ROBUST KERNEL ADAPTIVE LEARNING FOR ONLINE SUPERVISED SYSTEMS

By
ROSCHA POKHAREL

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY UNIVERSITY OF FLORIDA 2014
To my parents
I would like to acknowledge and thank Dr. Jose C. Principe for his continuous guidance and encouragement through several discussions and exchange of ideas at every step of my Ph.D. career. I thank Dr. John G. Harris for introducing me to University of Florida and to CNEL at the first place and for guiding me through this by being in my advisory committee. I also thank my advisory committee members, Dr. Anand Rangarajan for helping me think critically of my research from machine learning perspective and Dr. Yunmei Chen for being kind enough to be on the committee. I thank Dr. Sohan Seth for being a mentor, an enthusiastic collaborator, and a brother.

These four years would not have been as much fun without my dear friends and colleagues in CNEL. I thank Rakesh Chalasani, Austin Brockmeier, and Goktug Cinar for their friendship and several fruitful discussions, Badong Chen, Pinging Zhu, Evan Kriminger, Luis Sanchez Giraldo, Songlin Zhao, Bilal Fadlallah, Jihye Bae for their helpful insight on my research, In Jun Park, Matthew Emigh, Gavin Philips, Zheng Cao, Miguel deLima Teixeira, Gabreil Nalathambi, Guibiao Xu, Mehrnaz Hazrati, Eder Santana, Catia Silva for sharing their positive spirit and curiosity. I also thank my seniors Alexander Singh Alvarado, Erion Hasanbelliu, Vaibhab Gharg, Jie Xu, Ravi Shekhar, Manu Rastogi, Kittipat 'Bot' Kampa, Stefan Craciun and Jeremy Anderson for their valuable suggestions during my initial days in CNEL.

A special thanks to Shweta Sharma, Subodh Sharma, Shirish Ranabhat, Manju Karki, Nilesh Timilsena and Shraddha Timilsena for being my family away from home and Achala Acharya for being a caring roommate and a great friend.

Finally, I thank my father Dr.Champak Pokharel and my mother Gayatri Pokharel for their unconditional love and support every step of the way, my brother Puskal Pokharel, sister Puspa Pokharel and sister-in-law Sukriti Subedi for their never ending care and guidance, and my dearest husband Ritesh Paudel for being ever so patient, supportive and my greatest source of joy.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>8</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>9</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>11</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>13</td>
</tr>
<tr>
<td>2 THEORETICAL BACKGROUND</td>
<td>17</td>
</tr>
<tr>
<td>2.1 Statistical Learning Theory</td>
<td>17</td>
</tr>
<tr>
<td>2.2 Kernel Methods and Reproducing Kernel Hilbert Space</td>
<td>17</td>
</tr>
<tr>
<td>2.2.1 Reproducing Property</td>
<td>18</td>
</tr>
<tr>
<td>2.2.2 Universal Approximation Property</td>
<td>18</td>
</tr>
<tr>
<td>2.2.3 Sum of Reproducing Kernels</td>
<td>19</td>
</tr>
<tr>
<td>2.3 Kernel Adaptive Filtering</td>
<td>19</td>
</tr>
<tr>
<td>2.4 Kernel Least Mean Square</td>
<td>20</td>
</tr>
<tr>
<td>2.5 Correntropy</td>
<td>22</td>
</tr>
<tr>
<td>3 ROBUST FILTERING USING MIXTURE KERNEL LEAST MEAN SQUARE</td>
<td>24</td>
</tr>
<tr>
<td>3.1 Background</td>
<td>24</td>
</tr>
<tr>
<td>3.2 Existing Multi-Kernel Adaptive Filtering Algorithms</td>
<td>25</td>
</tr>
<tr>
<td>3.2.1 Multikernel Least Mean Square</td>
<td>26</td>
</tr>
<tr>
<td>3.2.2 Multikernel Normalized Least Mean Square</td>
<td>27</td>
</tr>
<tr>
<td>3.3 Existing Multiple Kernel Learning (MKL) Techniques</td>
<td>27</td>
</tr>
<tr>
<td>3.4 MxKLMS: Multi-Kernel Adaptive Filtering</td>
<td>29</td>
</tr>
<tr>
<td>3.5 Convergence Analysis of MxKLMS</td>
<td>33</td>
</tr>
<tr>
<td>3.5.1 Energy Conservation Relation</td>
<td>34</td>
</tr>
<tr>
<td>3.5.2 Mean Square Convergence Criteria</td>
<td>36</td>
</tr>
<tr>
<td>3.5.3 Steady-State Mean Square Performance</td>
<td>38</td>
</tr>
<tr>
<td>3.6 Experiments and Results</td>
<td>41</td>
</tr>
<tr>
<td>3.6.1 Nonlinear Channel Equalization</td>
<td>41</td>
</tr>
<tr>
<td>3.6.2 Non-Stationary Online Signal Prediction</td>
<td>42</td>
</tr>
<tr>
<td>3.6.3 Model Selection</td>
<td>46</td>
</tr>
<tr>
<td>3.6.4 Online Prediction of Santa Fe Laser Data</td>
<td>48</td>
</tr>
<tr>
<td>3.6.5 Online Prediction of Lorenz Chaotic Time Series</td>
<td>51</td>
</tr>
<tr>
<td>3.6.6 Long-Term Time Series Prediction</td>
<td>54</td>
</tr>
<tr>
<td>3.7 Summary</td>
<td>57</td>
</tr>
</tbody>
</table>
4 QUANTIZED MIXTURE KERNEL LEAST MEAN SQUARE

4.1 Background

4.2 Quantized KLMS: Overview

4.3 Quantized MxKLMS

4.4 Experiments and Results
   4.4.1 Non-Stationary Signal Prediction
   4.4.2 Short-term Prediction of Lorenz Chaotic Time Series

4.5 Summary

5 MIXTURE KERNEL MAXIMUM CORRENTROPY CRITERIA

5.1 Background
   5.1.1 Maximum Correntropy Criteria
   5.1.2 Generalized Convexity of Correntropy Cost Function
   5.1.3 Kernel MCC

5.2 Mixture Kernel Maximum Correntropy Criteria (MxKMC)

5.3 Experiments and Results
   5.3.1 Synthetic Data
   5.3.2 Noisy Short-term Prediction of Lorenz Chaotic Time Series

5.4 Summary

6 ROBUST CLASSIFICATION USING KERNEL ADAPTIVE CLASSIFIER

6.1 Background
   6.1.1 C-loss Function
   6.1.2 Generalized Convexity of C-loss

6.2 Kernel Adaptive Classifier Algorithm

6.3 Property of KAC Using C-loss: Self-Regularization

6.4 Experiments and Results
   6.4.1 Synthetic Data-1
   6.4.2 Synthetic Data-2
   6.4.3 Pima Indian Diabetes Dataset
   6.4.4 Wisconsin Breast Cancer Dataset
   6.4.5 Blood Transfusion Service Center Dataset
   6.4.6 Comparison with SVM and NN classifiers

6.5 Summary

7 NON-STATIONARY ONLINE CLASSIFICATION USING MIXTURE KERNEL ADAPTIVE CLASSIFIER

7.1 Mixture Kernel Adaptive Classifier for Non-Stationary Classification

7.2 Experiments

7.3 Summary
8 CONCLUSION ........................................................................... 118
  8.1 Summary ........................................................................... 118
  8.2 Avenues for Future Work ...................................................... 120
REFERENCES ........................................................................... 122
BIOGRAPHICAL SKETCH .......................................................... 128
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Performance comparison of MxKLMS with KLMS and LMS in nonlinear channel</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>equalization</td>
<td></td>
</tr>
<tr>
<td>3-2</td>
<td>Result of model selection using MxKLMS</td>
<td>46</td>
</tr>
<tr>
<td>3-3</td>
<td>Result of prediction of Santa Fe laser data</td>
<td>49</td>
</tr>
<tr>
<td>3-4</td>
<td>Result of prediction of Lorenz chaotic signal</td>
<td>54</td>
</tr>
<tr>
<td>3-5</td>
<td>Result of prediction of real world non-stationary wind signal</td>
<td>57</td>
</tr>
<tr>
<td>4-1</td>
<td>Comparison of QMxKLMS-Eu, QMxKLMS-KEu, MxKLMS and KLMS on prediction</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>of synthetic non-stationary signal</td>
<td></td>
</tr>
<tr>
<td>4-2</td>
<td>Comparison of QMxKLMS-Eu, QMxKLMS-KEu, MxKLMS, MKNLMS-CS, MKLMS,</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>and QKLMS on prediction of Lorenz chaotic time series</td>
<td></td>
</tr>
<tr>
<td>5-1</td>
<td>Performance comparison for noisy prediction on synthetic data</td>
<td>86</td>
</tr>
<tr>
<td>5-2</td>
<td>Performance comparison for noisy prediction of Lorenz chaotic time series</td>
<td>88</td>
</tr>
<tr>
<td>6-1</td>
<td>Performance comparison of KAC, neural network and SVM on benchmark datasets</td>
<td>106</td>
</tr>
<tr>
<td>6-2</td>
<td>Classification performance of KAC with C-loss and square loss on benchmark</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>datasets</td>
<td></td>
</tr>
<tr>
<td>7-1</td>
<td>Mean percentage accuracy of classification of synthetic Gaussian dataset at</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>different time steps.</td>
<td></td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1-1</td>
<td>General overview of kernel adaptive learning for online supervised learning system.</td>
<td>14</td>
</tr>
<tr>
<td>2-1</td>
<td>Block Diagram of Kernel Adaptive Filtering</td>
<td>20</td>
</tr>
<tr>
<td>2-2</td>
<td>2D contour plot of correntropy</td>
<td>22</td>
</tr>
<tr>
<td>3-1</td>
<td>MxKLMS filter structure</td>
<td>33</td>
</tr>
<tr>
<td>3-2</td>
<td>Result of nonlinear channel equalization</td>
<td>43</td>
</tr>
<tr>
<td>3-3</td>
<td>Result of synthetic non-stationary signal prediction</td>
<td>44</td>
</tr>
<tr>
<td>3-4</td>
<td>Result of model selection using MxKLMS</td>
<td>47</td>
</tr>
<tr>
<td>3-5</td>
<td>Result of prediction of Santa Fe laser data</td>
<td>50</td>
</tr>
<tr>
<td>3-6</td>
<td>Result of prediction of Lorenz chaotic time series</td>
<td>52</td>
</tr>
<tr>
<td>3-7</td>
<td>Result of prediction of real world non-stationary wind-data</td>
<td>56</td>
</tr>
<tr>
<td>4-1</td>
<td>Effect on quantization threshold for QMxKLMS-Eu and QMxKLMS-KEu on synthetic non-stationary data</td>
<td>66</td>
</tr>
<tr>
<td>4-2</td>
<td>Performance comparison of QMxKLMS-Eu, QMxKLMS-KEu and QKLMS on prediction of synthetic non-stationary signal</td>
<td>68</td>
</tr>
<tr>
<td>4-3</td>
<td>QMxKLMS Kernel weight evolution across samples for synthetic non-stationary signal</td>
<td>70</td>
</tr>
<tr>
<td>4-4</td>
<td>Effect on quantization threshold for QMxKLMS-Eu and QMxKLMS-KEu on Lorenz chaotic time series</td>
<td>72</td>
</tr>
<tr>
<td>4-5</td>
<td>Performance comparison of QMxKLMS-Eu, QMxKLMS-KEu, MKNLMS-CS, and MKNLMS on prediction of Lorenz time series for small network size</td>
<td>73</td>
</tr>
<tr>
<td>4-6</td>
<td>Performance comparison of QMxKLMS-Eu, QMxKLMS-KEu, MKNLMS-CS, and MKNLMS on prediction of Lorenz chaotic series for minimum test NMSE</td>
<td>75</td>
</tr>
<tr>
<td>5-1</td>
<td>Frequency doubling</td>
<td>85</td>
</tr>
<tr>
<td>5-2</td>
<td>Comparison of test NMSE across training samples for synthetic data prediction</td>
<td>86</td>
</tr>
<tr>
<td>5-3</td>
<td>Kernel weights across samples for synthetic data</td>
<td>87</td>
</tr>
<tr>
<td>5-4</td>
<td>Comparison of test NMSE across training samples for Lorenz chaotic data.</td>
<td>88</td>
</tr>
<tr>
<td>5-5</td>
<td>Kernel weights across samples for Lorenz chaotic data</td>
<td>90</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6-1</td>
<td>Properties of C-loss</td>
<td>92</td>
</tr>
<tr>
<td>6-2</td>
<td>The empirical risk function in a 2D space of errors $e = y - f(x)$ for c-loss</td>
<td>93</td>
</tr>
<tr>
<td>6-3</td>
<td>Decision boundaries obtained for Synthetic Data-1 using KAC with c-loss ($\sigma_c = 0.5$), KAC with square loss, and C-SVM</td>
<td>98</td>
</tr>
<tr>
<td>6-4</td>
<td>Decision boundaries obtained for Synthetic Data-2 using KAC with c-loss ($\sigma_c = 0.5$), KAC with square loss, and C-SVM</td>
<td>100</td>
</tr>
<tr>
<td>6-5</td>
<td>Performance of KAC with C-loss ($\sigma_c = 0.5$) and sq-loss on Pima Indian diabetes dataset across epochs for different values of $\sigma$</td>
<td>101</td>
</tr>
<tr>
<td>6-6</td>
<td>Performance of KAC with C-loss ($\sigma_c = 0.5$) and sq-loss on Wisconsin breast cancer dataset across epochs for different values of $\sigma$</td>
<td>103</td>
</tr>
<tr>
<td>6-7</td>
<td>Performance of KAC with C-loss ($\sigma_c = 0.5$) and sq-loss on blood transfusion dataset across epochs for different values of $\sigma$</td>
<td>105</td>
</tr>
<tr>
<td>7-1</td>
<td>Decision boundary obtained for MxKAC using c-loss ($\sigma_c = 0.5$) at time $t = 0$ and time $t = 1$ for multiple kernels kernel sizes $\sigma = {0.5, 1, 1.5, 2}$</td>
<td>114</td>
</tr>
<tr>
<td>7-2</td>
<td>Decision boundaries obtained for KAC using c-loss ($\sigma_c = 0.5$) for different kernel sizes at time $t = 0$ and time $t = 50$</td>
<td>115</td>
</tr>
<tr>
<td>7-3</td>
<td>Result on classification of synthetic non-stationary data using MxKAC</td>
<td>117</td>
</tr>
</tbody>
</table>
Kernel methods are popular for their unique ability to solve non-linear problems linearly by the implicit mapping of data into high dimension feature space. They have therefore, been established as powerful methods for solving machine learning problems. However, a method of kernel selection for these methods, specifically online kernel methods, still remains as an important problem without a widely accepted solution. Selecting an appropriate kernel can significantly improve the performance of an online learning system in small data sets but it is crucial in non-stationary environments.

