ \boldsymbol{F_{12}^T} & \boldsymbol{R_{22}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{-Fi_{11}^{-1} Fi_{12}^T} & \boldsymbol{I} \end{bmatrix}. \tag{8.23}$$

From (8.23), $\boldsymbol{P_{new}}$ can be obtained after deletion of $\boldsymbol{F_{11}, F_{12}}$ and $\boldsymbol{F_{12}^T}$ from $\boldsymbol{K_{curr}}$ as follows:

$$\begin{aligned} \left(\boldsymbol{Ri_{22} - Fi_{12}^T Fi_{11}^{-1} Fi_{12}}\right) \boldsymbol{R_{22}} &= \boldsymbol{I}, \\ \boldsymbol{P_{new}} = \boldsymbol{R_{22}^{-1}} &= \boldsymbol{Ri_{22} - Fi_{12}^T Fi_{11}^{-1} Fi_{12}}. \end{aligned} \tag{8.24}$$

8.1.4 Decision Phase

After processing the training dataset $\boldsymbol{X_{curr}^{tr}}$ in the current window, output function can be written for any set of k samples $\boldsymbol{X_k = \{x_1, x_2, ..., x_k\}}$ as follows:

$$\boldsymbol{f(X_k)} = \begin{bmatrix} \boldsymbol{K(X_{curr}^{tr}, x_1)} \\ \vdots \\ \boldsymbol{K(X_{curr}^{tr}, x_k)} \end{bmatrix}^T \boldsymbol{PR}, \tag{8.25}$$

where $\boldsymbol{K(X_{curr}^{tr}, x_i)}$ denotes kernel vector for the i^{th} sample $\boldsymbol{x_i}$. Further, perform the following steps to decide whether any sample is outlier or not:

(i) We calculate distance (Λ) between the predicted value of the t^{th} training sample and R as follows:

$$\boldsymbol{\Lambda(x_t)} = |\boldsymbol{f(x_t)} - r| = \left|\hat{R}_t - r\right|.. \tag{8.26}$$

(ii) After calculating distances $\boldsymbol{\Lambda}$ as per (8.26), we sort the differences in decreasing order. Further, reject few percent of training samples based on the deviation. Most deviated samples are rejected first because they are most probably far from distribution of the target data. The threshold (θ) is decided based on these

deviations as follows:

$$\theta = \Lambda(\lfloor \nu * N \rfloor), \tag{8.27}$$

where $0 < \nu \leq 1$ is the fraction of rejection of training samples for deciding threshold value. N is the number of training samples and $\lfloor\ \rfloor$ denotes floor operation. We consider 5% of rejection, i.e. $\nu = 0.05$. Now, generate a decision function to decide whether any new sample $\boldsymbol{p}$ belongs to the target or outlier, where $\boldsymbol{z} = [z^1, z^2, ..., z^n]$. This function is defined by the following equation:

$$Sign(\theta - \Lambda(\boldsymbol{z})) = \begin{cases} 1, & \boldsymbol{p} \text{ is classified as target} \\ -1, & \boldsymbol{p} \text{ is classified as outlier.} \end{cases} \tag{8.28}$$

The above proposed approach is summarized as pseudo code in **Algorithm 8.1**.

Algorithm 8.1 OS-KOC: Boundary framework-based approach

Input: Training set $\boldsymbol{X}$= $(\boldsymbol{x}_1, c), (\boldsymbol{x}_2, c), \ldots, (\boldsymbol{x}_{N_0}, c), \ldots, (\boldsymbol{x}_t, c), \ldots$

Output: Whether each sample belongs to Target class or Outlier class

Initialization Phase

Pass initial set of samples $\boldsymbol{X_0, R_0}$ to the classifier as: $\{(\boldsymbol{x}_1, R_1), (\boldsymbol{x}_2, R_2), ..., (\boldsymbol{x}_{N_0}, R_{N_0})\}$ // For first chunk of N_0 samples, follow the steps as below.

Employ kernel feature mapping: $\boldsymbol{\Phi_0 = \phi(X_0)}$.

Output Weight $\boldsymbol{W_0}$ for $\boldsymbol{(X_0, R_0)}$:

$$\boldsymbol{W_0 \leftarrow P_0 R_0}$$
$$\boldsymbol{P_0 \leftarrow K_0^{-1}}$$

End of Initialization Phase

For second chunk onwards, following steps are required

$$\boldsymbol{K \leftarrow K_0}$$
$$\boldsymbol{P \leftarrow P_0}$$

for $i = 1$ to last chunk of data in $\boldsymbol{X}$ **do**

    Set size of chunk at the current stage as s

    Update the final kernel matrix $\boldsymbol{K}$ and its inverse $\boldsymbol{P}$ in two steps as given below:

    Step 1: Forgetting Phase

    Remove the impact of f old samples from the current inverse $\boldsymbol{P}$ using (8.24), i.e.,

$$\boldsymbol{P_{new} = R_{22}^{-1} = Ri_{22} - Fi_{12}^T Fi_{11}^{-1} Fi_{12}}$$

    Update the kernel matrix $\boldsymbol{K}$ by removing those rows and columns which were generated due to the old s samples.

Step 2: Retraining Phase

Update the kernel matrix $\boldsymbol{K}$ as per (8.11), i.e.,

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K_u} & \boldsymbol{K_{u,v}} \\ (\boldsymbol{K_{u,v}})^{\boldsymbol{T}} & \boldsymbol{K_v} \end{bmatrix}$$

Calculate $\boldsymbol{K_v}$ for i^{th} chunk by using (8.12)

Calculate $\boldsymbol{K_{u,v}}$ for i^{th} chunk by using (8.13)

Compute the inverse of updated kernel matrix $\boldsymbol{K}$, i.e., $\boldsymbol{P} = \boldsymbol{K^{-1}}$ using block inverse as discussed in (8.15)-(8.19)

$b = b + s$

Update the Output Weight using the value of $\boldsymbol{R}$ and updated $\boldsymbol{P}$ as:

$$\boldsymbol{\beta = PR}$$

Compute the predicted value by using output function $f(X_k)$, which is defined in (8.25)

Calculate distances (Λ) between predicted value of training sample and R as per (8.26), i.e.,

$$\boldsymbol{\Lambda(x_t)} = |\boldsymbol{f(x_t)} - r| = \left|\hat{R}_t - r\right|$$

Sort the distances in decreasing order

Compute θ using (8.27), i.e., $\theta = \Lambda(\lfloor \nu * N \rfloor)$

Use (8.28) to decide whether a new sample z belongs to target or not

$$Sign(\theta - \Lambda(\boldsymbol{z})) = \begin{cases} 1, & \boldsymbol{z} \text{ is classified as target} \\ -1, & \boldsymbol{z} \text{ is classified as outlier} \end{cases}$$

end for

8.2 Reconstruction Framework-based Approach: OS-AEKOC

In reconstruction framework-based method i.e. OS-AEKOC, model is trained by only target data $\boldsymbol{X}$ and approximates all data to itself. Figure 8.3 shows the architecture for OS-AEKOC. Here, the given stream of training data $\boldsymbol{X}$ is defined same as discussed above in the boundary framework. Input layer that takes data for t^{th} input sample is coded as $(\boldsymbol{x}_t, \boldsymbol{x}_t)$ because the target is identical to the input layer. Further, kernel feature mapping is employed between input and hidden layer, which is same as discussed in the above section. Output weight matrix $\boldsymbol{\beta_a}$ until t^{th} sample is represented as $(\beta_a)_t$. $\widehat{\boldsymbol{X}} = (\widehat{\boldsymbol{x}}_1, \widehat{\boldsymbol{x}}_2 ... \widehat{\boldsymbol{x}}_t, ...)$ is the set of predicted output vectors, where $\widehat{\boldsymbol{x}}_t = [\widehat{x}_{t1}, \widehat{x}_{t2}, ..., \widehat{x}_{tn}]$ is the predicted output vector for the t^{th} sample. $\widehat{\boldsymbol{c}} = [\widehat{c}_1, \widehat{c}_2 ... \widehat{c}_t, ...]$ is the predicted class vector, where $\widehat{c}_t$ is the predicted class for the t^{th} sample. Formulation of OS-AEKOC is discussed next.

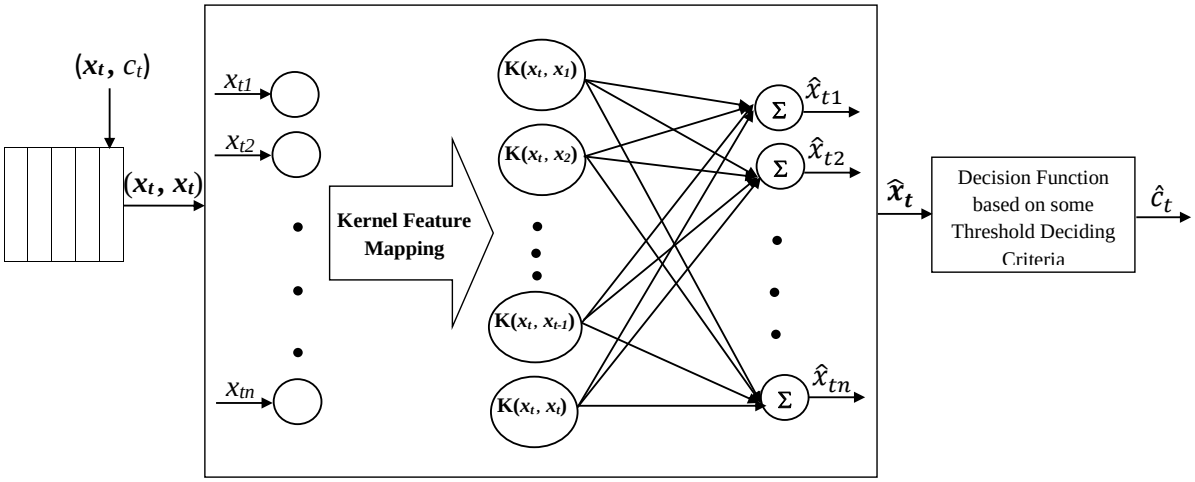


Figure 8.3: A schematic diagram of online sequential KRR-based Auto-Encoder for OCC: Reconstruction framework-based

Similar to OS-KOC, minimization problem of OS-AEKOC is written for kernel feature mapping using a non-linear feature mapping $\boldsymbol{X} \rightarrow \boldsymbol{\Phi}$ and representer theorem [103].

For the initial chunk $\{\boldsymbol{X_0}, \boldsymbol{X_0}\}$ of size N_0, the objective is to minimize the output weight $(\boldsymbol{\beta_a})_{\boldsymbol{0}}$ and the reconstruction error $\boldsymbol{E_0}$ between the expected $(\boldsymbol{X_0})$ and the predicted values $((\boldsymbol{\beta_a})_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{\Phi_0})$. Minimization problem for this can be written as follows:

$$\begin{aligned} \underset{(\boldsymbol{\beta_a})_{\boldsymbol{0}}, \boldsymbol{E_0}}{\text{Minimize}} : L &= \frac{1}{2}\left\|(\boldsymbol{\beta_a})_{\boldsymbol{0}}\right\|_F^2 + C\frac{1}{2}\left\|\boldsymbol{E_0}\right\|_F^2 \\ \text{Subject to} : \ & (\boldsymbol{\beta_a})_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{\Phi_0} = \boldsymbol{X_0^T} - \boldsymbol{E_0^T}, \end{aligned} \tag{8.29}$$

where C is used as a regularization parameter.

Next, Representer Theorem [103] is exploited, which describes the weight matrix $(\boldsymbol{\beta_a})_{\boldsymbol{0}}$ as a linear combination of the training data representation in non-linear space $(\boldsymbol{\Phi_0})$ and a reconstruction matrix $(\boldsymbol{W_a})_{\boldsymbol{0}}$. That is, the output weight matrix for the kernel feature mapping case is given as follows:

$$(\boldsymbol{\beta_a})_{\boldsymbol{0}} = \boldsymbol{\Phi_0}(\boldsymbol{W_a})_{\boldsymbol{0}}. \tag{8.30}$$

It is to be noted that $\boldsymbol{\beta_0}$ and $\boldsymbol{W_0}$ are weight vectors in OS-KOC due to single output node architecture, however, $(\boldsymbol{\beta_a})_{\boldsymbol{0}}$ and $(\boldsymbol{W_a})_{\boldsymbol{0}}$ are weight matrices in OS-AEKOC due to multi-output node architecture.

Thus, the minimization problem in (8.29) can be reformulated as follows:

$$\begin{aligned} \underset{(\boldsymbol{W_a})_{\boldsymbol{0}}, \boldsymbol{E_0}}{\text{Minimize}} : L &= \frac{1}{2}Tr\Big((\boldsymbol{W_a})_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{\Phi_0^T}\boldsymbol{\Phi_0}(\boldsymbol{W_a})_{\boldsymbol{0}}\Big) + C\frac{1}{2}\left\|\boldsymbol{E_0}\right\|_F^2 \\ \text{Subject to} : \ & (\boldsymbol{W_a})_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{\Phi_0^T}\boldsymbol{\Phi_0} = \boldsymbol{X_0^T} - \boldsymbol{E_0^T}. \end{aligned} \tag{8.31}$$

Now, by substituting kernel matrix $\boldsymbol{K_0} = (\boldsymbol{\Phi_0})^{\boldsymbol{T}}\boldsymbol{\Phi_0} = \boldsymbol{K}(\boldsymbol{X_0}, \boldsymbol{X_0})$ in the above equation, the following minimization problem is obtained:

$$\begin{aligned} \underset{(\boldsymbol{W_a})_{\boldsymbol{0}}, \boldsymbol{E_0}}{\text{Minimize}} : L &= \frac{1}{2}Tr\Big((\boldsymbol{W_a})_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{K_0}(\boldsymbol{W_a})_{\boldsymbol{0}}\Big) + C\frac{1}{2}\left\|\boldsymbol{E_0}\right\|_F^2 \\ \text{Subject to} : \ & (\boldsymbol{W_a})_{\boldsymbol{0}}^{\boldsymbol{T}}\boldsymbol{K_0} = \boldsymbol{X_0^T} - \boldsymbol{E_0^T}. \end{aligned} \tag{8.32}$$

By using Lagrangian relaxation [182], (8.32) can be written as:

$$\underset{\boldsymbol{\alpha}}{\text{Maximize}} \quad \underset{(\boldsymbol{W_a})_0, \boldsymbol{E_0}}{\text{Minimize}} : L = \frac{1}{2} Tr\Big((\boldsymbol{W_a})_0^{\boldsymbol{T}} \boldsymbol{K_0} (\boldsymbol{W_a})_0\Big) + C\frac{1}{2} \|\boldsymbol{E_0}\|_F^2 \\ -\boldsymbol{\alpha}((\boldsymbol{W_a})_0^{\boldsymbol{T}} \boldsymbol{K_0} - \boldsymbol{X_0^T} + \boldsymbol{E_0^T}), \tag{8.33}$$

where $\boldsymbol{\alpha} = [\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}, \ldots, \boldsymbol{\alpha_i}, \ldots, \boldsymbol{\alpha_{N0}}]$ are the Lagrange multipliers, and $\boldsymbol{\alpha_i} = [\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{in}]$, which are non-negative and employed to combine the constraint with the objective function. By using Karush-Kuhn-Tucker (KKT) theorem [183], take partial derivatives of (8.33) with respect to all variables $(\boldsymbol{W_a})_0, \boldsymbol{E_0}$ and α, i.e.,

$$\begin{aligned} &\frac{\partial L}{\partial (\boldsymbol{W_a})_0} = 0 \Rightarrow (\boldsymbol{W_a})_0 = \boldsymbol{\alpha}, \\ &\frac{\partial L}{\partial \boldsymbol{E_0}} = 0 \Rightarrow C\boldsymbol{E_0} = \boldsymbol{\alpha}, \\ &\frac{\partial L}{\partial \boldsymbol{\alpha}} = 0 \Rightarrow (\boldsymbol{W_a})_0^{\boldsymbol{T}} \boldsymbol{K_0} - \boldsymbol{X_0^T} + \boldsymbol{E_0^T} = 0. \end{aligned} \tag{8.34}$$

Following weight is obtained from solving equation in (8.34):

$$(\boldsymbol{W_a})_0 = \boldsymbol{P_0 X_0}, \text{where } \boldsymbol{P_0} = \left(\boldsymbol{K_0} + \frac{\boldsymbol{1}}{\boldsymbol{C}} \boldsymbol{I}\right)^{-\boldsymbol{1}}. \tag{8.35}$$

Finally, weight matrix for initial samples is obtained, which needs to be updated as below.

Initially, $\boldsymbol{K}$ and $\boldsymbol{P}$ are calculated as per (8.35). Both are updated continuously for any upcoming new samples $\boldsymbol{X^v}$ as per (8.11)-(8.19). After processing the training dataset $\boldsymbol{X_{curr}^{tr}}$ in the current sliding window, output function for any set of k samples $X_k = \{x_1, x_2, \ldots, x_k\}$ can be written as follows:

$$\boldsymbol{f(X_k)} = \begin{bmatrix} \boldsymbol{K(X_{curr}^{tr}, x_k)} \\ \vdots \\ \boldsymbol{K(X_{curr}^{tr}, x_1)} \end{bmatrix}^T \boldsymbol{P X_{curr}^{tr}} \tag{8.36}$$

where $\boldsymbol{K(X_{curr}^{tr}, x_i)}$ denotes kernel vector for the i^{th} sample $\boldsymbol{x_i}$. Afterwards, we calculate error and decide whether any sample belongs to target class or outlier class

as per pseudo code discussed in **Algorithm 8.2**.

Algorithm 8.2 OS-AEKOC: Reconstruction framework-based approach

Input: Training set $\boldsymbol{X}= (\boldsymbol{x}_1, c_1), (\boldsymbol{x}_2, c_2), \ldots, (\boldsymbol{x}_{N_0}, c_{N_0}), \ldots, (\boldsymbol{x}_t, c_t), \ldots$
Output: Whether each sample belongs to Target class or Outlier class

Initialization Phase
Pass initial set of samples $\boldsymbol{X_0}$ to the classifier as: $\boldsymbol{\{(x_1, x_1), (x_2, x_2), ..., (x_{N_0}, x_{N_0})\}}$
For first chunk of N_0 samples, follow the steps as below.
Employ kernel feature mapping: $\boldsymbol{\Phi_0 = \phi(X_0)}$.
Output Weight $\boldsymbol{(W_a)_0}$ for $\boldsymbol{(X_0, X_0)}$:

$$\boldsymbol{(W_a)_0 \leftarrow P_0 X_0}$$
$$\boldsymbol{P_0 \leftarrow K_0^{-1}}$$

End of Initialization Phase
For second chunk onwards, following steps are required

$$\boldsymbol{K \leftarrow K_0}$$
$$\boldsymbol{P \leftarrow P_0}$$

for $i = 1$ to last chunk of data in $\boldsymbol{X}$ **do**
    Set size of chunk at the current stage as s
    Update the final kernel matrix $\boldsymbol{K}$ and its inverse $\boldsymbol{P}$ in two steps as given below:
Step 1: Forgetting Phase
    Remove the impact of f old samples from the current inverse $\boldsymbol{P}$ using (8.24), i.e.,

$$\boldsymbol{P_{new} = R_{22}^{-1} = Ri_{22} - Fi_{12}^T Fi_{11}^{-1} Fi_{12}}$$

    Update the kernel matrix $\boldsymbol{K}$ also by removing those rows and columns which were generated due to the old s samples.
Step 2: Retraining Phase
    Update the kernel matrix $\boldsymbol{K}$ as per (8.11), i.e.,

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K_u} & \boldsymbol{K_{u,v}} \\ \boldsymbol{(K_{u,v})^T} & \boldsymbol{K_v} \end{bmatrix}$$

    Calculate $\boldsymbol{K_v}$ for i^{th} chunk by using (8.12)
    Calculate $\boldsymbol{K_{u,v}}$ for i^{th} chunk by using (8.13)
    Compute the inverse of updated kernel matrix $\boldsymbol{K}$, i.e., $\boldsymbol{P = K^{-1}}$ using block inverse as discussed in (8.15)-(8.19)
    $b = b + s$
    Update the Output Weight using the value of $\boldsymbol{X_{curr}^{tr}}$ and updated $\boldsymbol{P}$ as $\boldsymbol{(W_a) = PX_{curr}^{tr}}$
    Compute the predicted value by using output function $f(X_k)$, which is defined in (8.36)

20: Calculate sum of square error (Λ) between predicted ($\hat{\boldsymbol{x}}_{\boldsymbol{t}}$) and actual value ($\boldsymbol{x}_t$) of t^{th} training sample:

21:

$$\Lambda(\boldsymbol{x}_t) = \sum_{j=1}^{n} (x_t^j - \hat{x}_t^j)^2 \tag{8.37}$$

22: Sort the distances in decreasing order

23: Compute θ using (8.27), i.e., $\theta = \Lambda(\lfloor \nu * N \rfloor)$

24: Use (8.28) to decide whether a new sample z belongs to target or not

$$Sign(\theta - \Lambda(\boldsymbol{z})) = \begin{cases} 1, & \boldsymbol{z} \text{ is classified as target} \\ -1, & \boldsymbol{z} \text{ is classified as outlier} \end{cases}$$

25: **end for**

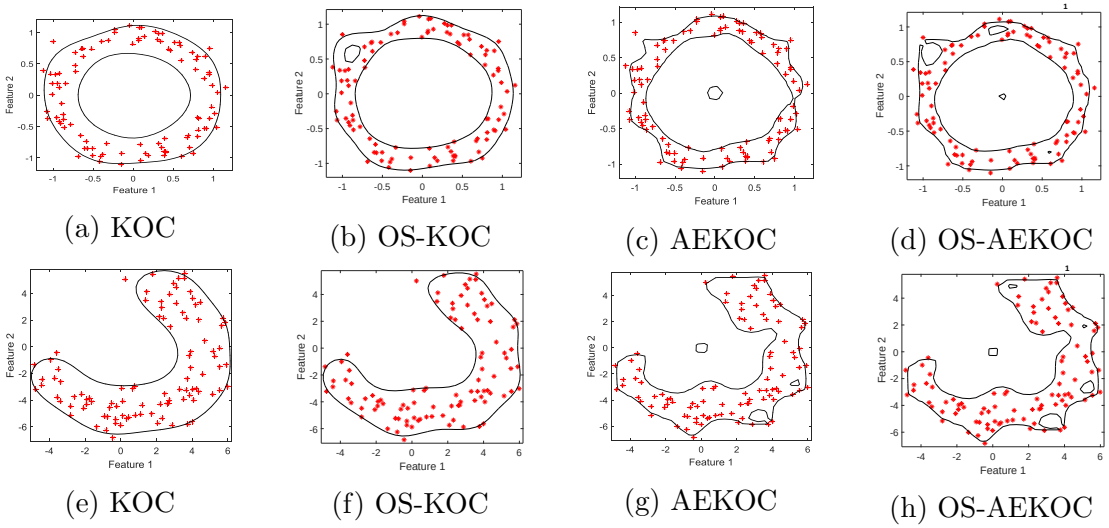
8.3 Experiments

In this section, the performance of the classifiers is tested on both types of the datasets, i.e. synthetic and real-world. Sixteen synthetic and four real-world datasets are used for the non-stationary environment (see Table 8.1). We also test on two synthetic datasets from the stationary environment for verifying the boundary construction capability of the classifier. All experiments are executed on MATLAB 2014b in Windows 7 (64 bit) environment with 64 GB RAM, 3.00 GHz Intel Xeon processor. All existing and proposed classifiers are implemented and tested in the same environment.

8.3.1 Boundary Construction on Synthetic Stationary Datasets

As one-class classifiers are also called as data descriptors, and hence, the proposed methods are first tested on stationary synthetic datasets to verify their data describing (boundary creation) ability. The motive to test on the stationary datasets is to show that the boundary construction capability of online classifiers is at least as good as their corresponding batch learning-based classifiers. Here, OS-KOC, OS-AEKOC are the online version of KOC [7] and AEKOC (Chapter 3), respectively. Therefore, we compare these classifiers for verifying boundary construction capability. It can be

easily visualized in Figure 8.4 that the proposed online classifiers have created similar boundary like their corresponding offline one-class classifiers. Boundary creation for ring dataset is challenging task for one-class classifiers as ring dataset can be covered by a single circle as well as by two concentric circles. However, two concentric circles are a more appropriate boundary, which can be seen in Figure 8.4(a) to 8.4(d). It can be observed in Figure 8.4(a) to 8.4(h) that the boundary framework-based classifiers create smoother boundary compared to the reconstruction framework-based classifier for both the synthetic datasets. Banana dataset checks the ability of the classifier to create a convex boundary. Overall, it can be observed from these figures that the proposed online one-class classifiers have similar boundary construction capability as the offline one-class classifiers.