This dissertation proposes a unifying treatment of both filtering and classification using the same model, learning algorithms and cost functions. It presents a novel online kernel selection method based on kernel least means square (KLMS) filtering, called mixture kernel least means square (MxKLMS). MxKLMS competitively combines hypotheses learned in multiple reproducing kernel Hilbert spaces (RKHSs) such that the final hypothesis exists in the sum of these spaces. This architecture allows an automatic online kernel(s) selection from a pool of predefined kernels, such that the learning system can adapt to changing properties of the signal by changing the kernels. The detail theoretical analysis corroborates its formulation and empirical findings. Moreover, by using a mechanism of tactically discarding irrelevant samples, its memory requirement and computational burden are reduced. We compare the performance of
MxKLMS trained with the conventional MSE criterion, with correntropy and show better performance in presence of impulsive noise.

Apart from filtering, another major topic of supervised learning is classification. Using the same correntropy criterion on the MxKLMS model, we show its advantages in classification. Therefore, an information theoretic learning based loss function called correntropy-loss (c-loss) is used in the KLMS framework to obtain a robust online classifier called kernel adaptive classifier (KAC). This is extended to the MxKLMS framework to get a robust online classifier, called mixture kernel adaptive classifier (MxKAC) and is tested in non-stationary data classification. Thus, we obtain a robust method suitable for online supervised learning that includes filtering as well as classification.
CHAPTER 1
INTRODUCTION

Kernel methods have been popular due to their solid mathematical foundations and a wide range of applications. A number of batch kernel learning methods, such as support vector machines (SVM) [8], kernel principal component analysis (KPCA) [54], kernel Fisher discriminant analysis (KFDA) [35], etc., and online kernel learning methods such as [19, 25, 45, 55, 70] have been proposed and successfully applied to important signal processing problems. Online methods have gained much popularity and preference over batch methods in recent years owing to some key differences between them. In batch methods, weight changes are accumulated over an entire presentation of the training data before being applied, making them computationally expensive. Whereas, in online methods, weights are updated using the local stochastic gradient to determine the direction of weight and update the weight at each training sample. This can be noisy and may converge slowly to optimal solution, but have less computational complexity and can handle big data well. This motivates us to develop improved methods for online learning to tackle supervised learning problems. Therefore, this dissertation presents improved kernel adaptive learning (KAL) methods for online supervised learning, based on kernel adaptive filtering (KAF) [45] framework.

KAF is a family of online kernel methods for filtering, which have successfully recast the linear adaptive filtering algorithms into high dimensional reproducing kernel Hilbert space (RKHS) and are shown to be very effective in solving system identification problems. An adaptive filter works by self-adjusting its parameters based on the minimization of cost function defined as a function of difference between the input and the desired response. Linear adaptive filters are only able to find the output that exists in the span of input data. Therefore, better solutions may exist outside the convex hull of the inputs. Hence, the reproducing kernel Hilbert space (RKHS) theory was used to solve non-linear filtering problems, giving rise to a family of non-linear adaptive filters.
A lucrative property of KAF is that, it is a family of recursive algorithms that lead to online solutions given by a convex optimization with a squared error cost function, unlike the neural networks. Furthermore, with the use of a universal kernel [20] which induces a dense space for a continuous function $f$, KAF can provide universal approximation to the desired function [45]. These beneficial aspects make it even more appealing to study its behavior and seek theoretical improvements to the current state of the art. With the kernel least mean square (KLMS) filter at the core, this work presents improved robust algorithms suitable for online supervised learning system, by focusing on two main aspects: the kernel map and the cost function.

Kernel methods are based on the implicit mapping of data into high dimensional feature space where a non-linear problem in input space can be formulated as a linear problem and thus can be solved using linear algorithms. Such a mapping is produced by a continuous, symmetric and semi positive definite kernel and so a careful selection of a kernel is required to achieve faster convergence and in some cases, the best solution. With the use of a universal kernel, the impact of kernel selection is minimal when data
is stationary and infinite. However, in case of finite data and non-stationary data, the performance in terms of speed of convergence and accuracy is highly dependent on the kernel choice. Therefore, kernel selection is one of the major problems in kernel learning literature without practical and widely accepted solution. Using cross validation for kernel selection is one of the common ways but is unreasonable for systems with timely changing statistics. This motivates us to find a better and adaptive method for appropriate kernel selection. Therefore, we present a novel online method called mixture kernel least mean square (MxKLMS), which automatically selects an appropriate kernel from a pool of available kernels by adapting to any system changes. MxKLMS gives a multi-kernel adaptive filtering formulation suitable for online kernel method such as KLMS, as an alternative to the multiple kernel learning (MKL) [16] formulation suitable only for batch methods where all training samples are available at once. MxKLMS algorithm is based on competitive combination of multiple KLMS adaptive filters solved in multiple RKHSs. By selecting the best kernel(s) at different times, MxKLMS can successfully model local characteristics, significantly improving the prediction accuracy.

Another important aspect of kernel adaptive learning is the cost function, as the learning process is based on minimizing the expected cost. The conventional adaptive filters use the least mean square (LMS) cost, due to its beneficial properties such as, convexity, mathematical tractability and simplicity. They exploit the second-order statistics of the error and can give a robust performance in presence of Gaussian noise. However, in presence of non-Gaussian noise and outliers, LMS is not a suitable cost function and thus, a robust cost function is required. Recently, information theoretic learning (ITL) have been efficiently used to train adaptive systems especially for noisy outlier environment. ITL optimizes the information content of the prediction error based on minimization of error entropy, to achieve the best performance in terms of information filtering [47]. Adaptive filters trained using different ITL criteria have demonstrated robust performances in various noisy conditions. One of such criteria is the recently proposed
generalized similarity measure called correntropy [30], that has been implemented in various ways with adaptive filtering, some of the examples being, correntropy based matched filtering [43], correntropy mace filter [23], maximum correntropy criterion for linear adaptive filters (MCC) [60] and for kernel adaptive filters (KMC) [76], etc. These methods have outperformed LMS in presence of non-Gaussian noise, corroborating the advantage of correntropy based criterion over LMS in adaptive filtering. Therefore, this work implements the MCC in the MxKLMS framework to achieve robust filtering method called, mixture kernel maximum correntropy criteria (MxKMC).