(a) KOC (b) OS-KOC (c) AEKOC (d) OS-AEKOC

(e) KOC (f) OS-KOC (g) AEKOC (h) OS-AEKOC

Figure 8.4: Performance of the proposed online classifiers on synthetic dataset: Boundary and reconstruction framework-based approaches

8.3.2 Performance Comparison on Non-stationary Synthetic Datasets

The proposed online one-class classifiers are tested on non-stationary datasets to verify two things; first, adaptiveness and second, handling capability of large streaming

Table 8.1: Dataset description for non-stationary datasets

Synthetic Non-stationary Datasets					
Dataset Name	#Feat.	Drift	#Target	#Outlier	#Records
1CDT [49]	2	400	8000	8000	16,000
1CHT[49]	2	400	8000	8000	16,000
4CE1CF[49]	2	750	34650	138600	173,250
4CR[49]	2	400	36100	108300	144,400
GEARS-2C-2D[186]	2	2,000	100000	100000	200,000
4CRE-V1[49]	2	1,000	31250	93750	125,000
4CRE-V2[49]	2	1,000	45750	137250	183,000
5CVT[49]	2	1,000	8000	16000	24,000
2CDT [49]	2	400	8000	8000	16,000
2CHT[49]	2	400	8000	8000	16,000
1CSurr[49]	2	≈600	20200	35083	55,283
UG-2C-2D[186]	2	1,000	50000	50000	100,000
UG-2C-3D[186]	3	2,000	100000	100000	200,000
UG-2C-5D[186]	5	2,000	100000	100000	200,000
MG-2C-2D[186]	2	2,000	100000	100000	200,000
FG-2C-2D[187]	2	2,000	150000	50000	200,000

Real Non-stationary Datasets					
	#Feat.	Drift	#Target	#Outlier	#Records
Electricity [188]	8	Real	19237	26075	45,312
Poker-hand[170]	10	Real	415526	413675	829,201
Keystroke[189]	10	Real	400	1200	1600
Abalone[170]	10	Real	2770	1407	4177

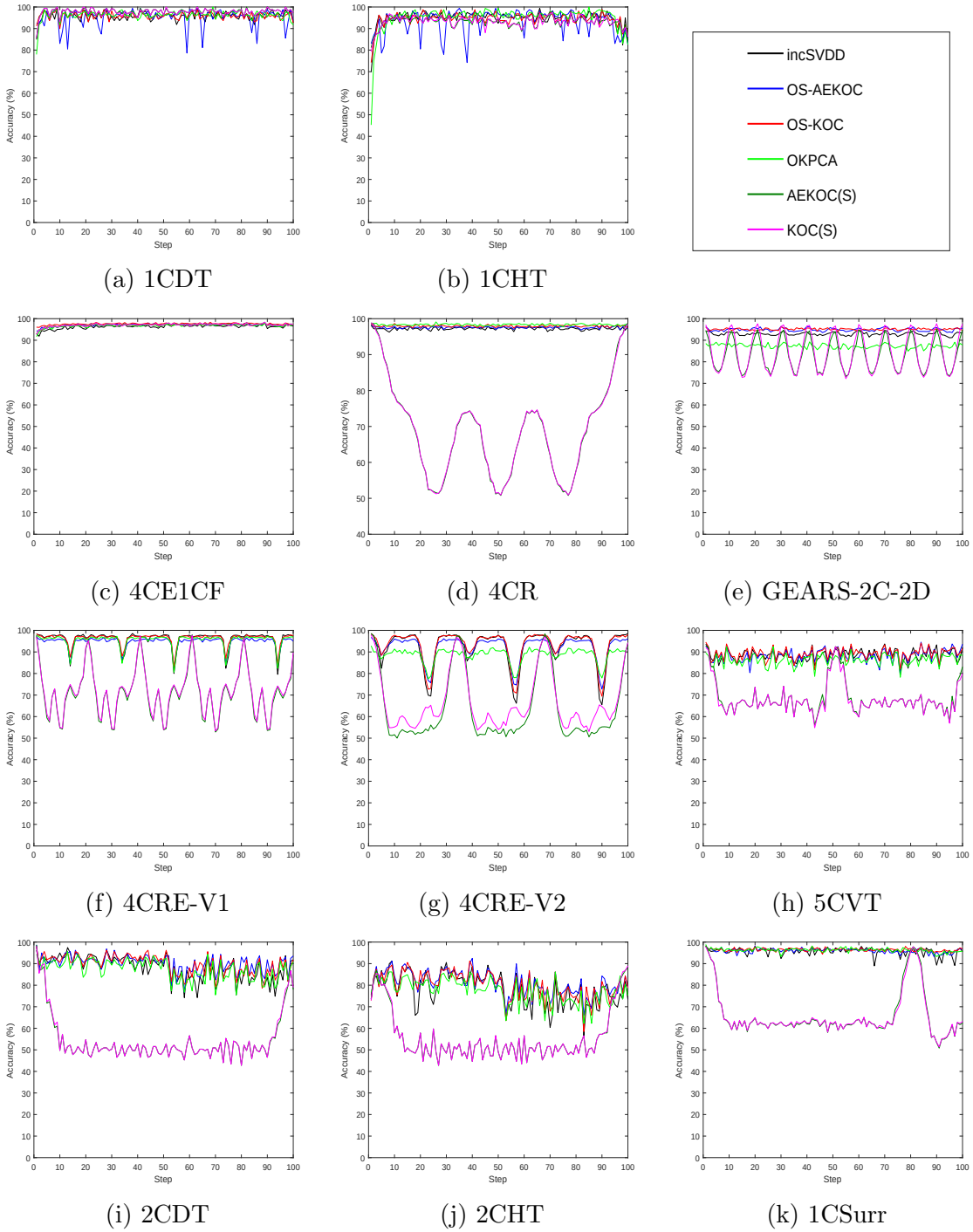
datasets with limited memory. All non-stationary synthetic datasets have different types of drift associated with them like periodic drift, non-periodic drift etc. Drift occurrence of each dataset is briefly described in Table 8.1 and also visualized on the web page (`https://goo.gl/j2wQw4`). The synthetic datasets used in this chapter are taken from ([49, 186, 187] and [189]). All synthetic and real datasets were originally proposed for binary or multi-class classification in the non-stationary environment. We modify these datasets for the OCC task by assuming one-class as a normal class and the remaining all classes as an outlier class.

Classifiers are experimented in two modes viz., static (i.e. offline classifiers) and sliding window (i.e. online classifiers). One-class classifiers in static mode are named as KOC(S) and AEKOC(S). Here, 'S' stands for static mode. In static mode, one-class classifiers are just trained on initially available samples and then tested on all remaining samples like batch learning. OS-KOC, OS-AEKOC, incSVDD [40] and OKPCA [160] are tested in the sliding window mode. Here, the classifiers adapt to the environment as per the upcoming new samples in the sliding window and forget the old samples. Except for the large real-world datasets, the sliding window and the block size are fixed as 150 and 50, respectively (see the last two columns of Table 8.2 and 8.3).

The accuracy of these datasets is plotted in Figure 8.5 and 8.6. These plots are created by dividing the dataset into hundred batches and the mean accuracy achieved by all the batches in 100 steps is plotted. Here, the y-axis and the x-axis denote accuracy and steps, respectively for all of the plots in Figure 8.5 and 8.6.

Next, we discuss results for different datasets based upon the nature of the drift.

(a) **Drift only in the outlier class:** The three datasets that have this behavior are 1CDT, 1CHT, and 4CE1CF. It can be observed from Table 8.2 and Figure 8.5(a), 8.5(b), 8.5(c) that one-class classifiers in static mode yield similar or comparative results as a sliding window mode for these datasets. This is because the normal class in these datasets are not changing their distribution over time, only the outlier class changes its distribution. However, the same can't be stated for the remaining synthetic datasets as the distribution of both classes is changing in



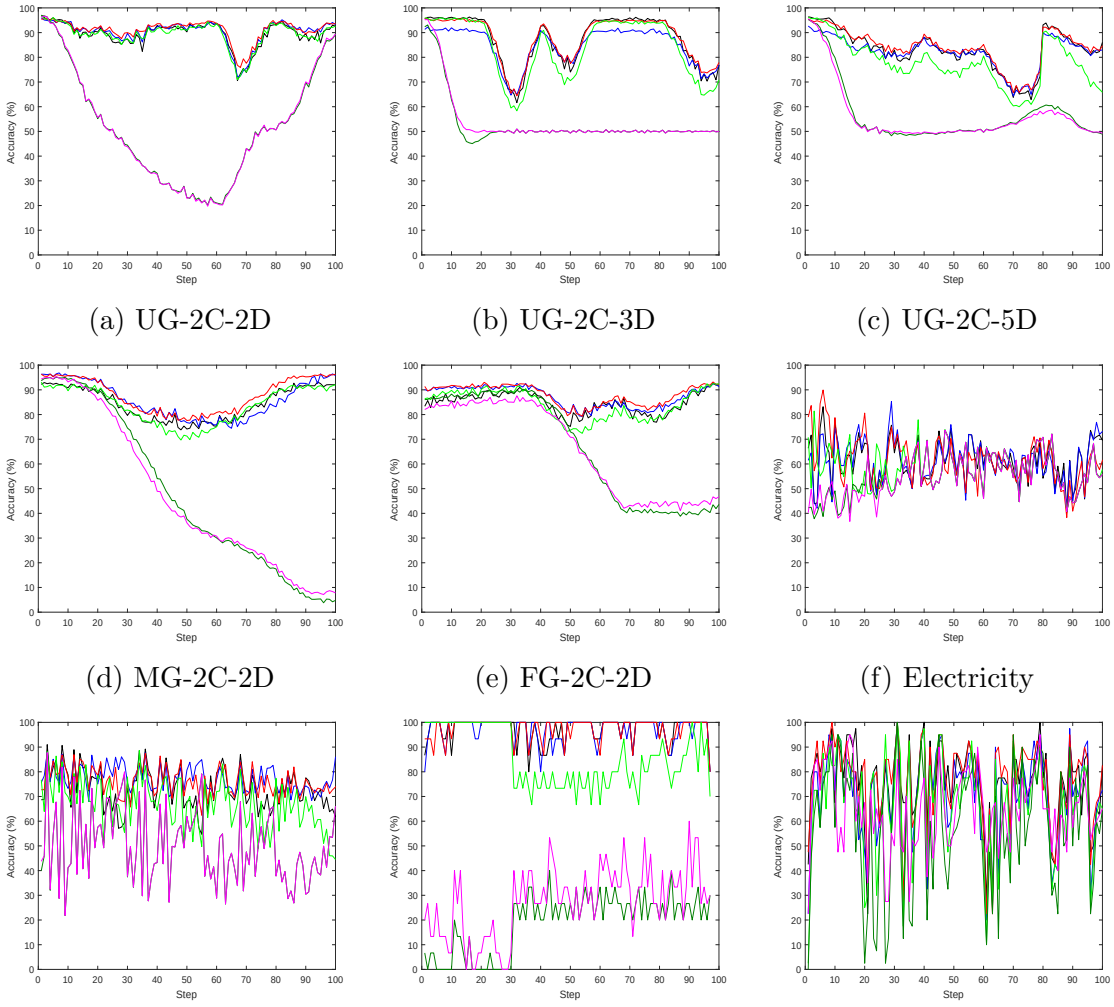
(a) 1CDT (b) 1CHT

(c) 4CE1CF (d) 4CR (e) GEARS-2C-2D

(f) 4CRE-V1 (g) 4CRE-V2 (h) 5CVT

(i) 2CDT (j) 2CHT (k) 1CSurr

Figure 8.5: Performance of one-class classifiers on all datasets in 100 steps. This figure is continued to Figure 8.6



(a) UG-2C-2D (b) UG-2C-3D (c) UG-2C-5D

(d) MG-2C-2D (e) FG-2C-2D (f) Electricity

(g) Poker-hand (h) Keystroke (i) Abalone

Figure 8.6: Continuation of Figure 8.5

the remaining datasets. OKPCA performs better for 1CDT and 1CHT datasets, however, OS-KOC performs better for 4CE1CF datasets compared to the other online one-class classifiers, which are present in Table 8.2.

(b) **Periodic drift in outlier and non-periodic drift in the normal class:** The dataset which has this behavior is 4CR. For 4CR dataset, the normal class rotates on a circular path and three different distributions of outlier also rotate on the circular path but they create a periodic drift collectively. Periodic drift can also be visualized by observing the repeating nature of the generated performance plot for 4CR dataset of the classifiers in the static mode (see Figure 8.5(d)). As expected static mode doesn't perform well for this dataset and the proposed classifiers yield better accuracy compared to all except OKPCA.