It is a known fact that filtering is based on prediction which is also a basis for solving classification problem. A classifier predicts response $y \in \{-1, 1\}$ or $\{0, 1\}$ (for binary classification) where each value of $y$ corresponds to classes $C_1$ and $C_2$. The value of $y$ can be interpreted as the probability that the class is $C_1$ or $C_2$, and thus, can be obtained from the predicted output $\hat{y}$ in winner take all fashion, as in neural network classifier. Therefore, this work extends the robust methods developed for filtering into classification by using correntropy as an approximation to the 0 - 1 loss used in classification. This new use of correntropy is called the correntropy loss or c-Loss [59]. c-loss gives a smooth approximation to the 0 - 1 Loss and the hinge loss for finite range of errors and for different values of its $\sigma$ parameter. Using c-loss in the Kernel Least Mean Square (KLMS) adaptive filtering [41] framework, we obtain a robust sequential adaptive classifier called kernel adaptive classifier (KAC). It is robust to overfitting even after prolonged training, compared to square loss and its performance is comparable to SVM even in very noisy conditions. Moreover, by using c-loss in MxKLMS framework we obtain robust mixture kernel adaptive classifier (MxKAC) for non-stationary online classification problems. Such problems arise from drifting data structures across time, that result in changes in class labels, discriminant function and thus, the decision boundary. We show the effectiveness of MxKAC in solving such problems by classifying a simple drifting mixture of Gaussian dataset.
CHAPTER 2
THEORETICAL BACKGROUND

2.1 Statistical Learning Theory

Given \( n \) observations, where each observation consists of a pair: a vector \( u_i \in \mathbb{R}^m \), \( \forall i = 1, \ldots, n \) and an associated desired response \( y_i \in \mathbb{R} \) (for filtering) or \( y_i \in \{-1, 1\} \) (for binary classification), assumed to be i.i.d realization of a random pair \((U, D)\). A learning machine is defined by a set of possible mappings \( u \rightarrow f(u, w) \), where the functions \( f(u, w) \) themselves are labeled by the adjustable parameters \( w \) \[8\]. Since a particular choice of \( w \) generates a trained machine, any learning machine requires some form of learning rule, which depends on an objective function. The objective function is defined as a difference between the unknown desired response \( y \) and the predicted response \( f(u, w) \), as \( L[f : (u, y)] = \text{loss}[f(u, w), y] \). In the general, we would like to minimize a population risk functional, defined by \[68\],

\[
R(w) = \int L(f(u, w), y) dP(u, y) \tag{2–1}
\]

Since the actual distribution \( P(u) \) is unknown, the empirical risk, \( R_{\text{emp}}(w) \) for a finite number of observations \( N \) is given by:

\[
R_{\text{emp}}(w) = \frac{1}{N} \sum_{i=1}^{N} L(f(u, w), y_i) \tag{2–2}
\]

For supervised problems, it is well known that a problem is learnable if \( R_{\text{emp}} \) converges uniformly to the true risk \( R \), which implies that the empirical risk minimizer is consistent \[8, 69\].

2.2 Kernel Methods and Reproducing Kernel Hilbert Space

Kernel methods are based on transforming the input data \( u_n \in \mathbb{U} \) into a high dimensional feature space of vectors \( \varphi(u_i) \) using a Mercer’s kernel. Mercer’s kernel is defined as a continuous, symmetric, positive-definite function \( \kappa : \mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R} \) \[3\]. One of
the commonly used Mercer’s kernels is Gaussian kernel:

\[ \kappa(u_i, u_j) = \exp \left( -\frac{\|u_i - u_j\|^2}{2\sigma^2} \right) \]  \hspace{1cm} (2–3)

where \( \sigma > 0 \) is the kernel width. Without loss of generality, in this chapter we will only consider the translation-invariant Gaussian kernel. According to the Mercer’s theorem, any Mercer’s kernel \( \kappa(u_i, u_j) \) induces a mapping \( \varphi \) from the input space \( \mathbb{U} \) to a high dimensional feature space \( \mathbb{F} \) such that the following relation holds:

\[ \kappa(u_i, u_j) = \langle \kappa(\cdot, u_i), \kappa(\cdot, u_j) \rangle = \langle \varphi(u_i), \varphi(u_j) \rangle \]  \hspace{1cm} (2–4)

and thus, \( \varphi(u_i) = \kappa(\cdot, u_i) \) defines the Hilbert space associated with the kernel. Some of the key properties of such kernels are briefly summarized as follows:

2.2.1 Reproducing Property

A particularly interesting characteristic of the feature space induced by a Mercer’s kernel is that it is a reproducing kernel Hilbert space (RKHS): i.e., the span of functions \{\kappa(\cdot, u) : u_n \in \mathbb{U}\} defines a unique functional Hilbert space [3, 38, 50]. The fundamental property of such spaces is the reproducing property of the kernel defined as:

\[ f(u) = \langle \kappa(\cdot, u), f \rangle, \forall f \in \mathbb{F} \]  \hspace{1cm} (2–5)

2.2.2 Universal Approximation Property

Another property of interest is the universal approximation property that can be achieved using a universal kernel [37], i.e., for any continuous input-output mapping \( f : \mathbb{U} \to \mathbb{R}, \forall \tau > 0, \exists \{u_i\}_{i \in N} \in \mathbb{U} \) and real number \( \{\alpha_i\}_{i \in N} \), such that,

\[ \|f - \sum_{i=1}^{N} \alpha_i \kappa(u_i, \cdot)\|_2 < \tau \]  \hspace{1cm} (2–6)

This property guarantees the convergence to the optimal solution even in case of nonlinear problems, given infinitely many samples.
2.2.3 Sum of Reproducing Kernels

Let $\kappa_1(u_i, u_j)$ and $\kappa_2(u_i, u_j)$ be the reproducing kernels corresponding to the classes $F_1$ and $F_2$ of functions with norms $\| \cdot \|_1$ and $\| \cdot \|_2$ respectively. Let, the corresponding RKHSs be $F_{\kappa_1}$ and $F_{\kappa_2}$ respectively. Then, $\kappa = \kappa_1 + \kappa_2$ is also a positive definite kernel inducing RKHS $F$ [3] and the class $F$ corresponding to $\kappa$ and RKHS $F$ is characterized as the class of all functions $f = f_1 + f_2$, where $f_i \in F_i$ and $f \in F$. For each such decomposition, the norm $\| \cdot \|$ in $F$ is defined by,

\[
\| f \|^2 = \min \left[ \| f_1 \|^2_1 + \| f_2 \|^2_2 \right]
\]

the minimum for all the decompositions of $f = f_1 + f_2$.

This can be easily extended to the finite sum of kernels:

\[
\kappa(u_i, u_j) = \sum_{m=1}^{P} \kappa_i(u_i, u_j)
\]

where, the final function $f \in F$ is given by, $f = \sum_{m=1}^{P} f_m$ and its norm is given by

\[
\| f \|_{F_\kappa}^2 = \inf_{\sum_{m=1}^{P} f_m} \sum_{i=1}^{P} \| f_m \|_{F_{\kappa_m}}^2
\]

Moreover, [56] shows that, for Gaussian kernel, it can be shown for $\beta > 0$, $F_{\beta \kappa} = F_\kappa$ and $\| \beta f \|_{F_\kappa}^2 = \beta \| f \|_{F_\kappa}^2$, so that, for $\beta \in [0, 1]$,

\[
\| f \|_{F_\kappa}^2 = \inf_{\sum_{m=1}^{P} \beta_m f_m \in F_{\kappa_m}} \sum_{i=1}^{P} \beta_m \| f_m \|_{F_{\kappa_m}}^2
\]

suggesting that integral versions of these representations may be available.

2.3 Kernel Adaptive Filtering

Adaptive filters are powerful recursive filtering methods that are able to implement finite impulse response (FIR) filters by updating filter parameters with each sample. These are based on learning the adjustable filter parameters $w$ by minimizing the distance between the filter output $y_i$ and the desired response $y_i$. By using kernels with
adaptive filters, a set of non-linear adaptive filters called kernel adaptive filters (KAF) [45] can be obtained that are able to learn highly non-linear problems simply by projecting samples into high dimensional spaces. The basic structure and working of a KAF can be explained by a block diagram as shown in Fig. 2-1. Central to the taxonomy of KAF is the kernel affine projection algorithm (KAPA) [31] that approaches the nonlinear input output map as, \( f(u(n)) = f(u(n-1)) + \varphi(u(n))e(n) \), where, \( e(n) \) is the error at the \( n^{th} \) iteration and \( \varphi \) is the non-linear feature map. The simplest algorithm of the family is the KLMS [42] using the stochastic gradient, and the most complex is the KRLS [13], which uses all the information of the input data up to the current sample.

2.4 Kernel Least Mean Square

Let \( u_n \in \mathbb{U} \subseteq \mathbb{R}^l \) be an input vector at time \( n \), and \( y_n \in \mathbb{Y} \subseteq \mathbb{R} \) be the desired response, which is a non-linear function of the input \( u_n \). The goal is to learn
a continuous input-output mapping $f : \mathbb{U} \to \mathbb{Y}$, based on the incoming input-output pair $\{u_i, y_i\}_{i=1}^N$. In KLMS, the hypothesis space for learning $f$ is the RKHS $\mathcal{F}$, induced by the positive-definite kernel $\kappa : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$ [3]. The underlying function $f \in \mathcal{F}$ can be learned by solving the following empirical risk minimization (ERM) problem:

$$
\min_{f \in \mathcal{F}} \sum_{i=1}^N (y_i - f(u_i))^2
$$

which has been shown to be well-posed in the sense of Hadamard in [31], when the least mean square algorithm is used in the minimization. Now, according to the reproducing property explained in (2–5),

$$
\langle f, \kappa(u, \cdot) \rangle_{\mathcal{F}} = f(u)
$$

And so, (2–8) can be written as,

$$
\min_{f \in \mathcal{F}} \sum_{i=1}^N (y_i - \langle f, \kappa(u_i, \cdot) \rangle_{\mathcal{F}})^2
$$

Following the usual KAF notations, let us denote the approximate functional $f$ by $\Omega \in \mathcal{F}$, so that the ERM in (2–10) is:

$$
\min_{\Omega \in \mathcal{F}} \sum_{i=1}^N (y_i - \langle \Omega, \varphi(u_i) \rangle_{\mathcal{F}})^2
$$

where, $\kappa(u, .) = \varphi(u)$, using Mercer’s theorem. Hence, the KLMS adaptive filtering boils down to solving this ERM problem (2–11) in the RKHS $\mathcal{F}$.

KLMS can simply be explained as LMS performed on the input-output pair $\{u_i, y_i\}_{i=1}^N$. Given the latest input-output pair $\{\varphi(u_n), y_n\}$, the current $\Omega_n$ is updated as

$$
\begin{align*}
\Omega_0 &= 0, e_n = y_n - \Omega_{n-1}^\top \varphi(u_n) \\
\Omega_n &= \Omega_{n-1} + \eta e_n \varphi(u_n) \\
\Omega_n &= \sum_{i=1}^n \eta e_i \varphi(u_i)
\end{align*}
$$

21
where, $\eta$ is the learning rate for the gradient update. Then the output at $n + 1$ is

$$y_{n+1} = \sum_{i=1}^{n} \eta e_i \kappa(u_i, u_{n+1})$$  \hspace{1cm} (2–13)

This solution is equivalent to that suggested by the representer theorem [53],

$$f(u) = \sum_{i=1}^{N} \alpha_i \kappa(u_i, u)$$  \hspace{1cm} (2–14)

where $\{\alpha_i = \eta e_i\}_{i=1}^{N}$ is the coefficient vector for linear combination.

## 2.5 Correntropy

**Definition:** Let $\{x_t, t \in T\}$ be a stochastic process with $T$ being an index set. Correntropy of the random variable $\|x_t, x_{t-\tau}\|$, where $x_t$ and $x_{t-\tau}$ are $x$ at time index $t$ and $t - \tau$ respectively, is given by,

$$V(t, t - \tau) = V(\tau) = E[\kappa(x_t, x_{t-\tau})]$$  \hspace{1cm} (2–15)
For finite samples $N$, and the Gaussian kernel,

$$\hat{V}_{\sigma_c}(\tau) = \frac{1}{N} \sum_{i=1}^{N} \kappa_{\sigma_c}(x_i, x_{i-\tau})$$  \hspace{1cm} (2–16)

where $\sigma_c$ is a kernel size for the Gaussian kernel. Unlike correlation which considers only the second order moments, correntropy takes into account all the even order moments, and thus, can provide more information pertaining to the higher order statistics of the data. A localized similarity measure is obtained with proper kernel size selection for the kernel. Fig. 2-2 shows contour plot of correntropy performance surface for 2D data for $\sigma_c = 0.5$. We can notice the gradual change in norms from $L_2$, for closer samples, to $L_1$ and finally $L_0$ for samples far apart [30]. Various successful applications of correntropy have been developed in the recent years in different fields of signal processing [46].
CHAPTER 3
ROBUST FILTERING USING MIXTURE KERNEL LEAST MEAN SQUARE

3.1 Background

With the evolving KAF literature, it is crucial to address a key issue of kernel selection in a principled manner. Selecting a kernel that can closely approximate the desired hypothesis largely impacts performance of a kernel method. Various (batch) multiple kernel learning (MKL) techniques \cite{12, 16, 27, 48, 61, 65} have been proposed recently to address this issue. However, the challenge in adaptive filtering is not just finding the best kernel for the entire spatial or temporal data, but also changing it based on the local behavior of the signal, which can be crucial in non-stationary environments. The batch methods are not suitable for sequential learning, since the latter requires the learning process to take place within a fixed reproducing kernel Hilbert space (RKHS) as the samples arrive one at a time. Hence, we proposed the mixture kernel least mean square (MxKLMS) \cite{44} adaptive filter, as a multi-kernel extension to the kernel least mean square (KLMS) \cite{41}, one of the simplest KAFs. The MxKLMS competitively combines multiple KLMS filters, each learned in a separate RKHS, such that the combination gives the final hypothesis in the sum of spaces at each time sample. The competitive combination is obtained by using a competitive gate that selects only the most relevant kernel(s) at every instance. These attributes make MxKLMS a suitable choice for multi-kernel adaptive filtering both theoretically and computationally.

Apart from MxKLMS, other algorithms such as multiple kernel normalized least mean square (MKNLMS) \cite{75} and multiple kernel least mean square (MKLMS) \cite{67} have also been proposed recently as the multi-kernel extension of KLMS. MxKLMS differs from these approaches in two aspects. First, MxKLMS does not a priori assume

\begin{flushend}

Portions of this Chapter are published for in the following manuscript: Rosha Pokharel, Sohan Seth, Jose C. Principe, "Mixture Kernel Least Mean Square", IJCNN, pp. 1-7, Aug. 2013.

24
the functional form consisting of weighted sum of multiple kernels. Instead, it arrives at this functional form as a consequence of its formulation, whereby the KLMS is simultaneously implemented in multiple RKHSs and the outputs are linearly combined to get the final output. It can therefore be explained in terms of empirical risk minimization (ERM), fortifying its theoretical foundation. Second, MxKLMS uses a competitive gating function to combine the outputs of individual RKHSs such that the filters compete among each other to explain the signal at each sample. It also makes MxKLMS robust to abrupt changes in the system, thus, significantly improving the performance. On the other hand, the other methods do not use a competitive framework for kernel selection.

3.2 Existing Multi-Kernel Adaptive Filtering Algorithms

Besides MxKLMS other multi-kernel adaptive filtering methods such as multikernel LMS (MKLMS) [67], multikernel NLMS (MKNLMS) [75] extend KLMS to accommodate multiple kernels. These methods share a strong commonality in the sense that they take a common functional form:

\[ y_{n+1} = \sum_{m=1}^{P} \sum_{i=1}^{n} \alpha_{m,i} \kappa_{m}(u_i, u_{n+1}) \]  

(3–1)

where \( i \) is index for samples and \( m \) for multiple kernels. Such construction allows the function to exist in the direct sum of RKHSs induced by individual kernels. As a consequence, MKNLMS, MKLMS estimate \( nP \) parameters and MxKLMS estimate \((n + 1)P\) parameters. However, these algorithms differ in the way the coefficients are estimated, both conceptually and computationally. While MKLMS and MKNLMS assume the functional form (3–1) a priori, and employs standard optimization tools to learn them with appropriate constraints, MxKLMS arrives at this form as a result of combining the outputs of KAF learned on individual RKHSs through a nonlinear gating function. Moreover, MKNLMS and MKLMS update all the \( nP \) filter coefficients at every new iteration, compared to MxKLMS which only updates the current \( P \) coefficients out of \((n + 1)P\) coefficients. Therefore, MxKLMS is computationally less expensive than the
other two. For the sake of completeness, the two algorithms are summarized briefly as follows:

### 3.2.1 Multikernel Least Mean Square

Multikernel least mean square arrives at the functional form (3–1) using the concept of vector valued RKHS, and time variant mapping to its element RKHSs. To elaborate, it maps the input samples into $P$ different RKHSs using time-varying mapping given by,

$$
\mathbf{u} \rightarrow \varphi^n(u) = [c^n_1 \varphi_1(u), c^n_2 \varphi_2(u), \ldots, c^n_P \varphi_P(u)]^T,
$$

where $n$ is the time index and $\{c^n_m\}_{n=1,2,\ldots}, \forall m \in \{1, \ldots, P\}$ is a sequence of time-varying parameters that is used to approximate the output $y_n$ such that, $\hat{y}_n = \langle \Omega, \varphi^n(u) \rangle$. The update of $\Omega$ can be conceptually performed based on the KLMS approach of using stochastic gradient descent on squared error, $e^2_n$, in the RKHS as follows:

$$
e^n_n = y_n - \Omega^n_{n-1} \varphi^n(u_n)
$$

$$
\Omega^n_n = \Omega^n_{n-1} + \eta e^n_n \varphi^n(u_n) = \eta \sum_{i=1}^n e_i \varphi^i(u_i)
$$

$$
\hat{y}^n_{n+1} = \eta \sum_{i=1}^n e_i \langle \varphi^i(u_i), \varphi^{n+1}(u_{n+1}) \rangle
$$

where, $\eta$ is the learning rate. Here the concept of vector valued RKHS is invoked to estimate the inner product as follows,

$$
\langle \varphi^i(u_i), \varphi^{n+1}(u_{n+1}) \rangle
$$

$$
= \sum_{m=1}^P \langle c^i_m \varphi_m(u_i), c^{n+1}_m \varphi^{n+1}_m(u_{n+1}) \rangle
$$

$$
= \sum_{m=1}^P c^i_m c^{n+1}_m \kappa_m(u_i, u_{n+1})
$$

which leads to the final form

$$
\hat{y}^n_{n+1} = \eta \sum_{i=1}^n \sum_{m=1}^P \omega_{n+1,i,m} \kappa_m(u_i, u_{n+1})
$$
where, \( \omega_{n+1,i,m} = e_i c_{m}^{n+1} c_m \). These coefficients are then simply learned by normalized LMS algorithm as,

\[
\omega_{n+1,i,m} = \omega_{n,i,m} - \eta e_i \frac{\kappa_m(u_i, u_{n+1})}{\epsilon + \kappa_m^2(u, u_{n+1})}
\]  

(3–7)

3.2.2 Multikernel Normalized Least Mean Square

Multikernel normalized least mean square algorithm directly assumes the functional form (3–1). The filter coefficients are then updated recursively by taking their projection onto a zero-instantaneous error hyperplane in a parameter space [75]. This uses \( P \) different kernels \( \kappa_m, m \in \{1, \ldots, P\} \) and also creates a dictionary based on the coherence-based sparsification criteria. However, as mentioned earlier, we will present this method with no sparsification criteria.

Let, \( h_{n,m} \in \mathbb{R}, m \in \{1, \ldots, P\} \) be the coefficient vectors at \( n \)th iteration. Then, the estimate \( \hat{y} \) at \( n+1 \)th iteration is given by:

\[
\hat{y}_{n+1} = \sum_{m=1}^{P} \sum_{i=1}^{n} h_{i,m} \kappa_m(u_i, u_{n+1}) = \text{tr} \left( H_n^T K_n \right)
\]  

(3–8)

where \( K_{n+1} = [\kappa_{n+1,1}, \kappa_{n+1,2}, \ldots, \kappa_{n+1,n}] \), and \( H_n = [h_1, \ldots, h_n] \), with \( h_i = [h_{i,1}, h_{i,2}, \ldots, h_{i,P}] \in \mathbb{R}^P \), and

\[
\kappa_{n+1,i} = [\kappa_1(u_{n+1}, u_i), \kappa_2(u_{n+1}, u_i), \ldots, \kappa_P(u_{n+1}, u_i)]^T \in \mathbb{R}^P.
\]

The coefficient matrix is then updated as

\[
H_{n+1} = \tilde{H}_n + \eta \frac{y_n - (\tilde{K}_n \tilde{H}_n)}{\rho + \|K_n\|_2} \tilde{K}_n
\]

(3–9)

where, \( \eta \in [0, 2] \) is the learning rate, \( \rho \) is a small positive constant, \( \tilde{H}_n = [H_n \ 0] \) and \( \tilde{K}_n = [K_n \ \tilde{K}_n] \), with \( \tilde{K}_n = [\kappa_1(u_n, u_0), \ldots, \kappa_P(u_n, u_0)]^T \), \( 0 \in \mathbb{R}^P \) is a zero vector. A generalization to KNLMS can be obtained when \( P = 1 \).

3.3 Existing Multiple Kernel Learning (MKL) Techniques

The MxKLMS along with the existing multi-kernel adaptive filtering methods differ vastly from the MKL techniques, one of the main aspects being their functional form,
which is,

\[ f(\mathbf{u}) = \sum_{m=1}^{P} \sum_{i=1}^{n} \gamma_{m} \alpha_{i} \kappa_{m}(\mathbf{u}_{i}, \mathbf{u}) \]  

(3–10)

A clear difference can be noticed between the multi-kernel adaptive filtering formulation in (3–1) and the MKL formulation in (3–10). In the former, the learning takes place in the multiple hypotheses spaces that are fixed, while in the later, the learning takes place in the final hypothesis space which is changing at each iteration. This is a significant factor in case of sequential learning methods where the samples come one at a time, and so the MKL techniques are not suitable here.