(c) **Periodic drift in both the normal and the outlier class:** The four datasets that have this behavior are Gears-2C-2D, 4CRE-V1, 4CRE-V2 and 5CVT. For these datasets, classifiers in static mode yield inferior performance compared to sliding window mode classifiers due to the periodic drift occurrence in the normal as well as the outlier class, which can also be noticed in the Figure 8.5(e), 8.5(f), 8.5(g), and 8.5(h) (the performance dips down at the regular periodic interval for static mode). However, the proposed and existing online classifiers smoothly adapt to this periodic drift. The adaptation of OS-KOC in the non-stationary environment has been visualized for Gears-2C-2D dataset along with some other datasets on the web page(`https://goo.gl/8bvkps`). As the Gears-2C-2D dataset has a smaller drift, and therefore, classifiers in static mode perform better for this dataset compared to other three synthetic datasets. Boundary framework-based one-class classifier, i.e. OS-KOC outperforms all the present classifiers in Table 8.2. Specifically, OS-KOC outperforms the OKPCA by significant margin, i.e. 4.51%, 2.83%, 7.74%, and 0.87%, for 4CRE-V2, 5CVT, Gears-2C-2D, and 4CRE-V1, respectively.

(d) **Non-periodic drift in both the normal and the outlier class:** The two datasets that have this behavior are 2CDT and 2CHT. Drift continuously increases

diagonally and horizontally for these datasets, respectively. Adaption of OS-KOC on 2CDT dataset is visualized in the Figure 8.7. Here, all plotted samples belong to the normal class but at distinct timestamps. Two colors are used to discriminate the normal samples at two consecutive timestamps. Reconstruction framework-based classifier, i.e. OS-AEKOC performs best among all the classifiers and our proposed classifiers outperform incSVDD and OKPCA significantly by more than 2% and 3% margin for 2CDT and 2CHT datasets, respectively. Among all sixteen synthetic datasets, proposed and existing one-class classifiers achieve the least accuracy for 2CHT dataset, i.e. when horizontal non-periodic drift occurs.

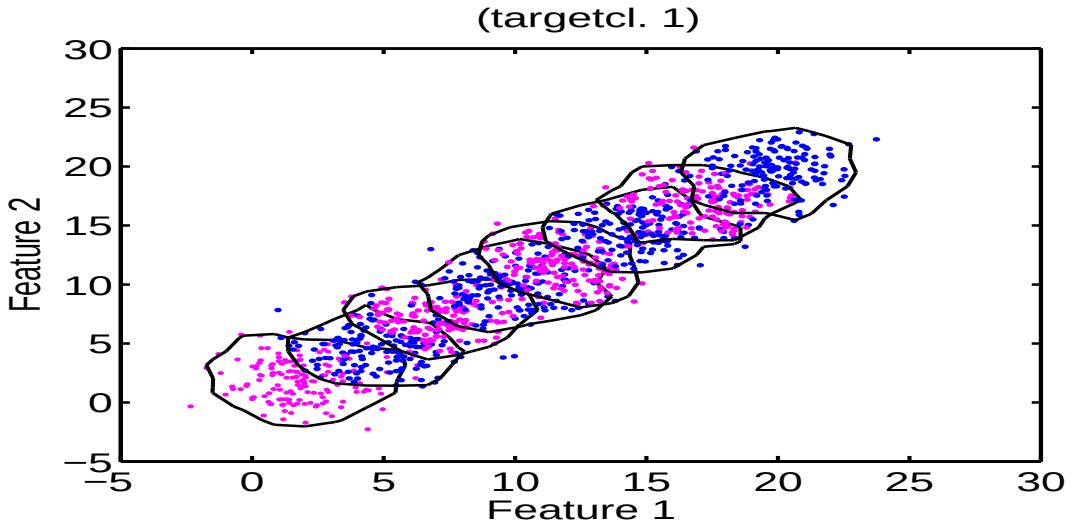


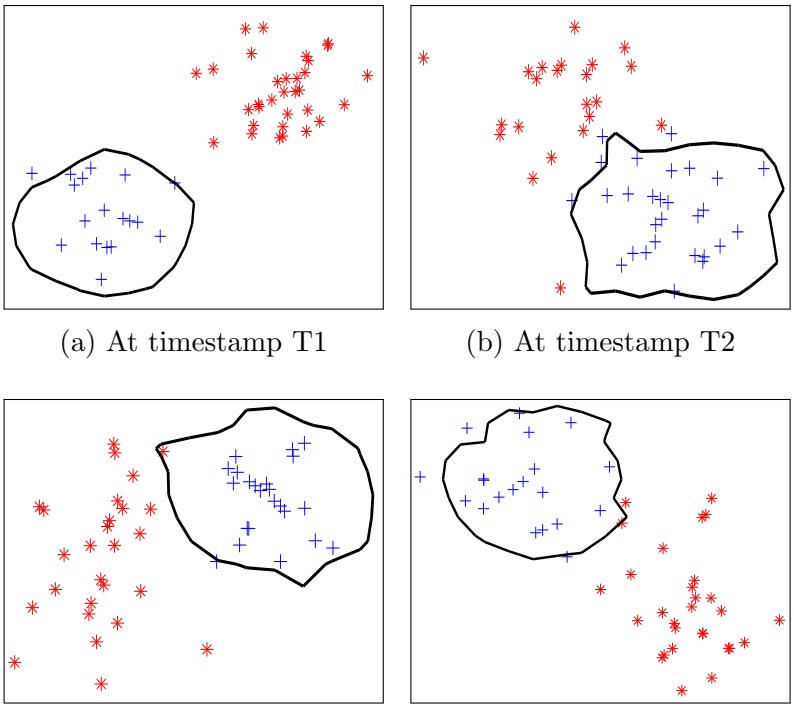
Figure 8.7: Adaption of OS-KOC on 2CDT dataset

(e) **Drift created by surrounding the normal class by outlier:** In 1CSurr dataset, normal and outlier, both classes are drifting and normal data is compactly surrounded by outliers in a non-periodic way. OS-KOC performs best among all, however, OS-AEKOC yields better accuracy than incSVDD but inferior to OKPCA.

Next, we discuss results for different datasets based upon the occurrence of drift with the Gaussian distribution.

(a) **Drift as per Unimodal Gaussian distribution of the normal and the outlier class:** Normal and the outlier classes are generated by unimodal Gaussian

distribution for three datasets viz., UG-2C-2D, UG-2C-3D, and UG-2C-5D, of two, three and five dimensions, respectively. Distribution of UG-2C-2D and the adaptive boundary created by the OS-KOC on this dataset for different timestamps are visualized in the Figure 8.8. Distribution of these datasets are also visualized on this web page (`https://goo.gl/j2wQw4`) and adaptiveness of OS-KOC on UG-2C-2D is visualized on this web page (`https://goo.gl/8bvkps`).



(a) At timestamp T1 (b) At timestamp T2

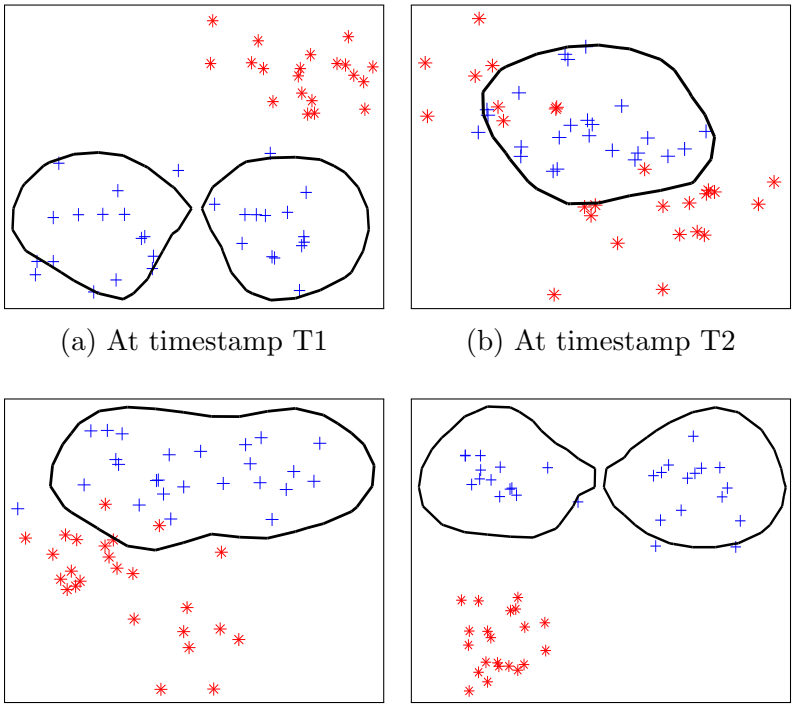
(c) At timestamp T3 (d) At timestamp T4

Figure 8.8: Adaption on UG-2C-2D dataset

OS-KOC performs best among all and outperforms incSVDD and OKPCA for all three datasets. Overall, OS-AEKOC performs well on the two-dimensional dataset, however, for three and five-dimensional datasets, i.e. UG-2C-3D and UG-2C-5D, it still performs better than OKPCA but inferior to incSVDD.

(b) **Drift as per Multi-modal Gaussian distribution of the normal and the outlier class:** The normal and the outlier classes are generated by multi-modal Gaussian distribution for two datasets viz., MG-2C-2D and FG-2C-2D. Distribution of MG-2C-2D and the adaptive boundary created by the OS-KOC on this

dataset for different timestamps are visualized in the Figure 8.9. In the MG-2C-2D dataset, normal and outlier both show multi-modal behavior alternatively. However, in FG-2C-2D dataset, multi-modality is only present in the normal class, which is represented by four distinct Gaussian distribution (The outlier class has single Gaussian distribution). More detailed visualization of both datasets as well as behavior of OS-KOC on these datasets are available on these web pages(`https://goo.gl/j2wQw4` and `https://goo.gl/8bvkps`). OS-KOC outperforms incSVDD by more than 3% and OKPCA by nearly 4% of margin. After 2CHT dataset, classifiers yield the least accuracy on MG-2C-2D dataset due to the presence of multimodality in this dataset.



(a) At timestamp T1 (b) At timestamp T2

(c) At timestamp T3 (d) At timestamp T4

Figure 8.9: Adaption on MG-2C-2D dataset

8.3.3 Drift in non-stationary real world datasets

Four real-world datasets are employed for the performance analysis. Electricity dataset is a well know dataset for streaming data analysis. It is collected from mar-

kets where the price varies as per demand and supply. The main task here is to identify the change of the price relative to a moving average of the last 24 hours. Proposed classifiers clearly outperform static mode based classifiers as well as other online classifiers available in Table 8.3. In Poker-hand dataset, the original dataset contains ten classes where one class belongs to non-poker hand and the remaining classes belong to nine types of poker hands. We club these nine poker hands and treat them as a single class. The main task here is to identify the poker hand after training the one-class classifier on non-poker hand. As seen in Table 8.3 and Figure 8.6(g), OS-AEKOC performs best among all the classifiers for this dataset and both of the proposed classifiers outperform incSVDD and OKPCA by the significant margin of more than 3% and 10% respectively.

The third dataset is a Keystroke dynamics dataset, where user verification is based on the typing rhythm of the user instead of the traditional way (login id and password) without any extra cost of hardware. We need to update the user profile regularly as typing rhythm of a user evolves over time, and therefore, the distribution also changes and drift occurs. The classifier is trained on the data of one user and then the trained model is used for the verification purpose of that user. As seen in Table 8.3 and Figure 8.6(h), OS-KOC yields the best accuracy among all and significantly outperforms OKPCA, however, yields slightly better accuracy compared to incSVDD. The fourth dataset, i.e. Abalone dataset, contains originally twenty-nine classes, which are converted for the OCC task by considering classes 9-29 as the normal class and class 1-8 as the outlier class. As seen in Table 8.3 and Figure 8.6(i), OS-KOC performs best among all for this dataset, however, OS-AEKOC performs better than OKPCA but inferior to incSVDD.

8.3.4 Efficiency Analysis

Time efficiency is the main concern in stream processing for various applications. The proposed classifiers are very simple and computationally more efficient compared to other classifiers. Since the presented online classifiers are based on KRR, they inherit the fast learning property of KRR also. For verifying this fact in an unbiased

Table 8.2: Accuracy of one-class classifiers over 16 synthetic non-stationary datasets

Datasets	OS-KOC	KOC (S)	OS-AEKOC	AEKOC (S)	incSVDD	OKPCA	Chunk Size	Sliding Window size
1CDT	95.96	97.73	95.30	**97.79**	95.85	**96.22**	50	150
2CDT	89.84	54.38	**90.22**	54.24	88.17	87.34	50	150
1CHT	94.56	93.16	93.60	93.11	94.89	**94.97**	50	150
2CHT	80.04	55.64	**81.08**	55.61	78.06	77.50	50	150
4CR	97.95	69.23	97.58	69.17	97.26	**98.34**	50	150
4CRE-V1	**96.27**	71.99	94.74	71.44	96.10	95.40	50	150
4CRE-V2	**93.19**	67.11	92.26	63.76	92.25	88.68	50	150
5CVT	**89.22**	68.26	88.41	68.68	88.55	86.38	50	150
1CSurr	**96.46**	66.00	95.99	66.06	95.70	96.27	50	150
4CE1CF	**97.58**	97.09	97.12	97.03	96.32	96.75	50	150
FG-2C-2D	**88.04**	65.78	87.21	66.49	84.72	84.09	50	150
UG-2C-2D	**91.22**	51.16	90.42	51.13	89.54	89.36	50	150
UG-2C-3D	**87.77**	53.57	84.84	52.99	87.45	84.11	50	150
UG-2C-5D	**84.12**	56.09	82.14	56.74	83.10	77.79	50	150
MG-2C-2D	**87.99**	46.80	86.06	47.56	84.51	83.79	50	150
GEARS -2C-2D	**95.12**	84.59	94.89	83.22	92.99	87.37	50	150

Table 8.3: Accuracy of one-class classifiers over real world non-stationary datasets

Datasets	OS-KOC	KOC	OS-AEKOC	AEKOC	incSVDD	OKPCA	Chunk Size	Sliding Window size
Electricity	**61.64**	55.03	61.49	55.14	61.25	58.67	200	2500
Poker-hand	76.12	49.96	**77.18**	49.90	73.29	66.95	200	2500
Keystroke	**97.38**	27.38	96.97	18.83	97.31	85.86	50	150
Abalone	**76.73**	64.27	70.72	57.66	75.32	66.77	50	150

manner, existing classifiers are implemented and tested in the same environment as the proposed classifiers. We have reported time consumed by the classifiers in three parts viz., training, forgetting and testing time in Table 8.4. We are reporting forgetting time explicitly in Table 8.4 since it is crucial in online learning.

The reason behind computational efficiency of the proposed algorithms can be understood by the analysis of (8.18) and (8.19). Recall, the complexity of matrix inversion and matrix multiplication is $\mathcal{O}(n^3)$. However, this complexity is further reduced in three ways. First, if any multiplication algorithm (like Strassen matrix multiplication) of complexity $\mathcal{O}(n^{2+\epsilon})$ is used, then the complexity of block matrix inversion is further reduced to $\mathcal{O}(n^{2.807})$ [190]. Second, Woodbury formula [185] is used to solve (8.18) as shown in (8.19). After using Woodbury formula [185] in (8.19), the inverse of two matrices, i.e. $\boldsymbol{K_u}$ and $\boldsymbol{K_v}$ are required to be computed. Here, $\boldsymbol{K_u}$ is the larger matrix of the size of the previous sliding window and $\boldsymbol{K_v}$ is the smaller matrix of the size of newly added samples for training. As per **Algorithm 8.1** and **Algorithm 8.2**, $\boldsymbol{K_u^{-1}}$ is already computed during the calculations for the previous sliding window. Therefore, there is no need to compute $\boldsymbol{K_u^{-1}}$ for the current window and only the inverse of the smaller size of the matrix $\boldsymbol{K_v}$ needs to be calculated, which is the third improvement. Similarly, the forgetting mechanism in (8.24) is also employed by available inverses instead of calculating inverses explicitly. Hence, the overall computational time cost is further reduced to $\mathcal{O}(n^2)$. Moreover, Time is also saved during kernel matrix calculation as it reuses the previously computed kernel matrix for the data from the previous sliding window. Next, we discuss storage cost.

As presented classifiers are based on online learning, hence, training and testing can be performed on the fly. It means that there is no need to store any old obsolete training samples after once it has been processed. Proposed classifiers simply forget the old samples and adapt to the incoming samples. The number of training samples that need to be learned by the classifiers in one batch can be fixed using the size of the sliding window (as per memory constraint of the system). Therefore, the proposed classifiers are trained with less memory requirement.

Table 8.4: Training, forgetting and testing time (in sec.) with sliding window size=150 and chunk size = 50

Datasets	***OS-KOC***			***OS-AEKOC***			***incSVDD***			***OKPCA***		
	Training	Forgetting	Testing	Training	Forgetting	Testing	Training	Forgetting	Testing	Training	Forgetting	Testing
1CDT	1.47	0.64	128.55	1.40	0.52	126.51	9.05	8.31	155.24	8.47	6.83	148.42
2CDT	1.42	0.64	127.64	1.48	0.59	120.43	8.40	7.83	142.28	7.52	6.21	132.56
1CHT	1.41	0.60	126.13	1.35	0.51	124.25	9.06	8.35	152.65	8.13	7.62	142.03
2CHT	1.39	0.64	126.64	1.44	0.59	119.50	8.50	7.83	141.15	7.85	6.73	129.16
4CR	11.48	5.27	1149.44	12.27	5.15	1091.52	37.31	34.71	1292.02	31.52	28.86	1214.37
4CRE-V1	9.06	3.93	873.20	9.26	3.66	858.66	32.04	30.13	1045.92	28.91	25.43	924.82
4CRE-V2	13.87	6.17	1371.48	14.34	5.94	1323.65	46.88	43.76	1584.57	41.63	39.16	1489.74
5CVT	1.83	0.81	170.08	1.80	0.72	165.89	8.34	7.88	201.62	7.29	6.84	190.29
1CSurr	4.46	1.90	403.17	4.26	1.66	394.50	20.90	19.34	480.99	17.41	15.87	462.38
4CE1CF	11.62	5.06	1151.00	11.51	4.57	1146.30	134.45	112.72	1542.84	123.28	109.37	1378.24
FG-2C-2D	16.58	6.64	1408.04	16.78	6.25	1385.00	141.94	129.59	1685.98	133.26	116.88	1485.92
UG-2C-2D	8.05	3.40	720.91	8.03	3.15	707.01	54.74	52.86	856.55	42.89	38.27	802.46
UG-2C-3D	15.71	6.48	1415.80	15.85	6.25	1391.73	94.47	86.05	1693.65	89.92	81.36	1452.67
UG-2C-5D	15.70	6.41	1404.62	15.36	5.93	1383.24	96.68	89.43	1688.55	90.73	86.34	1534.72
MG-2C-2D	15.69	6.57	1416.33	15.24	5.86	1389.52	93.32	83.85	1701.95	87.48	81.37	1624.82
GEARS -2C-2D	15.36	6.39	1391.69	14.97	5.77	1371.26	93.75	84.74	1682.46	88.17	81.93	1573.29
Electricity	3.75	1.62	341.49	3.71	1.46	334.26	19.40	17.19	404.73	16.81	14.85	381.43
Poker	765.35	329.13	6119.51	768.97	349.61	6045.79	789.16	798.12	7025.56	726.34	705.87	6826.31
Keystroke	0.13	0.08	10.66	0.09	0.04	10.35	0.43	0.28	12.55	0.41	0.23	11.03
Abalone	0.40	0.25	32.81	0.40	0.16	30.96	2.79	2.47	36.47	1.97	1.74	33.16

8.4 Summary

This chapter has presented online sequential learning with KRR for OCC. Two methods, viz., OS-KOC and OS-AEKOC, have been developed so far in this chapter, which are based on boundary and reconstruction frameworks. A forgetting mechanism is also embedded with these classifiers to remove the impact of obsolete old samples, which helps in smooth adaptation to the non-stationary environment. The proposed classifiers can handle data on the fly in an online and efficient manner for both stationary and non-stationary types of datasets. Performance evaluation over stationary datasets has exhibited that these online classifiers are equally capable as offline classifiers. Further, performance evaluation on non-stationary streaming datasets has exhibited that these classifiers are capable of handling large-sized datasets under system memory constraint. The proposed online one-class classifiers either outperformed existing online one-class classifiers (for most datasets) or yielded similar results (for some datasets). Overall, boundary framework-based one-class classifiers (OS-KOC) have performed better compared to reconstruction framework-based one-class classifiers (OS-AEKOC) as well as state-of-the-art online one-class classifiers. As computational cost is the primary concern in streaming data analysis, evaluation of the proposed classifiers verified that these are fast and computationally efficient as compared to other classifiers. Hence, it can be stated that the classifiers presented in this chapter are a viable alternative to the existing one-class classifiers.

Chapter 9

Conclusions and Future Work

This thesis primarily investigates the iterative (SVM-based) and non-iterative kernel learning-based approach (KRR-based) for outlier detection (i.e., identify the unknown) using OCC. In particular, we have developed various types of single and multi-layer methods for OCC using boundary and reconstruction frameworks. First, we have developed representation learning-based single (AEKOC) and multi-layer (MKOC) methods for OCC. Then, Multi-layer architecture is developed by stacking multiple KRR-based Auto-Encoder sequentially. Further, Graph-Embedding has been explored with this multi-layer architecture (GMKOC) using various types of Laplacian graphs. After this, we have explored multiple kernel learning approach for one-class classification (LMKAD and LMSVDD) where multiple kernels are employed simultaneously instead of sequentially. We have also extended the proposed single hidden layer-based architecture based one-class classifier for utilizing privileged information. For this purpose, the LUPI framework has been explored for OCC (i.e., KOC+ and AEKOC+). Finally, to tackle the challenges of non-stationary data streams, we have extended two KRR-based one-class classifiers (KOC and AEKOC) for online sequential learning (OS-KOC and OS-AEKOC). These proposed methods have been tested on various benchmark datasets and compared results with various kernel-based state-of-the-art OCC methods. Results analysis exhibits that the proposed methods have outperformed existing state-of-the-art kernel-based methods in terms of Gmean (η_g), mean of Gmean (η_m) and Friedman Rank (η_f).

9.1 Summary of Research Achievements

The objectives specified in Section 1.3 have been successfully fulfilled by the following main contributions:

(i) **KRR-based Auto-Encoder for OCC: Reconstruction Framework-based:** We proposed a KRR-based Auto-Encoder for OCC (AEKOC) in Chapter 3, which is a reconstruction framework-based classifier. Auto-Encoder helped the classifier in obtaining a better representation of the data. AEKOC reconstructed the data at the output layer and computed reconstruction error. The belongingness of a sample to be an outlier or not is decided based on this reconstruction error and a threshold criterion. The proposed method is less computationally expensive compared to kernel-based traditional OCC methods (like OCSVM and SVDD) because it follows the non-iterative approach of learning. The performance of AEKOC is compared with 3 boundary framework-based (OCSVM, SVDD, and KOC) and one-reconstruction framework-based (KPCA) methods. We have experimented on 23 benchmark datasets to test the performance of the classifiers. AEKOC exhibited only slightly better performance compared to OCSVM, SVDD, and KOC in terms of η_m on these datasets. It yielded the best results for only 6 out of 23 datasets in terms of η_g. When we compared in terms of η_f, AEKOC yielded better value compared to KOC; however, it didn't yield better value compared to OCSVM and SVDD. Hence, the KRR-based one-class classifier is further improved by combining the concept of boundary and reconstruction framework in a single architecture.

(ii) **Multi-layer KRR for OCC with and without Graph-Embedding: Boundary and Reconstruction Framework-based:** We have explored representation learning for multi-layer architecture in Chapters 4 and 5. In Chapter 4, we combined the concept of boundary and reconstruction framework in a single architecture and developed a KRR-based multi-layer architecture for OCC (referred to as MKOC). This multi-layer architecture is constructed by stacking various Auto-Encoders sequentially. These stacked Auto-Encoders provided a

better representation of data, and the output of these stacked Auto-Encoders were passed to a boundary framework-based one-class classifier (KOC). The output of this architecture was experimented with two types of threshold criteria, namely; $\theta 1$ and $\theta 2$. Based on these two threshold criteria, we developed 2 variants of MKOC, namely MKOC_$\theta 1$ and MKOC_$\theta 2$. We compared the proposed multi-layer architecture-based method with various single hidden layer-based one-class classifiers on various performance criteria. Both MKOC_$\theta 1$ and MKOC_$\theta 2$ yielded better results compared to the existing methods. Despite its better performance, both methods collectively yielded the best results for 12 of 23 datasets only in terms of η_g. Therefore, for enhancing the performance of this multi-layer architecture, the optimization problem of MKOC has been extended to use structural information between samples in its formulation in Chapter 5. This structural information is generated by different types of Laplacian graphs and embedded into the existing multi-layer architecture. This method was referred to as GMKOC. It was experimented with 4 types of Laplacian graphs (LE, LLE, LDA, and CDA) and 2 types of threshold criteria ($\theta 1$ and $\theta 2$). In this way, we developed 8 variants of GMKOC. Overall, Graph-Embedding-based multi-layer methods significantly improved the performance of the multi-layer architecture and collectively yielded the best results for 17 out of 23 datasets in terms of η_g. Overall, the proposed multi-layer classifiers (MKOC and GMKOC) collectively yielded the best results for 20 out of 23 datasets in terms of η_g. Moreover, GMKOC with LLE and CDA have emerged as the best classifier in terms of η_m and η_f, respectively.