To elaborate, let \( \Omega_{n} \) be the weight function obtained at iteration \( n \), that is learned in RKHS \( F_{\psi(n)} \), induced by kernel \( \kappa_{\psi(n)} \). When \( \psi(n) \) is updated to \( \psi(n + 1) \), a new kernel \( \kappa_{\psi(n+1)} \) is obtained as a result of new kernel combination. Thus, \( F_{\psi(n+1)} \) induced by \( \kappa_{\psi(n+1)} \) is different from \( F_{\psi(n)} \) and so are their functional mappings \( \varphi_{\psi(n+1)}(\mathbf{u}_{n+1}) \) and \( \varphi_{\psi(n)}(\mathbf{u}_{n}) \), respectively. Using KLMS update rule from (2–12), the new weight function is given by,

\[ \Omega_{n+1} = \sum_{i=1}^{n} \eta \varepsilon_{i} \varphi_{\psi(n)}(\mathbf{u}_{i}) \]  

(3–11)

Then, the output is,

\[ \hat{y}_{n+1} = \Omega_{n+1}^{T} \varphi_{\psi(n+1)}(\mathbf{u}_{n+1}) \]  

(3–12)

Clearly, from (3–11) and (3–12), the inner product between two feature vectors \( \varphi_{\psi(n)}(\mathbf{u}_{n}) \) and \( \varphi_{\psi(n+1)}(\mathbf{u}_{n+1}) \) belonging to two different RKHS \( F_{\psi(n)} \) and \( F_{\psi(n+1)} \), respectively, cannot be replaced by a Mercer’s kernel, and thus, MKL formulation does not trivially hold for KLMS. Certainly, the process can be carried out in practice, but it can no longer be framed as a kernel adaptive filter, the core of which lies in the reasoning of learning a function in a specific RKHS given by a kernel. MKL techniques are therefore, more appropriate for the batch methods where samples are available all at once.
3.4 MxKLMS: Multi-Kernel Adaptive Filtering

Let us start with a set of \( P \) different positive definite kernels \( \kappa_m, m \in \{1, \ldots, P\} \) inducing \( P \) respective RKHSs \( \mathcal{F}_{\kappa_m} \). Let, \( \varphi_m : U \to \mathcal{F}_{\kappa_m} \) be the corresponding mapping functions. MxKLMS is based on learning functions \( f_m \) in individual RKHS \( \mathcal{F}_{\kappa_m} \) and then linearly combining the functions, thus, \( f = \sum_{m=1}^{P} \beta_m f_m \). This allows sequentially learning \( \beta \) together with \( f_m \in \mathcal{F}_{\kappa_m} \) in their respective RKHSs. Notice that, unlike standard MKL algorithms where the hypothesis \( f \) is learned in the RKHS induced by \( \sum_m \beta_m \kappa_m \), here the hypothesis \( f_m \) is learned in the RKHS induced by the kernels \( \kappa_m \). However, the final function \( f \) still exists in the sum of spaces,

\[
\mathcal{F}_\kappa = \{ f = \sum_{m=1}^{P} f_m | f_m \in \mathcal{F}_{\kappa_m} \} \tag{3–13}
\]

with its norm defined by,

\[
\|f\|_{\mathcal{F}_\kappa}^2 = \inf_{f \in \mathcal{F}_\kappa} \sum_{m=1}^{P} \|f_m\|_{\mathcal{F}_{\kappa_m}}^2 \tag{3–14}
\]

since \( \beta_m \) is finite \([3, 56]\). This is a common theme among existing multi-kernel extensions of KLMS. The purpose of introducing the stochastic vector \( \beta \) is to create competition among different RKHSs, and thus, give higher weights to a few relevant kernels.

Now, \( f \) can be obtained as the stochastic approximation to the following ERM problem in each RKHS \( \mathcal{F}_{\kappa_m} \):

\[
\min_{f_m \in \mathcal{F}_{\kappa_m}} \sum_{i=1}^{N} [y_i - f_m(u_i)]^2 = \min_{f_m \in \mathcal{F}_{\kappa_m}, \beta_m > 0, \sum_m \beta_m = 1} \sum_{i=1}^{N} \left[ y_i - \sum_{m=1}^{P} \beta_m f_m(u_i) \right]^2 \tag{3–15}
\]

which requires learning an additional parameter \( \beta_m \). This is biconvex in \( f_m \) and \( \beta_m \), and can be solved by alternating minimization with respect to \( f_m \) and \( \beta_m \).

For certain \( \beta_m \), using reproducing property as in (2–9), the ERM problem using the KAF notations is

\[
\min_{\Omega_m \in \mathcal{F}_{\kappa_m}} \sum_{i=1}^{N} \left[ y_i - \sum_{m=1}^{P} \beta_m \Omega_m, \varphi_m(u_i) \right]^2 \tag{3–16}
\]
Algorithm 1: Mixture Kernel Least Mean Square Algorithm

Input:
\( P \) Kernels

Initialization

\( \hat{y}_0 = 0, \)

Gate parameter \( v_{m,0} = \frac{1}{P}, \forall m \in \{1, \ldots, P\}, \)

Learning rates: \( \eta, \mu > 0, \)

Repeat \( \forall i \in \{1, \ldots, n\} \):

1. Evaluate: \( \hat{y}_i \)
2. Compute error: \( e_i = y_i - \hat{y}_i \)
3. Update gate parameter: \( v_{m,i} = v_{m,i-1} - \mu \nabla v_{m,i} \)
4. Compute gate output: \( \beta_{m,i} \)
5. Update filter coefficient \( \alpha_{m,i} = \eta \beta_{m,i} e_i \)

Until convergence

Given the input-output pair \( \{\varphi_k(u_n), y_n\} \), at \( n^{th} \) iteration, \( \Omega_{k,n} \) is given by the following LMS update rule:

\[
\begin{align*}
\Omega_{k,0} &= 0 \\
e_n &= y_n - \sum_{m=1}^{P} \beta_m \left( \Omega_{m,n-1} \varphi_m(u_n) \right) \\
\Omega_{k,n} &= \Omega_{k,n-1} + \eta e_n \beta_k \varphi_k(u_n) \\
\Omega_{k,n} &= \sum_{i=1}^{n} \alpha_{k,i} \varphi_k(u_i)
\end{align*}
\] (3–17)

where, \( \alpha_{k,i} = \eta e_i \beta_k = [\alpha_{k,1}, \ldots, \alpha_{k,n}] \) is the coefficient vector for linear combination.

In addition to learning \( \Omega_{k,i} \), the stochastic weights \( \beta_k \) are learned using a non-linear gating function,

\[
\beta_k = \frac{\exp(v_k)}{\sum_{j=1}^{P} \exp(v_j)}
\] (3–18)

where, the gate parameter \( v_k \) is the intermediate weight at the \( n^{th} \) iteration. The gating function (7–6) maintains convexity in \( \beta \). This requires learning \( v_k \) using an additional
update rule:
\[ v_{k,n+1} = v_{k,n} - \mu \frac{\partial e_n^2}{\partial v_{k,n}} \]  
(3–19)

where, \( \mu \) is the learning rate. The gradient can be solved as follows:
\[ \nabla v_{k,n} = \frac{\partial e_n^2}{\partial v_{k,n}} = \frac{\partial e_n^2}{\partial \beta_{k,n}} \frac{\partial \beta_{k,n}}{\partial v_{k,n}} = 2e_n \nabla_y \nabla_s \]  
(3–20)

where, \( \nabla_y = \frac{\partial}{\partial \beta_{k,n}} (y - \hat{y}_n) \), and \( \nabla_s = \frac{\partial}{\partial v_{k,n}} \beta_{k,n} \). Then,
\[ \nabla_s = \frac{\partial}{\partial v_{k,n}} \left[ \frac{\exp(v_{k,n})}{\sum_{m=1}^{P} \exp(v_{m,n})} \right] \]
\[ = \frac{\exp(v_{k,n})}{\sum_{m=1}^{P} \exp(v_{m,n})} - \left( \frac{\exp(v_{k,n})}{\sum_{m=1}^{P} \exp(v_{m,n})} \right)^2 \]
\[ = \beta_{k,n} (1 - \beta_{k,n}) \]  
(3–21)

\[ \nabla_y = -\frac{\partial}{\partial \beta_{k,n}} \left[ \sum_{m=1}^{P} \beta_{m,n} \hat{y}_{m,n} \right] \]
\[ = -\frac{\partial}{\partial \beta_{k,n}} \left( \beta_{k,n} \hat{y}_{k,n} + \sum_{l=1, l \neq k}^{P} \beta_{l,n} \hat{y}_{l,n} \right) \]
\[ = -\hat{y}_{k,n} - \frac{\partial}{\partial \beta_{k,n}} \sum_{l=1, l \neq k}^{P} \beta_{l,n} \hat{y}_{l,n} \]  
(3–22)

Using the convexity on \( \beta \),
\[ \sum_{l=1, l \neq k}^{P} \beta_{l,n} \hat{y}_{l,n} = \sum_{l=1}^{P} \left( 1 - \sum_{j=1, j \neq l}^{P} \beta_{j,n} \right) \hat{y}_{l,n} \]  
(3–23)

So that,
\[ \frac{\partial}{\partial \beta_{k,n}} \sum_{l=1, l \neq k}^{P} \beta_{l,n} \hat{y}_{l,n} = -\sum_{l=1, l \neq k}^{P} \hat{y}_{l,n} \]  
(3–24)
Therefore,

\[ \nabla v_{k,n} = -2e_n \left[ \hat{y}_{k,n} - \sum_{l=1 \atop l \neq k}^{P} \frac{\sum_{i=1}^{n} \beta_{k,n} \left[ 1 - \beta_{k,n} \right]} {\beta_{k,n}} \right] \]

(3–25)

where, \( \hat{y}_{k,n} = \Omega_{k,n-1}^T \varphi_k(u_n) \). Therefore, the final function estimation \( \hat{y}_{n+1} \) is given by,

\[
\hat{y}_{n+1} = \Omega(u_{n+1}) = \sum_{m=1}^{P} \beta_{m} \Omega_{m}(u_{n+1})
\]

\[
= \sum_{m=1}^{P} \beta_{m} \Omega_{m,n}^T \varphi_{m}(u_{n+1})
\]

\[
= \sum_{m=1}^{P} \sum_{i=1}^{n} \beta_{m} \alpha_{m,i} \kappa_{m}(u_{i}, u_{n+1})
\]

(3–26)

Hence, we arrive at the MxKLMS formulation in (3–26) as a result of inherent optimization in multiple RKHSs. This algorithm can be thought of as projecting each sample into multiple RKHS where KLMS is performed simultaneously and the final prediction of each sample at \( n^{th} \) instance is given by the weighted combination of individual solution. This process can be visualized as shown in Fig. 3-1. The other existing multi-kernel extensions of KLMS differ from this approach as their weighted combination of kernels is assumed a priori, and the coefficients are found in a separate approach than that used to arrive at this expression.

Notice that, a competitive gate can be generally obtained using the softmax activation function:

\[ \beta_k = \frac{\exp\left( \frac{\psi_k}{\tau} \right)} {\sum_{j=1}^{P} \exp\left( \frac{\psi_j}{\tau} \right)} \]

(3–27)

where, \( \tau \) is a positive parameter called temperature [63]. High temperature causes all nearly equiprobable selection, making the filters cooperative. Low temperature causes greedy selection, making the filters competitive. The function defined in (7–6) is a special case of (3–27) with \( \tau = 1 \) chosen for simplicity. A sigmoid gate can also be used
that allows multiple filters to be used cooperatively:

\[
\beta_k = \frac{1}{1 + \exp(-\nu_k(n))}
\]  

(3–28)

### 3.5 Convergence Analysis of MxKLMS

Convergence of an adaptive filter is a key factor in determining its behavior. Mean square convergence analysis based on the energy conservation relation has been established as a popular tool for linear as well as kernel adaptive filters [1, 2, 10, 11, 31, 51, 74]. In this section, we first derive the convergence criteria for each component filter \(\Omega_k\) based on the energy conservation relation, and then show that given the convergence of each component filter, the entire filter MxKLMS converges. In addition, the steady state analysis of the filter is presented that establishes an upper bound for the steady state error.
3.5.1 Energy Conservation Relation

Consider the non-linear system where the desired signal $y_n$ is related to the input signal $u_i$ at $n^{th}$ instance as follows:

$$y_n = \Omega^o \varphi(u_n) + e^o_n$$  \hspace{1cm} (3–29)

where, $\Omega^o \in \mathbb{F}_\kappa^o$ is the optimal weight function in some RKHS induced by the feature map $\varphi$ and $e^o_n$ is the measurement noise, which is assumed to be zero-mean. This is a valid assumption in case of MxKLMS, as the goal here is still to find the final filter, that is closest to the optimal.