(iii) **Localized MKL for OCC: Boundary Framework-based:** MKL is required to capture different notions in the data. Unlike the multi-layer-based methods, the MKL-based method optimizes multiple kernels simultaneously in a single optimization function. In Chapter 6, we developed localized MKL-based one-class classifiers, which assign weights to each kernel based on locality present in the data. These weights are assigned with the help of a gating function in the op-

timization problem. This optimization problem is solved by two steps alternate optimization scheme. We have developed two localized MKL-based one-class classifiers for anomaly detection. One is developed by taking OCSVM as a base classifier and known as a localized MKL-based anomaly detector (LMKAD). Another one is developed by taking SVDD as a base classifier and is known as localized multiple kernel SVDD (LMSVDD). Further, both methods are experimented with three types of gating function (viz., Sigmoid (S), radial basis function (R), and softmax (So)) with two combinations of linear, polynomial and Gaussian kernels (viz., 'gpp' and 'gpl'). Overall, we generated 12 variants by using these gating function and multiple kernels. At this point in this thesis, we developed mainly 5 one-class classifiers with their 23 variants. Therefore, we compared all proposed and existing classifiers and observed that localized MKL-based methods collectively yielded the best results for 20 out of 23 datasets in terms of η_g. LMSVDD with Softmax gating function and 'gpp' kernel combination has emerged as the best classifier among all presented methods in this table in terms of both performance criteria (η_m and η_f).

(iv) **LUPI framework for KRR-based OCC: Boundary and Reconstruction Frameworks-based:** To handle privileged information, which is generally available with the training data in real-time, we incorporated KOC and AEKOC with the LUPI framework. In this way, we developed two one-class classifiers, namely KOC+ (boundary framework-based) and AEKOC+ (reconstruction framework-based). The proposed classifiers outperformed existing LUPI-based one-class classifiers (OCSVM+ and SVDD+). These methods are also computationally efficient because it was developed based on a non-iterative approach of learning.

(v) **Online learning for KRR-based OCC: Boundary and Reconstruction Frameworks-based:** The proposed methods discussed till now only work when the data is stationary, and the whole data is available for training before it starts. However, in the real-time scenario, data is available in the form of continuous streams. These continuous streams can be either stationary or non-stationary. To

handle this situation, we have enhanced boundary and reconstruction framework-based one-class classifiers (i.e., KOC and AEKOC) for online sequential learning, and referred to as OS-KOC and OS-AEKOC. For developing these classifiers, we introduced the concept of block inverse method and forgetting mechanism with KOC and AEKOC. The proposed methods have been tested for various types of drifts in a controlled environment. Proposed classifiers exhibited smooth adaption of drift in a non-stationary environment. They also exhibited better performance compared to existing online classifiers (incSVDD and OKPCA) for most of the datasets and yielded similar results for some datasets. Since computational cost is the primary concern in streaming data analysis, we have shown that the proposed classifiers are fast and computationally efficient as compared to existing online classifiers.

9.2 Future Research Directions

Despite significant progress in the topic of OCC, it can be explored in several interesting future directions as follows:

(i) **Multi-layer multi-kernel learning for OCC**: We have developed multi-layer and multi-kernel learning-based one-class classifier in Chapter 4 and Chapter 6, respectively. Here, one concept explores representation learning, and the other concept explores learning of different notions of similarity from various kernels. Both concepts can be combined under the same architecture for improving the performance. This combined architecture can be further developed for the Graph-Embedding approach to utilize the structural relationship between the samples.

(ii) **Online learning for multi-layer, multi-kernel, and LUPI-based methods:** We have developed various one-class classifiers for offline learning using various concepts, such as multi-layer, Graph-Embedded multi-layer, localized MKL, and LUPI. These classifiers can not handle non-stationary and stream-

ing data. Therefore, these classifiers can be developed for online learning for handling drift present in the data.

(iii) **KRR-based OCC methods for large scale learning using random projection, sketching and preconditioning:** All developed KRR-based one-class classifiers are only suitable for small-sized datasets. These methods are not suitable for large scale datasets due to the high time and memory requirements of KRR. In KRR, the dimensions of the matrix are the same as the number of samples in the dataset; therefore, direct methods are unrealistic for large-scale datasets [94]. Overcoming these limitations of KRR has motivated a variety of practical approaches, including gradient methods, as well as accelerated, stochastic, and preconditioned extensions, to improve time complexity [191, 192, 94, 193, 194, 195]. Random projections provide an approach to reduce memory requirements using methods like Nystrom approximation [196] and random features [197]. For a large class of problems, researchers exhibited that computation can be substantially reduced by combining Nystrom or random features methods, while the same optimal statistical accuracy of exact KRR is preserved [198, 91, 199, 200, 201]. In recent years, sketching and preconditioning for KRR are also developed for scaling up the KRR for large scale datasets [94, 195, 202]. By taking a cue from the approaches mentioned above, proposed KRR-based OCC methods in this thesis can be scaled-up for large-scale datasets.

Bibliography

[1] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In *Advances in Neural Information Processing Systems*, volume 12, pages 582–588, Denver, CO, 1999. MIT Press.

[2] L. M. Manevitz and M. Yousef. One-class svms for document classification. *Journal of Machine Learning Research*, 2(12):139–154, 2001.

[3] D. M. J. Tax and R. P. W. Duin. Support vector data description. *Machine Learning*, 54(1):45–66, 2004.

[4] H. Hoffmann. Kernel PCA for novelty detection. *Pattern Recognition*, 40(3):863–874, 2007. Software available at `http://www.heikohoffmann.de/kpca.html`.

[5] S. Luca, D. A. Clifton, and B. Vanrumste. One-class classification of point patterns of extremes. *Journal of Machine Learning Research*, 17(1):6581–6601, 2016.

[6] D. M. J. Tax. *One-class classification; Concept-learning in the absence of counter-examples*. PhD thesis, Technische Universiteit Delft, Netherlands, 2001.

[7] Q. Leng, H. Qi, J. Miao, W. Zhu, and G. Su. One-class classification with extreme learning machine. *Mathematical Problems in Engineering*, 2015:1–11, 2015.

[8] A. Iosifidis, V. Mygdalis, A. Tefas, and I. Pitas. One-class classification based on extreme learning and geometric class information. *Neural Processing Letters*, 45(2):577–592, 2017.

[9] W. Yan. One-class extreme learning machines for gas turbine combustor anomaly detection. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2909–2914, Vancouver, BC, Canada, 2016. IEEE.

[10] V. M. Janakiraman and D. Nielsen. Anomaly detection in aviation data using extreme learning machines. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1993–2000, Vancouver, BC, Canada, 2016. IEEE.

[11] S. Das, B. L. Matthews, A. N. Srivastava, and N. C. Oza. Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 47–56, Washington, DC, USA, 2010. ACM.

[12] M. Gönen and E. Alpaydin. Localized multiple kernel learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 352–359, Helsinki, Finland, 2008. ACM.

[13] V. Vapnik and A. Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5-6):544–557, 2009.

[14] V. Vapnik and R. Izmailov. Learning using privileged information: similarity control and knowledge transfer. *Journal of Machine Learning Research*, 16(2023-2049):2, 2015.

[15] C. O'Reilly, A. Gluhak, M. A. Imran, and S. Rajasegarar. Anomaly detection in wireless sensor networks in a non-stationary environment. *IEEE Communications Surveys & Tutorials*, 16(3):1413–1432, 2014.

[16] G. G. Sundarkumar and V. Ravi. A novel hybrid undersampling method for mining unbalanced datasets in banking and insurance. *Engineering Applications of Artificial Intelligence*, 37:368–377, 2015.

[17] V. Mygdalis, A. Iosifidis, A. Tefas, and I. Pitas. One class classification applied in facial image analysis. In *Proceedings of the IEEE International Conference on Image Processing*, pages 1644–1648, Phoenix, AZ, USA, 2016. IEEE.

[18] G. Cao, A. Iosifidis, and M. Gabbouj. Neural class-specific regression for face verification. *IET Biometrics*, 7(1):63–70, 2018.

[19] Y. Guerbai, Y. Chibani, and B. Hadjadji. The effective use of the one-class svm classifier for handwritten signature verification based on writer-independent parameters. *Pattern Recognition*, 48(1):103–113, 2015.

[20] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Proceedings of the 8th IEEE International Conference on Data Mining*, pages 502–511, Pisa, Italy, 2008. IEEE.

[21] G. Wu, M. R. Bouadjenek, and S. Sanner. One-class collaborative filtering with the queryable variational autoencoder. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, pages 921–924, Paris, France, 2019. ACM.

[22] W. Li, Q. Guo, and C. Elkan. A positive and unlabeled learning algorithm for one-class classification of remote-sensing data. *IEEE Transactions on Geoscience and Remote Sensing*, 49(2):717–725, 2010.

[23] C. Bergamini, L. S. Oliveira, A. L. Koerich, and R. Sabourin. Fusion of biometric systems using one-class classification. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1308–1313, Hong Kong, China, 2008. IEEE.

[24] A. Erçil and B. Buke. One class classification using implicit polynomial surface fitting. In *Object Recognition Supported by User Interaction for Service Robots*, volume 2, pages 152–155, Quebec City, Quebec, Canada, Canada, 2002. IEEE.

[25] F. J. Müller, B. M. Schuldt, R. Williams, D. Mason, G. Altun, E. P. Papapetrou, S. Danner, J. E. Goldmann, A. Herbst, N. O. Schmidt, et al. A bioinformatic assay for pluripotency in human cells. *Nature Methods*, 8(4):315, 2011.

[26] J. Mourão-Miranda, D. R. Hardoon, T. Hahn, A. F. Marquand, S. C. R. Williams, J. Shawe-Taylor, and M. Brammer. Patient classification as an outlier detection problem: an application of the one-class support vector machine. *Neuroimage*, 58(3):793–804, 2011.

[27] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15, 2009.

[28] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[29] M. M. Moya, M. W. Koch, and L. D. Hostetler. One-class classifier networks for target recognition applications. Technical report, Sandia National Labs., Albuquerque, United States, 1993.

[30] N. Japkowicz. *Concept-learning in the absence of counter-examples: An autoassociation-based approach to classification.* PhD thesis, Rutgers, The State University of New Jersey, 1999.

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

[32] D. R. Hardoon and L. M. Manevitz. fmri analysis via one-class machine learning techniques. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1604–1605, Edinburgh, Scotland, 2005. Morgan Kaufmann Publishers Inc.

[33] A. B. Gardner, A. M. Krieger, G. Vachtsevanos, and B. Litt. One-class novelty detection for seizure analysis from intracranial eeg. *Journal of Machine Learning Research*, 7(6):1025–1044, 2006.

[34] M. Wu and J. Ye. A small sphere and large margin approach for novelty detection using training data with outliers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):2088–2092, 2009.

[35] H. G. Bu, J. Wang, and X. B. Huang. Fabric defect detection based on multiple fractal features and support vector data description. *Engineering Applications of Artificial Intelligence*, 22(2):224–235, 2009.

[36] Y. S. Choi. Least squares one-class support vector machine. *Pattern Recognition Letters*, 30(13):1236–1240, 2009.

[37] W. Zhu and P. Zhong. A new one-class svm based on hidden information. *Knowledge-Based Systems*, 60:35–43, 2014.

[38] W. Zhang. Support vector data description using privileged information. *Electronics Letters*, 51(14):1075–1076, 2015.

[39] E. Burnaev and D. Smolyakov. One-class svm with privileged information and its application to malware detection. In *Proceedings of the IEEE International Conference on Data Mining Workshops*, pages 12–15, Barcelona, 2016. IEEE.

[40] D. M. J. Tax and P. Laskov. Online SVM learning: from classification to data description and back. In *Proceedings of the 13th IEEE Workshop on Neural Networks for Signal Processing*, pages 499–508, Toulouse, France, 2003. IEEE.

[41] F. Desobry, M. Davy, and C. Doncarli. An online kernel change detection algorithm. *IEEE Transactions on Signal Processing*, 53(8):2961–2974, 2005.

[42] A. Anaissi, N. L. D. Khoa, T. Rakotoarivelo, M. M. Alamdari, and Y. Wang. Adaptive online one-class support vector machines with applications in structural health monitoring. *ACM Transactions on Intelligent Systems and Technology*, 9(6):64, 2018.

[43] Y. Yang and Q. M. J. Wu. Multilayer extreme learning machine with subnetwork nodes for representation learning. *IEEE Transactions on Cybernetics*, 46(11):2570–2583, 2016.

[44] C. M. Wong, C. M. Vong, P. K. Wong, and J. Cao. Kernel-based multilayer extreme learning machines for representation learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(3):757–762, 2018.

[45] J. Tang, C. Deng, and G. B. Huang. Extreme learning machine for multilayer perceptron. *IEEE Transactions on Neural Networks and Learning Systems*, 27(4):809–821, 2016.

[46] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[47] S. Yan, D. Xu, B. Zhang, H. J. Zhang, Q. Yang, and S. Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1), 2007.

[48] M. Gönen and E. Alpaydın. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12(7):2211–2268, 2011.

[49] V. M. A. Souza, D. F. Silva, J. Gama, and G. E. A. P. A. Batista. Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In *Proceedings of SIAM International Conference on Data Mining*, pages 873–881, Vancouver, Canada, 2015. SIAM.

[50] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. More efficiency in multiple kernel learning. In *Proceedings of the 24th International Conference on Machine Learning*, pages 775–782, Corvalis, Oregon, USA, 2007. ACM.

[51] S. S. Khan and M. G. Madden. A survey of recent trends in one class classification. In *Proceedings of the 20th Irish Conference on Artificial Intelligence and Cognitive Science*, pages 188–197, Dublin, Ireland, 2009. Springer.

[52] J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London*, 209:415–446, 1909.

[53] M. E. Abbasnejad, D. Ramachandram, and R. Mandava. A survey of the state of the art in learning the kernels. *Knowledge and information systems*, 31(2):193–221, 2012.

[54] D. M. J. Tax. Ddtools, the data description toolbox for matlab, Jan 2018. version 2.1.3.

[55] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 1741–1745, Portland, OR, USA, 2003. IEEE.

[56] H. Yang, I. King, and M. R. Lyu. Multi-task learning for one-class classification. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8, Barcelona, Spain, 2010. IEEE.

[57] X. He, G. Mourot, D. Maquin, J. Ragot, P. Beauseroy, A. Smolarz, and E. Grall-Maës. Multi-task learning with one-class svm. *Neurocomputing*, 133:416–426, 2014.

[58] Y. Xue and P. Beauseroy. Multi-task learning for one-class svm with additional new features. In *Proceedings of the 23rd International Conference on Pattern Recognition*, pages 1571–1576, Cancún, Mexico, 2016. IEEE.

[59] N. M. Khan, R. Ksantini, I. S. Ahmad, and L. Guan. Covariance-guided one-class support vector machine. *Pattern Recognition*, 47(6):2165–2177, 2014.

[60] R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Proceedings of the 6th International Conference on Data Mining*, pages 488–498, Hong Kong, China, 2006. IEEE.

[61] P. Y. Hao. Fuzzy one-class support vector machines. *Fuzzy Sets and Systems*, 159(18):2317–2336, 2008.

[62] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the SIAM International Conference on Data Mining*, pages 25–36, Austin, Texas, USA, 2003. SIAM.

[63] K. A. Heller, K. M. Svore, A. D. Keromytis, and S. J. Stolfo. One class support vector machines for detecting anomalous windows registry accesses. In *Proceedings of the 3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security*, volume 9, pages 31–37, Florida, USA, 2003. IEEE.

[64] A. Rabaoui, H. Kadri, and N. Ellouze. New approaches based on one-class svms for impulsive sounds recognition tasks. In *IEEE Workshop on Machine Learning for Signal Processing*, pages 285–290, Cancun, Mexico, 2008. IEEE.

[65] L. Zhuang and H. Dai. Parameter optimization of kernel-based one-class classifier on imbalance learning. *Journal of Computers*, 1(7):32–40, 2006.

[66] L. A. Clifton, H. Yin, and Y. Zhang. Support vector machine in novelty detection for multi-channel combustion data. In *International Symposium on Neural Networks*, pages 836–843, Chengdu, China, 2006. Springer.

[67] L. A. Clifton, H. Yin, D. A. Clifton, and Y. Zhang. Combined support vector novelty detection for multi-channel combustion data. In *IEEE International Conference on Networking, Sensing and Control*, pages 495–500, London, UK, 2007. IEEE.

[68] P. Hayton, S. Utete, D. King, S. King, P. Anuzis, and L. Tarassenko. Static and dynamic novelty detection methods for jet engine health monitoring. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):493–514, 2006.

[69] K. Lee, D. W. Kim, D. Lee, and K. H. Lee. Improving support vector data description using local density degree. *Pattern Recognition*, 38(10):1768–1771, 2005.

[70] K. Y. Lee, D. W. Kim, K. H. Lee, and D. Lee. Density-induced support vector data description. *IEEE Transactions on Neural Networks*, 18(1):284–289, 2007.

[71] D. Le, T.and Tran, W. Ma, and D. Sharma. An optimal sphere and two large margins approach for novelty detection. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–6, Barcelona, Spain, 2010. IEEE.

[72] Y. Xiao, B. Liu, L. Cao, X. Wu, C. Zhang, Z. Hao, F. Yang, and J. Cao. Multi-sphere support vector data description for outliers detection on multi-distribution data. In *Proceedings of the IEEE International Conference on Data Mining Workshops*, pages 82–87, Miami, Florida, USA, 2009. IEEE.

[73] Y. H. Liu, Y. C. Liu, and Y. J. Chen. Fast support vector data descriptions for novelty detection. *IEEE Transactions on Neural Networks*, 21(8):1296–1313, 2010.

[74] X. Peng and D. Xu. Efficient support vector data descriptions for novelty detection. *Neural Computing and Applications*, 21(8):2023–2032, 2012.

[75] J. Muñoz-Marí, L. Bruzzone, and G. Camps-Valls. A support vector domain description approach to supervised classification of remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 45(8):2683–2692, 2007.

[76] Y. H. Liu, Y. C. Liu, and Y. Z. Chen. High-speed inline defect detection for tft-lcd array process using a novel support vector data description. *Expert Systems with Applications*, 38(5):6222–6231, 2011.

[77] Z. Ge, F. Gao, and Z. Song. Batch process monitoring based on support vector data description method. *Journal of Process Control*, 21(6):949–959, 2011.

[78] Y. Zhao, S. Wang, and F. Xiao. Pattern recognition-based chillers fault detection method using support vector data description (svdd). *Applied Energy*, 112:1041–1048, 2013.

[79] S. Wang, J. Yu, E. Lapira, and J. Lee. A modified support vector data description based novelty detection approach for machinery components. *Applied Soft Computing*, 13(2):1193–1205, 2013.

[80] G. Chen, X. Zhang, Z. J. Wang, and F. Li. Robust support vector data description for outlier detection with noise or uncertain data. *Knowledge-Based Systems*, 90:129–137, 2015.

[81] J. A. K. Suykens, T. V. Gestel, and J. D. Brabanter. *Least squares support vector machines*. World Scientific, 2002.

[82] T. V. Gestel, J. A. K. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learning*, 54(1):5–32, 2004.

[83] C. Saunders, A. Gammerman, and V. Vovk. Ridge regression learning algorithm in dual variables. In *Proceedings of the 15th International Conference on Machine Learning*, ICML '98, pages 515–521, Madison, Wisconsin, USA, 1998. Morgan Kaufmann Publishers Inc.

[84] G. B. Huang, Q. Y. Zhu, and C. K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, 2006.

[85] G. B. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, 2012.

[86] G. B. Huang. An insight into extreme learning machines: random neurons, random features and kernels. *Cognitive Computation*, 6(3):376–390, 2014.

[87] N. Cristianini and J. Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[88] C. Cortes and M. Mohri. On transductive regression. In *Advances in Neural Information Processing Systems*, pages 305–312, Vancouver, Canada, 2007. MIT Press.

[89] M. Solnon, S. Arlot, and F. Bach. Multi-task regression using minimal penalties. *Journal of Machine Learning Research*, 13(9):2773–2812, 2012.

[90] Y. Zhang, J. Duchi, and M. Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *Journal of Machine Learning Research*, 16(1):3299–3340, 2015.

[91] A. Alaoui and M. W. Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems*, pages 775–783, Montreal, Quebec, Canada, 2015. MIT Press.

[92] L. Zhang and P. N. Suganthan. Benchmarking ensemble classifiers with novel co-trained kernel ridge regression and random vector functional link ensembles [research frontier]. *IEEE Computational Intelligence Magazine*, 12(4):61–72, 2017.

[93] S. Si, C. J. Hsieh, and I. S. Dhillon. Memory efficient kernel approximation. *Journal of Machine Learning Research*, 18(1):682–713, 2017.

[94] H. Avron, K. L. Clarkson, and D. P. Woodruff. Faster kernel ridge regression using sketching and preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1116–1138, 2017.

[95] H. Avron, M. Kapralov, C. Musco, C. Musco, A. Velingker, and A. Zandieh. Random fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 253–262, Sydney, NSW, Australia, 2017. PMLR.

[96] S. An, W. Liu, and S. Venkatesh. Face recognition using kernel ridge regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, Minneapolis, Minnesota, USA, 2007. IEEE.

[97] B. G. V. Kumar and R. Aravind. Face hallucination using olpp and kernel ridge regression. In *Proceedings of the IEEE International Conference on Image Processing*, pages 353–356, San Diego, California, USA, 2008. IEEE.

[98] C. Chu, Y. Ni, G. Tan, C. J. Saunders, and J. Ashburner. Kernel regression for fmri pattern prediction. *NeuroImage*, 56(2):662–673, 2011.

[99] F. Douak, F. Melgani, and N. Benoudjit. Kernel ridge regression with active learning for wind speed prediction. *Applied energy*, 103:328–340, 2013.

[100] J. Chen, L. Wu, K. Audhkhasi, B. Kingsbury, and B. Ramabhadrari. Efficient one-vs-one kernel ridge regression for speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2454–2458, Shanghai, China, 2016. IEEE.

[101] S. Wang, E. Zhu, and J. Yin. Video anomaly detection based on ulgp-of descriptor and one-class elm. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2630–2637, Vancouver, BC, Canada, 2016. IEEE.

[102] V. Mygdalis, A. Iosifidis, A. Tefas, and I. Pitas. Laplacian one class extreme learning machines for human action recognition. In *Proceedings of the 18th IEEE International Workshop on Multimedia Signal Processing*, pages 1–5, Montreal, QC, Canada, 2016. IEEE.

[103] A. Argyriou, C. A. Micchelli, and M. Pontil. When is there a representer theorem? vector versus matrix regularizers. *Journal of Machine Learning Research*, 10:2507–2529, 2009.

[104] D. Lituiev. Autoencoders: a bibliographic survey. `https://dslituiev.github.io/deeplearning/2017/04/19/autoencoders-bibliographic-survey.html`, 2019. [Online; accessed 30-April-2019].

[105] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in Neural Information Processing Systems*, pages 3–10, Denver, Colorado, USA, 1994. Morgan-Kaufmann.

[106] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[107] D. Erhan, Y. Bengio, A. Courville, P. A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(2):625–660, 2010.

[108] F. Zhuang, D. Luo, X. Jin, H. Xiong, P. Luo, and Q. He. Representation learning via semi-supervised autoencoder for multi-task learning. In *Proceedings of the IEEE International Conference on Data Mining*, pages 1141–1146, Atlantic City, NJ, USA, 2015. IEEE.

[109] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin. Variational autoencoder for deep learning of images, labels and captions. In *Advances in Neural Information Processing Systems*, pages 2352–2360, Barcelona, Spain, 2016. Curran Associates Inc.

[110] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, Helsinki, Finland, 2008. ACM.

[111] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang. Deep structured energy based models for anomaly detection. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, volume 48 of *ICML'16*, pages 1100–1109, New York, NY, USA, 2016. PMLR.

[112] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the World Wide Web Conference*, WWW '18, pages 187–196, Lyon, France, 2018. IW3C2.

[113] C. Zhou and R. C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 665–674, Halifax, NS, Canada, 2017. ACM.

[114] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[115] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 2, Anchorage, Alaska, USA, 2008. IEEE.

[116] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, pages 1753–1760, Vancouver, Canada, 2009. Curran Associates Inc.

[117] F. Zhuang, X. Cheng, P. Luo, S. J. Pan, and Q. He. Supervised representation learning: Transfer learning with deep autoencoders. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 4119–4125, Buenos Aires, Argentina, 2015. AAAI Press.