Let us define some notations and variables for the convenience of future use.

**Weight error vector**: The weight error vector for combination filter is defined as the difference between the optimal weight function $\Omega^o$ and the combination filter weight function $\Omega$. Also, since the final filter is a convex combination of filters $\Omega_m \in \mathbb{F}_\kappa^m$ such that, $\Omega = \sum_{m=1}^{P} \beta_m \Omega_m$, the weight error vector for each component filter can be represented as the difference between the optimal weight function $\Omega^o$ and the component filter weight function $\Omega_m$. This is a well known approach also used in convex combination of adaptive filters \[2\]

$$\Delta \Omega_{m,n-1} = \Omega^o - \Omega_{m,n-1}$$

for the $m^{th}$ component filter $\forall m \in \{1, ..., P\}$

$$\Delta \Omega_{n-1} = \Omega^o - \Omega_{n-1}$$  \hspace{1cm} (3–30)

for the combination filter

Similarly, the a priori error, a posteriori error and prediction error can be defined as follows:
A priori error:

\[ e_{m,n}^a = \Delta \Omega_{m,n-1}^\top \varphi_m(u_n) \] (3–31)
for the \(m\)th component filter, \(\forall m \in \{1, \ldots, P\}\)

\[ e_n^a = \Delta \Omega_{n-1}^\top \varphi_m(u_n) \] (3–32)
for the combination filter

A posteriori error:

\[ e_{m,n}^p = \Delta \Omega_{m,n}^\top \varphi_m(u_n) \] (3–33)
for the \(m\)th component filter, \(\forall m \in \{1, \ldots, P\}\)

\[ e_n^p = \Delta \Omega_n^\top \varphi_m(u_n) \] (3–34)
for the combination filter

Prediction error:

\[ e_n = y_n - \hat{y}_n \]
\[ = (\Omega^\top - \Omega_{n-1}^\top) \varphi_m(u_n) + e^o_n \]
\[ = e_{a,n} + e^o_n \] (3–35)

Now, subtracting \(\Omega^o\) from both sides of the MxKLMS update equation (3–17) for each filter,

\[ \Delta \Omega_{m,n} = \Delta \Omega_{m,n-1} - \eta e_n \beta_{m,n} \varphi_m(u_n) \] (3–36)

We will be using \(\beta_{m,n}\) instead of simply \(\beta_m\) from here on as the evolution of \(\beta_m\) over time also plays an important role for the convergence analysis. Multiplying the transpose of
(3–36) by \( \varphi_m(u_n) \) on both sides,

\[
e^p_{m,n} = e^a_{m,n} - \eta e_n \beta_{m,n} \kappa_m(u, u_i)
\]

\[
e^a_{m,n} - \eta e_n \beta_{m,n}
\] (3–37)

where, \( \kappa_m(u, u_i) = 1 \) for normalized Gaussian kernel. Squaring both sides of (3–36) and after some manipulations, we get,

\[
\| \Delta \Omega_{m,n} \|^2 + e^{a^2}_{m,n} = \| \Delta \Omega_{m,n-1} \|^2 + e^{p^2}_{m,n}
\] (3–38)

This energy relation is analogous to that of LMS filter and KLMS derived in [10]. It shows the relationship between the energies of the weight-error vector and a priori and a posteriori estimation errors for each \( m \)th component filter.

3.5.2 Mean Square Convergence Criteria

In this section, we derive the criteria for mean square convergence given the energy conservation relation in (3–38). Using (3–37) in (3–38), and after further manipulations, we get,

\[
\| \Delta \Omega_{m,n} \|^2 - \| \Delta \Omega_{m,n-1} \|^2 = -2\eta e^a_{m,n}e_n \beta_{m,n} + \eta^2 e^2_n \beta^2_{m,n}
\] (3–39)

Taking expectations on both sides and using the relation \( e_n = e^a_n + e^o_n \) from (3–35):

\[
E [ \| \Delta \Omega_{m,n} \|^2 ] - E [ \| \Delta \Omega_{m,n-1} \|^2 ] = -2\eta E [ e^a_{m,n} e^a_n \beta_{m,n} + e^a_{m,n} e^o_n \beta_{m,n} ] + \eta^2 E [ e^2_n \beta^2_{m,n} ]
\]

\[
+ 2\eta^2 E [ e^o_n e^2_n \beta^2_{m,n} ] + \eta^2 E [ e^2_n \beta^2_{m,n} ]
\] (3–40)

**Assumption 1**: Measurement noise \( e^o_n \) is zero-mean, independent and identically distributed (i.i.d.), with variance \( \sigma^2_o \) and independent of the a-priori error \( e^a_n \). This is a commonly used assumption in the convergence analysis of adaptive filters. A sufficient condition for independence between \( e^o_n \) and \( e^a_n \) is the independence between \( e^o_n \) and the input sequence \( u \) [11]. It follows that \( e^o_n \) is also independent of \( e^o_{m,n} \).
Assumption 2: Measurement noise $e_n^o$ is independent of $\beta_{m,n}$ so that, $E[e_n^o \beta_m] = E[e_n^o]E[\beta_{m,n}]$

Using these assumptions in (3–40), we get,

$$E[\|\Delta \Omega_{m,n}\|^2] - E[\|\Delta \Omega_{m,n-1}\|^2] = \eta^2 E[\{\sigma_o^2 + e_n^2\}\beta_{m,n}^2] - 2\eta E[e_{m,n}^a e_n^a \beta_{m,n}] \quad (3–41)$$

For mean square convergence,

$$E[\|\Delta \Omega_{m,n}\|^2] \leq E[\|\Delta \Omega_{m,n-1}\|^2]$$

$$\iff \eta^2 E[\{\sigma_o^2 + e_n^2\}\beta_{m,n}^2] \leq 2\eta E[e_{m,n}^a e_n^a \beta_{m,n}]$$

$$\implies \eta \leq \frac{2E[e_{m,n}^a e_n^a \beta_{m,n}]}{E[\{\sigma_o^2 + e_n^2\}\beta_{m,n}]} \quad (3–42)$$

To ensure the monotonic decrease of weight error power (WEP), $E[\|\Delta \Omega_{m,n}\|^2]$, the step size $\eta$ must be chosen such that, $\forall m$,

$$0 < \eta \leq \frac{2E[e_{m,n}^a e_n^a \beta_{m,n}]}{E[\{\sigma_o^2 + e_n^2\}\beta_{m,n}]} \quad (3–43)$$

This requires, $\forall m$,

$$E[e_{m,n}^a e_n^a \beta_{m,n}] > 0 \quad (3–44)$$

and it suffices to have $e_{m,n}^a e_n^a > 0$ for (3–44) as $\beta_{m,n}$ is positive. Therefore, sufficient condition for mean square convergence of each filter is,

$$e_{m,n}^a e_n^a > 0$$

$$0 < \eta \leq \frac{2E[e_{m,n}^a e_n^a \beta_{m,n}]}{E[\{\sigma_o^2 + e_n^2\}\beta_{m,n}]} \quad (3–45)$$

Given the condition (3–45) satisfies, for $\bar{n} > n$, $\epsilon > 0$, we have,

$$E[\|\Omega_{m,\bar{n}} - \Omega_o\|^2] \leq \epsilon \quad \forall m \in \{1, \ldots, P\} \quad (3–46)$$

Since, $\Omega = \sum_{m=1}^{P} \beta_m \Omega_m$, $\sum_{m=1}^{P} \beta_m = 1$, $\beta_m \geq 0$, it is easy to see that

$$E[\|\Omega_{\bar{n}} - \Omega_o\|^2] \leq \epsilon \quad (3–47)$$
Therefore, (3–45) is a sufficient condition for mean square convergence of MxKLMS.

3.5.3 Steady-State Mean Square Performance

For the steady state mean square performance analysis of the filter, we are interested in the limiting values as $n \to \infty$. So here we define the excess mean square error (EMSE) of the filters as:

$$J_{\text{ex}, \infty}^m = \lim_{n \to \infty} E \left[ e_{m,n}^2 \right]$$  \hspace{1cm} (3–48)

for the $m^{th}$ component filter, $\forall m \in \{1, \ldots, P\}$

$$J_{\infty}^m = \lim_{n \to \infty} E \left[ e_n^2 \right]$$  \hspace{1cm} (3–49)

for the combination filter.

Now, (3–42) holds and the algorithm reaches steady state,

$$\lim_{n \to \infty} E \left[ \| \Delta \Omega_{m,n} \|^2 \right] = \lim_{n \to \infty} E \left[ \| \Delta \Omega_{m,n-1} \|^2 \right]$$ \hspace{1cm} (3–50)

And so,

$$\eta^2 \lim_{n \to \infty} E \left[ \{ \sigma_o^2 + e_n^2 \} \beta_{m,n}^2 \right] = 2 \eta \lim_{n \to \infty} E \left[ e_{m,n}^a e_n^a \beta_{m,n} \right]$$ \hspace{1cm} (3–51)

We know that (3–51) is true for all component filters. Hence, summing (3–51) for all $m \in \{1, \ldots, P\}$, we get:

$$\frac{\eta^2}{2} \lim_{n \to \infty} E \left[ \sum_{m=1}^{P} \beta_{m,n}^2 \{ \sigma_o^2 + e_n^2 \} \right] = \lim_{n \to \infty} E \left[ \sum_{m=1}^{P} \beta_{m,n} e_{m,n}^a e_n^a \right]$$ \hspace{1cm} (3–52)

Now, the combination filter a priori error $e_n^a$ can be written in terms of component filter a priori errors $e_{m,n}^a$ as follows:

$$e_n^a = (\Omega_o - \Omega_{n-1})^\top \varphi(u_n)$$

$$= \Omega_o(u_n) - \Omega_{n-1}(u_n)$$

$$= \sum_{l=1}^{P} \beta_{l,n} \Omega_o(u_n) - \sum_{l=1}^{P} \beta_{l,n} \Omega_{l,n-1}(u_n)$$
\[
\sum_{l=1}^{P} \beta_{l,n} (\Omega^\top - \Omega_{l,n-1})^\top \varphi_i(u_n)
\]
\[
= \sum_{l=1}^{P} \beta_{l,n} e_{l,n}^2
\]  
\[(3-53)\]

Assumption 3: In steady state, \(\beta_{g,n}\) is independent of \(e_n^2, \forall g \in \{1, \ldots, P\}\). Given a sufficiently small learning rate for \(v_{g,n}\), this condition is reasonable as in steady state, \(v_{g,n}\) and thus, \(\beta_{g,n}\) converges to some steady state value such that the assumption is true.

A simplest case of this is when \(v_{g,n}\) approaches its limiting values, say \(\pm v^+\), in steady state as in these situations \(\beta_{g,n} [1 - \beta_{g,n}]\) in \((3-19)\) is close to zero \([2]\). This also gives,
\[
E[\beta_{g,n} e_n^2] = E[\beta_{g,n}^2] E[e_n^2].
\]

Using \((3-53)\) in \((3-52)\), with assumption 3, the final expression for EMSE of overall filter can be obtained as:
\[
J_{ex}^\infty = \eta^2 \lim_{n \to \infty} E \left[ \sum_{m=1}^{P} \beta_{m,n}^2 \right] [\sigma_o^2 + J_{ex}^\infty]
\]  
\[(3-54)\]

Now, let us look at this expression by considering the following different cases.

**Case 1:** For some \(r \in \{1, \ldots, P\}\), \(\lim_{n \to \infty} E[\beta_{r,n}] \to 1\) so that, \(\lim_{n \to \infty} E[\beta_{l,n}] \to 0, \forall l \neq r, l \in \{1, \ldots, P\}\). Consequently, \(\lim_{n \to \infty} E \left[ \sum_{m=1}^{P} \beta_{m,n}^2 \right] \to 1\), giving,
\[
J_{ex}^\infty \approx \frac{\eta}{2 - \eta} \sigma_o^2
\]  
\[(3-55)\]

which is the steady-state EMSE for KLMS \([10]\). This suggests that the steady-state EMSE for MxKLMS is as good as that of KLMS when a single filter is selected. Also, the misadjustment of MxKLMS in this case is \([18, 51]\):
\[
M = \frac{J_{ex}^\infty}{\sigma_o^2} \approx \frac{\eta}{2 - \eta}
\]  
\[(3-56)\]

**Case 2:** For \(r = \{r_1, \ldots, r_R\} \subseteq \{1, \ldots, P\}\), \(\lim_{n \to \infty} E[\beta_{r,n}] \approx \beta_j^v\), where, \(j \in \{1, \ldots, R\}\), such that, \(\sum_{j=1}^{R} \beta_j^v \approx 1\). Consequently, for \(l = \bar{r} = \{l_1, \ldots, l_L\}\), \(\lim_{n \to \infty} E[\beta_{l,n}] \approx 0\). Since,
$0 < \beta_{\eta}^{u} < 1$, we have, $E\left[\beta_{\eta}^{u2}\right] < E\left[\beta_{\eta}^{u}\right]$, so that,

$$J_{ex}^{\infty} < \frac{\eta}{2 - \eta} \sigma_{o}^{2}$$  \hspace{1cm} (3–57)

Therefore, when the final filter is a combination of more than one filter, the steady-state EMSE is always less than that of KLMS, irrespective of initial pool of kernels. The misadjustment in this case is

$$M < \frac{\eta}{2 - \eta}$$  \hspace{1cm} (3–58)

From (3–55) and (3–57), we establish an upper bound on the steady-state EMSE for MxKLMS:

$$J_{ex}^{\infty} \leq \frac{\eta}{2 - \eta} \sigma_{o}^{2}$$  \hspace{1cm} (3–59)

and on misadjustment:

$$M \leq \frac{\eta}{2 - \eta}$$  \hspace{1cm} (3–60)

Clearly, the upper bound for EMSE increases with increasing step-size or noise power and does not depend on the kernel size. This is consistent with the fact that, for KLMS with any kernel size, the unknown weight vector $\Omega^{\circ}$ always lies in, or arbitrarily close to, the subspace spanned by the input sequence $\{\varphi(u_{i})\}$, adhering to the universal approximation property [10]. This implies that MxKLMS converges to the desired solution despite the initial pool of kernels provided, which we also show empirically later in the section 3.6.3. It should be noted that MxKLMS is at least as good as KLMS in approximating the unknown weight vector $\Omega^{\circ}$, which adds a significant advantage to MxKLMS. However, from (3–43) and (3–60), the convergence and the steady state error depends on the selection of an appropriate step size thus, leading to solutions with small misadjustment. Therefore, a careful selection of step-size is necessary. Considering
(3–54) without the limits, it is easy to see that,

$$\eta = \frac{2 E \left[ e_n^2 \right]}{E \left[ \sum_{m=1}^{p} \beta_{m,n}^2 \right] E \left[ \sigma^2 + e_n^2 \right]}$$  

(3–61)

Since the quantity \((x^2/\{a^2 + x^2\}) \leq 1\),

$$\eta \leq \frac{2}{E \left[ \sum_{m=1}^{p} \beta_{m,n}^2 \right]}$$  

(3–62)

Following the similar arguments as in Case 1 and Case 2 we can come to a simpler criteria for selecting the step-size, i.e., it is less than 2 for single kernel selection and has a larger upper bound for multiple kernel selection.

### 3.6 Experiments and Results

In this section, we present experiments to explain the general working of MxKLMS, and demonstrate its performance compared to KLMS based on experiments for nonlinear channel equalization, and online prediction of a synthetic non-stationary data. We also demonstrate the ability of MxKLMS to select the most relevant kernels from a large pool of 41 predefined kernels and compare the performance of MxKLMS with the existing multi-kernel methods MKLMS and MKNLMS based on online prediction of Santa Fe laser data and Lorenz time series.

#### 3.6.1 Nonlinear Channel Equalization

Channel equalization is a well known application for LMS and KLMS. The problem setting is as follows: when a transmitter sends binary signal \(\{s_1, s_2, \ldots, s_N\}\), the signal at the receiver end of channel is further corrupted by additive i.i.d. Gaussian noise, and is observed as \(\{u_1, u_2, \ldots, u_N\}\). By using channel equalization one can construct an inverse filter that reproduces the original signal with the lowest error rate possible. This can be formulated as a regression problem where, \(u_i = \{u_{i+D}, u_{i+D-1}, \ldots, u_{i+D-L+1}\}^\top\) and \(y_i = s_i\), where, \(D\) and \(L\) are the equalization time lag and time embedding length respectively. In this experiment, the channel is modeled by \(z_i = s_i + 0.5s_{i-1}, u_i = z_i - 0.9z_i^2 + q\nu\),
Table 3-1. Performance comparison of MxKLMS with KLMS and LMS in nonlinear channel equalization

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>NMSE</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKLMS</td>
<td>$\eta = 1, \mu = 0.8$</td>
<td>0.0984±0.013</td>
<td>18.35±2.61</td>
</tr>
<tr>
<td>KLMS</td>
<td>$\eta = 0.2$</td>
<td>0.129±0.04</td>
<td>15.53±0.88</td>
</tr>
<tr>
<td>LMS</td>
<td>$\eta = 0.01$</td>
<td>0.577±0.16</td>
<td>2.53±1.70</td>
</tr>
</tbody>
</table>

where $q_\nu$ is the white Gaussian noise with variance $\nu^2$ [31], and is selected such that $SNR = 10 \log_{10} \frac{E[x_i^2]}{E[q_i^2]} = 20$dB, where $x_i = z_i - 0.9z_i^2$. Here, $L = 5$, and $\sigma = 1.5$ gives the best performance compared to other values of $\sigma$ [75] and so the other Gaussian kernels with kernel sizes in proximity to this value are selected, i.e., $\sigma = \{0.5, 1, 1.5, 2\}$. The performances of LMS, KLMS with the best kernel and MxKLMS are compared in terms of learning curves and the MSE and prediction Gain. The prediction gain is defined as,

$$G_p = 10 \log_{10} \left( \frac{\sum_{i=\tau_0}^{T} \|y_i\|^2}{\sum_{i=\tau_0}^{T} \|y_i - \hat{y}_i\|^2_2} \right)$$

where, $\hat{y}$ and $y$ are the desired and actual outputs respectively. The learning rates are optimized to get best performance from all three methods and are listed in Table 3-1.

The experiment is performed on 100 different random realizations, each of 1000 sample length, and the average learning curve in Fig. 3-2A is obtained. The evolution of kernel weights across samples are shown in the Fig. 3-2B, which suggest that, the MxKLMS eventually picks two most relevant kernels and gives a better performance than using only the best kernel. Same conclusion can be drawn from the values of MSE and gain in Table. 3-1. Best performance is shown in bold letters.

3.6.2 Non-Stationary Online Signal Prediction

The purpose of this experiment is to demonstrate the ability of MxKLMS to effectively learn the changes in a non-stationary system, by adaptively tuning the kernel weights. Such an ability of creating competition among kernels adds to its robustness compared to the mono-kernel methods, thus, giving a better overall performance. In order to demonstrate such ability of MxKLMS a non-stationary data with two different
Figure 3-2. The figures show A) Learning curves in nonlinear channel equalization averaged over 100 simulations each with random initial condition B) Kernel weights for Gaussian kernels used.
Figure 3-3. The figures show A) non-stationary synthetic input signal B) normalized squared error of prediction for MxKLMS using Gaussian kernels with $\sigma = \{0.1, 0.3, 1\}$ and KLMSs using each of these Gaussian kernels C) Weights for each kernel in MxKLMS across samples. Figure continued.
Figure 3-3 continued.

modes is created using the following function:

\[ 5 \exp(0.5t) |\sin(2\pi ft)| \quad 0 < t < 1600 \]
\[ \exp(\sqrt{t}) |\sin(2\pi ft)| \quad t > 1600 \]

The data is preprocessed by removing the mean and normalizing its amplitude. Here, the past 10 samples are used to predict the next sample. The multiple kernels used are Gaussian kernels with different kernel sizes, \( \sigma = \{0.1, 0.3, 1\} \). Fig. 3-3 shows the input data, the normalized squared error of prediction for MxKLMS and KLMS, and the kernel weights evolution for MxKLMS. Quantitative comparison of these methods based on normalized MSE and gain is presented in Table. 4-1. From these results, it is clear that, MxKLMS can effectively do the prediction even when the system switches between two modes, by selecting the best kernel(s) at each region. For example, KLMS with \( \sigma = 1 \) performs worse than the other two until the 800th sample, after which KLMS with \( \sigma = 0.3 \) and \( \sigma = 1 \) start to perform better. This is reflected in the kernel weights
Table 3-2. Performance comparison of MxKLMS using best kernels, MxKLMS using 40 kernels and KLMS using each of the best kernels, on synthetic non-stationary signal.