[118] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[119] T. Gartner. *Kernels for structured data*, volume 72 of *Series in Machine Perception and Artificial Intelligence*. World Scientific, 2008.

[120] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics*, 12(1):6–26, 1999.

[121] H. Cai, V. W. Zheng, and K. C. C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[122] X. Wei, L. Xu, B. Cao, and P. S. Yu. Cross view link prediction by learning noise-resilient representation consensus. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1611–1619, Perth, Australia, 2017. IW3C2.

[123] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 203–209, San Francisco, California, USA, 2017. AAAI Press.

[124] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao. Scalable graph embedding for asymmetric proximity. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2942–2948, San Francisco, California, USA, 2017. AAAI Press.

[125] F. Nie, W. Zhu, and X. Li. Unsupervised large graph embedding. In *Proceedings of the 31st AAAI conference on Artificial Intelligence*, pages 2422–2428, San Francisco, California, USA, 2017. AAAI Press.

[126] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, pages 585–591, Vancouver, Canada, 2002. MIT Press.

[127] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[128] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[129] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[130] M. Sugiyama. Dimensionality reduction of multimodal labeled data by local fisher discriminant analysis. *Journal of Machine Learning Research*, 8(5):1027–1061, 2007.

[131] X. W. Chen and T. Huang. Facial expression recognition: a clustering-based approach. *Pattern Recognition Letters*, 24(9-10):1295–1302, 2003.

[132] V. D. Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems*, pages 721–728, Vancouver, Canada, 2003. MIT Press.

[133] F. R. Bach, G. R. G. Lanckriet, and M. I. Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the 21st International Conference on Machine Learning*, page 6, Banff, Alberta, Canada, 2004. ACM.

[134] Q. Fan, D. Gao, and Z. Wang. Multiple empirical kernel learning with locality preserving constraint. *Knowledge-Based Systems*, 105:107–118, 2016.

[135] T. Wang, D. Zhao, and Y. Feng. Two-stage multiple kernel learning with multiclass kernel polarization. *Knowledge-Based Systems*, 48:10–16, 2013.

[136] M. Goenen and E. Alpaydin. Localized multiple kernel machines for image recognition. In *Neural Information Processing Systems—Workshop on Understanding Multiple Kernel Learning Method*, Vancouver, Canada, 2009. MIT Press.

[137] M. Gonen and E. Alpaydin. Localized multiple kernel regression. In *20th International Conference on Pattern Recognition*, pages 1425–1428, Istanbul, Turkey, 2010. IEEE.

[138] L. Zhang and X. Hu. Locally adaptive multiple kernel clustering. *Neurocomputing*, 137:192–197, 2014.

[139] Y. Han, K. Yang, and G. Liu. $l\_\{p\}$ norm localized multiple kernel learning via semi-definite programming. *IEEE Signal Processing Letters*, 19(10):688–691, 2012.

[140] Y. Han, K. Yang, Y. Ma, and G. Liu. Localized multiple kernel learning via sample-wise alternating optimization. *IEEE Transactions on Cybernetics*, 44(1):137–148, 2014.

[141] Y. Lei, A. Binder, U. Dogan, and M. Kloft. Localized multiple kernel learning—a convex approach. In *Proceedings of the 8th Asian Conference on Machine Learning*, volume 63 of *Proceedings of Machine Learning Research*, pages 81–96, Hamilton, New Zealand, Nov 2016. PMLR.

[142] Y. Han, K. Yang, Y. Yang, and Y. Ma. Localized multiple kernel learning with dynamical clustering and matrix regularization. *IEEE Transactions on Neural Networks and Learning Systems*, 29(2):486–499, 2018.

[143] E. Cunha and B. Martins. Using one-class classifiers and multiple kernel learning for defining imprecise geographic regions. *International Journal of Geographical Information Science*, 28(11):2220–2241, 2014.

[144] R. Zurita-Milla, K. I. Balasubramanian, E. Izquierdo-Verdiguier, and R. A. de By. The combination of multiple kernel learning and one-class classifier in classifying smallholder cotton fields. In *The 5th International Symposium on Recent Advances in Quantitative Remote Sensing: RAQRS'V*, pages 1–1, Torrent, Valencia, Spain, 2017. University of Twente.

[145] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[146] M. GöNen and E. AlpaydıN. Localized algorithms for multiple kernel learning. *Pattern Recognition*, 46(3):795–807, 2013.

[147] V. Vapnik, A. Vashist, and N. Pavlovitch. Learning using hidden information: Master class learning. *NATO Science for Peace and Security Series, D: Information and Communication Security*, 19:3–14, 2008.

[148] J. Feyereisl and U. Aickelin. Privileged information for data clustering. *Information Sciences*, 194:4–23, 2012.

[149] X. Xu, W. Li, and D. Xu. Distance metric learning using privileged information for face verification and person re-identification. *IEEE Transactions on Neural Networks and Learning Systems*, 26(12):3150–3162, 2015.

[150] S. Motiian, M. Piccirilli, Donald A. Adjeroh, and G. Doretto. Information bottleneck learning using privileged information for visual recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1496–1505, Las Vegas, NV, USA, 2016. IEEE.

[151] S. Wang, S. Chen, T. Chen, and X. Shi. Learning with privileged information for multi-label classification. *Pattern Recognition*, 81:60–70, 2018.

[152] J. Lambert, O. Sener, and S. Savarese. Deep learning under privileged information using heteroscedastic dropout. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8886–8895, Salt Lake City, UT, USA, 2018. IEEE.

[153] M. Chevalier, N. Thome, G. Hénaff, and M. Cord. Classifying low-resolution images by integrating privileged information in deep cnns. *Pattern Recognition Letters*, 116:29–35, 2018.

[154] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(3):10, 2010.

[155] H. B. McMahan. A survey of algorithms and analysis for adaptive online learning. *Journal of Machine Learning Research*, 18(1):3117–3166, 2017.

[156] Y. Zhang, N. Meratnia, and P. J. M. Havinga. Ensuring high sensor data quality through use of online outlier detection techniques. *International Journal of Sensor Networks*, 7(3):141–151, 2010.

[157] Y. Zhang, N. Meratnia, and P. J. M. Havinga. Distributed online outlier detection in wireless sensor networks using ellipsoidal support vector machine. *Ad Hoc Networks*, 11(3):1062–1074, 2013.

[158] S. Rajasegarar, C. Leckie, J. C. Bezdek, and M. Palaniswami. Centered hyperspherical and hyperellipsoidal one-class support vector machines for anomaly detection in sensor networks. *IEEE Transactions on Information Forensics and Security*, 5(3):518–533, 2010.

[159] G. Ratsch, S. Mika, B. Scholkopf, and K. R. Muller. Constructing boosting algorithms from svms: an application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1184–1199, 2002.

[160] V. Chatzigiannakis and S. Papavassiliou. Diagnosing anomalies and identifying faulty nodes in sensor networks. *IEEE Sensors Journal*, 7(5):637–645, 2007.

[161] M. Xie, J. Hu, and B. Tian. Histogram-based online anomaly detection in hierarchical wireless sensor networks. In *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 751–759, Liverpool, United Kingdom, 2012. IEEE.

[162] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 187–198, Seoul, Korea, 2006. VLDB Endowment.

[163] Y. Zhang, N. Meratnia, and P. Havinga. Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks. In *Proceedings of the IEEE International Conference on Advanced Information Networking and Applications Workshops*, pages 990–995, Bradford, United Kingdom, 2009. IEEE.

[164] M. R. Watson, N. U. H. Shirazi, A. K. Marnerides, A. Mauthe, and D. Hutchison. Malware detection in cloud computing infrastructures. *IEEE Transactions on Dependable and Secure Computing*, 13(2):192–205, 2016.

[165] P. Feng, W. Wang, S. Dlay, S. M. Naqvi, and J. Chambers. Social force model-based mcmc-ocsvm particle phd filter for multiple human tracking. *IEEE Transactions on Multimedia*, 19(4):725–739, 2017.

[166] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1):1–30, 2006.

[167] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems. *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[168] T. Pahikkala and A. Airola. Rlscore: regularized least-squares learners. *Journal of Machine Learning Research*, 17(1):7803–7807, 2016.

[169] TU Delft one-class dataset repository, Last Accessed by 21 October 2019.

[170] D. Dua and C. Graff. UCI machine learning repository, 2017.

[171] C. C. Chang and C. J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:1–27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[172] L. K. Saul and S. T. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4(6):119–155, 2003.

[173] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[174] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[175] M. J. Box, D. Davies, and W. H. Swann. *Non-linear Optimization Techniques*. Mathematical and Statistical Techniques for Industry, Monograph no. 5. Oliver and Boyd, Edinburgh, 1969.

[176] W. C. Chang, C. P. Lee, and C. J. Lin. A revisit to support vector data description. *Department of Computer Science, National Taiwan University, Taipei, Taiwan*, 2013.

[177] M. Zhu. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2:30, 2004.

[178] C. D Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval, chapter 13, 2008.

[179] W. Li, D. Dai, M. Tan, D. Xu, and L. V. Gool. Fast algorithms for linear and kernel SVM+. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2258–2266, Las Vegas, NV, USA, 2016. IEEE.

[180] S. C. Chan, H. C. Wu, and K. M. Tsui. Robust recursive Eigendecomposition and subspace-based algorithms with application to fault detection in wireless sensor networks. *IEEE Transactions on Instrumentation and Measurement*, 61(6):1703–1718, 2012.

[181] C. O'Reilly, A. Gluhak, and M. A. Imran. Adaptive anomaly detection with kernel eigenspace splitting and merging. *IEEE Transactions on Knowledge and Data Engineering*, 27(1):3–16, 2015.

[182] C. Lemaréchal. Lagrangian relaxation. In *Computational Combinatorial Optimization*, pages 112–156. Springer, 2001.

[183] P. E. Gill, W. Murray, and M. H Wright. *Practical optimization*. Academic Press, 1981.

[184] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[185] N. J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[186] K. B. Dyer, R. Capo, and R. Polikar. Compose: A semisupervised learning framework for initially labeled nonstationary streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):12–26, 2014.

[187] G. Ditzler and R. Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, 2013.

[188] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[189] K. Killourhy and R. Maxion. Why did my detector do that?! In *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, pages 256–276, Berlin, Heidelberg, 2010. Springer.

[190] M. D. Petkovic and P. S. Stanimirovic. Block recursive computation of generalized inverses. *Electronic Journal of Linear Algebra*, 26(1):26, 2013.

[191] L. L. Gerfo, L. Rosasco, F. Odone, E. D. Vito, and A. Verri. Spectral algorithms for supervised learning. *Neural Computation*, 20(7):1873–1897, 2008.

[192] G. E. Fasshauer and M. J. McCourt. Stable evaluation of gaussian radial basis function interpolants. *SIAM Journal on Scientific Computing*, 34(2):A737–A762, 2012.

[193] A. Gonen, F. Orabona, and S. Shalev-Shwartz. Solving ridge regression using sketched preconditioned SVRG. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1397–1405, New York City, NY, USA, 2016. PMLR.

[194] S. Ma and M. Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. In *Advances in Neural Information Processing Systems*, pages 3778–3787, Long Beach, CA, USA, 2017. Curran Associates Inc.

[195] A. Rudi, L. Carratino, and L. Rosasco. Falkon: An optimal large scale kernel method. In *Advances in Neural Information Processing Systems*, pages 3888–3898, Long Beach, CA, USA, 2017. Curran Associates Inc.

[196] C. K. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, pages 682–688, Denver, CO, USA, 2001. MIT Press.

[197] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184, Vancouver, Canada, 2008. Curran Associates Inc.

[198] F. Bach. Sharp analysis of low-rank kernel matrix approximations. In *Proceedings of the 26th Annual Conference on Learning Theory*, pages 185–209, Princeton University, NJ, USA, 2013. PMLR.

[199] A. Rudi, R. Camoriano, and L. Rosasco. Less is more: Nyström computational regularization. In *Advances in Neural Information Processing Systems*, pages 1657–1665, Montreal, Quebec, Canada, 2015. Curran Associates Inc.

[200] A. Rudi and L. Rosasco. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems*, pages 3215–3225, Long Beach, CA, USA, 2017. Curran Associates Inc.

[201] F. Bach. On the equivalence between kernel quadrature rules and random feature expansions. *Journal of Machine Learning Research*, 18(1):714–751, 2017.

[202] A. Chowdhury, J. Yang, and P. Drineas. An iterative, sketching-based framework for ridge regression. In *Proceedings of the 35th International Conference on Machine Learning*, pages 988–997, Stockholmsmässan, Stockholm, Sweden, 2018. PMLR.