<table>
<thead>
<tr>
<th>Method</th>
<th>Gaussian Kernel($\sigma$)</th>
<th>Parameters</th>
<th>NMSE</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKLMS</td>
<td>[0.1, 0.3, 1]</td>
<td>$\eta = 3, \mu = 1$</td>
<td>0.0011</td>
<td>29.40</td>
</tr>
<tr>
<td>MxKLMS</td>
<td>[$10^{-10}, 10^{-9.5}$...$10^{10}$]</td>
<td>$\eta = 10, \mu = 3$</td>
<td>0.0032</td>
<td>25.11</td>
</tr>
<tr>
<td>KLMS</td>
<td>0.1</td>
<td>$\eta = 0.5$</td>
<td>0.0182</td>
<td>17.25</td>
</tr>
<tr>
<td>KLMS</td>
<td>0.3</td>
<td>$\eta = 0.5$</td>
<td>0.0033</td>
<td>24.95</td>
</tr>
<tr>
<td>KLMS</td>
<td>1.0</td>
<td>$\eta = 0.5$</td>
<td>0.0041</td>
<td>23.92</td>
</tr>
</tbody>
</table>

for MxKLMS, where higher weight is given to the better performing kernels. As the performance of KLMS with $\sigma = 0.1$ starts to get better than the other two around 2000<sup>th</sup> sample, the kernel weights start to switch. Thus, MxKLMS is able to select between different kernels in different regions. Similar behavior can be noticed in the regions around 5000<sup>th</sup> and 7000<sup>th</sup> samples.

3.6.3 Model Selection

The softmax function used as a non-linear gate in MxKLMS creates competition among the kernels used, by picking the most relevant kernels from the set of predefined ones at every instance. This inherent feature of MxKLMS makes it appropriate for tasks such as model selection. In this experiment, we demonstrate this property by using a pool of 41 predefined kernels, on the same synthetic dataset used in section 3.6.2. The number of kernels selected depends on the value of $\tau$ as mentioned earlier in section 3.4. The effects of $\tau$ in the number of kernels selected, and thus, in the performance of over all system, are shown in Fig. 3-4A, Fig. 3-4B where, the kernels with weights greater than or equal to $\frac{1}{41}$ are considered as selected. Notice that, MxKLMS with competitive gate (with $\tau = 1$) gives a better performance than using a cooperative gate (with $\tau > 1$). Fig. 3-4 shows the number of models selected across the number of samples for $\tau = 1$. As the system gets stable after around 1000 samples, only two to three kernels are selected, with kernel sizes $\sigma = \{0.1, 0.3, 1\}$, the most dominant being $\sigma = 0.3$. We know that these are the relevant kernels from the performance obtained in section 3.6.2. Moreover, the average performance of the
Figure 3-4. The figures show A) Number of models selected across samples B) Normalized MSE for different values of $\tau$ C) Number of models selected across samples for $\tau = 1$ and $\tau \gg 1$. Figure continued.
Figure 3-4 continued.

system obtained after the system gets stable (1000th sample), in terms of normalized MSE and gain are comparable to that obtained using only the relevant kernels in section 3.6.2. These results are tabulated in Table. 4-1.

3.6.4 Online Prediction of Santa Fe Laser Data

An example of real chaotic data is the laser data from the Santa Fe time series competition [71]. The data represents the output intensity of a Far-Infrared-Laser in a chaotic state and corresponds to a stationary low-dimensional chaotic behavior [21]. In this experiment, we compare the performance of MxKLMS with the other two multi kernel adaptive filtering methods MKNLMS and MKLMS, based on the short term prediction of 1000 sample long laser data, with parameters optimized for the best performance of respective methods. The optimal embedding dimension obtained from
Table 3-3. Performance comparison of MxKLMS, MKNLMS and MKLMS for Santa Fe laser data.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>NMSE</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKLMS</td>
<td>$\eta = 3$, $\mu = 1$</td>
<td>0.067</td>
<td>11.71</td>
</tr>
<tr>
<td>MKNLMS</td>
<td>$\eta = 0.5$</td>
<td>0.278</td>
<td>5.41</td>
</tr>
<tr>
<td>MKLMS</td>
<td>$\eta = 0.3$, $\hat{\eta} = 0.5$</td>
<td>0.127</td>
<td>8.94</td>
</tr>
</tbody>
</table>

 experimentation is $L = 5^1$, i.e., the current value $u_i$ is predicted from the past 5 values $u_i = [u(i - 5), u(i - 4), \ldots, u(i - 1)]^T$. The data is preprocessed by removing mean and normalizing its amplitude, as shown in Fig. 3-5A. The kernel sizes used for Gaussian kernels are selected such that they are close to the optimal kernel size, $\sigma = 0.6$ [75] and are $\sigma = \{0.1, 0.5, 1\}$.

Fig. 3-5B compares the normalized squared error across samples for the three methods, and Fig. 3-5 compares the evolution of their filter coefficients. A clear distinction among these methods can be noticed in terms of their performance. This is due to the competitive gate used in the proposed MxKLMS, compared to the other two methods that do not incorporate such properties. The difference in their working can be noticed in terms of the $n \times P$ filter coefficients. Given that the coefficients are directly proportional to the prediction error, a noticeable change in the coefficients’ amplitudes can be expected at every abrupt signal property change, followed by a gradual stabilization. Clearly, MxKLMS coefficients demonstrate this behavior. However, the coefficients in MKNLMS stabilize very quickly showing minimal learning after 500 samples, even when the signal properties are abruptly changing around 200, 500, 600, 900 samples. The MKLMS coefficients neither stabilize like in MKNLMS nor adhere to the expected behavior but show random fluctuations around the regions of abrupt

\[1\] The embedding dimension for Laser data in [72] has been estimated to be three using entropy based criterion.
Figure 3-5. The figures show A) Laser data after removing mean and normalizing amplitude B) Normalized squared error across samples C) Evolution of filter coefficients across samples. For MKNLMS, the plot beyond 500 samples is magnified in a box within. Figure continued.
signal changes and therefore does better than MKNLMS and worse than MxKLMS. The corresponding normalized MSE and gain are presented in Table. 3-3.

3.6.5 **Online Prediction of Lorenz Chaotic Time Series**

The purpose of this experiment is to compare the performance of MxKLMS with the two existing multi-kernel methods: MKLMS and MKNLMS, in terms of their learning curves, training and test MSE, and training and test prediction gain. By using a chaotic time series such as Lorenz system’s we hope to demonstrate the robust ability of the proposed method in chaotic environment. The Lorenz chaotic system is described by the following differential equations [32]:

\[
\begin{align*}
\frac{dx}{dt} &= \gamma x + yz, \\
\frac{dy}{dt} &= \delta (z - y), \\
\frac{dz}{dt} &= -zy + \rho y - z.
\end{align*}
\]

The parameters \( \gamma = \frac{8}{3}, \delta = 10, \rho = 28 \), are used to obtain sample data using first-order approximation with step size 0.01. We utilize the \( x \)-dimension of the time
Figure 3-6. The figures show A) Lorenz chaotic signal after removing mean and normalizing amplitude B) Ensemble learning curves averaged over 10 trials C) Normalized training MSE across iterations averaged over 10 trials, and smoothed using moving average of 300 sample window (large bump in case of MxKLMS is due to changing states of Lorenz system, smaller bumps can be noticed in case of MKNLMS and MKLMS as well) D) Evolution of kernel weights for MxKLMS across samples. Figure continued.
series, and predict the current value \( u_i \) from past 5 values \( \mathbf{u}_i = [u(i - 5), u(i - 4), ..., u(i - 1)]^T \).

The signal is further preprocessed by removing mean and normalizing its amplitude to obtain input signal shown in Fig. 3-6A. Same set of Gaussian kernels with kernel sizes
Table 3-4. Performance comparison of MxKLMS, MKNLMS, MKLMS for short term prediction of Lorenz chaotic signal, obtained after averaging over 10 different trials.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Training MSE</th>
<th>Test MSE</th>
<th>Training Gain</th>
<th>Test Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKLMS</td>
<td>$\mu = 1, \eta = 1$</td>
<td>0.001±0.001</td>
<td>0.013±0.017</td>
<td>19.87±0.17</td>
<td>27.01±5.85</td>
</tr>
<tr>
<td>MKNLMS</td>
<td>$\rho = 10^{-4}, \eta = 0.1$</td>
<td>0.040±0.006</td>
<td>0.127±0.147</td>
<td>10.47±0.35</td>
<td>14.73±2.89</td>
</tr>
<tr>
<td>MKLMS</td>
<td>$\tilde{\eta} = 0.05, \eta = 0.08$</td>
<td>0.016±0.001</td>
<td>0.387±0.823</td>
<td>13.94±0.39</td>
<td>13.13±3.01</td>
</tr>
</tbody>
</table>

$\sigma \in \{0.01, 0.1, 0.5, 1\}$ are used for all the four methods. For fair comparison with MxKLMS, the MKLMS and MKNLMS have been implemented without the sparsification criteria that they have been presented with in [67] and [75], respectively. That is, the threshold values responsible for sparsification have been set to a very low value so that all the incoming centers are stored. The learning rates for each of these algorithms are selected so as to get the minimum prediction error, and are listed in Table 4-2.

The results of prediction are generated by averaging over 10 independent simulations, each with randomly picked segments of 3500 sample length. The final normalized test MSE is calculated as an average over last 1000 samples in the average ensemble learning curve for 10 simulations as shown in Fig. 3-6B. Each ensemble learning curve is obtained by plotting ensemble MSE on test data of 100 sample length at each training instant (keeping the filter fixed during testing phase). Also, the final normalized training MSE is calculated as an average over last 1000 samples in the average normalized training MSE curves for 10 simulations as shown in Fig. 3-6C.

The three methods are compared in Fig. 3-6B and Fig. 3-6C in terms of the MSE curves. Quantitative comparison is presented in Table 4-2 in terms of training and test MSE and prediction gain. It is clear that MxKLMS outperforms MKNKLMS and MKLMS in terms of normalized MSE and gain.

### 3.6.6 Long-Term Time Series Prediction

We evaluate the long term prediction abilities of the multikernel adaptive filtering algorithms with the real world wind data: a bivariate time series describing the wind velocity in the north-south ($V_N$) and east-west ($V_E$) directions, that is, $S$ =
Each time series belongs to one of following three regimes, low, medium, and high velocities. It has been recorded with a 2-D ultrasonic anemometer at a sampling frequency of 50Hz.  

For this experiment, a synthetic signal is compiled from the real world observations such that the complete signal is 1500 samples long, and it consists of three equal length consecutive segments belonging to one of the three regimes. The purpose of such arrangement is to create non-stationarity in signal and to compare the abilities of the algorithms to cope with abrupt changes. All three methods use same set of Gaussian kernels with kernel sizes \( \sigma \in \{0.2, 0.8, 2\} \), and are evaluated for 10-step ahead prediction of the wind speed from 20 dimensional input vector, for each signal component. That is, \( u_D(i) = [u_D(i - 30), u_D(i - 29), \ldots, u_D(i - 11)]^\top \), \( u_D \in \{V_N, V_E\} \), is used to predict current value \( u_D(i) \). The learning rates used for these methods are listed in Table. 3-5. Fig. 3-7 shows the prediction of the bivariate wind signal, compared to the original signals, along with the mean square error of prediction across samples, averaged over 10 different trials. The prediction performance of these methods can also be assessed using the prediction gain defined as,

\[
G_p = 10 \log_{10}\left(\frac{\sum_{i=T_0}^T \|d_i\|^2}{\sum_{i=T_0}^T \|d_i - y_i\|^2}\right)
\]

where \( d \) and \( y \) are the desired and actual outputs respectively, and the mean square error of prediction. Table 3-5 presents the prediction gain averaged over 10 different trials for the three methods. The average MSE and gain make it clear that these methods show a comparable performance in this dataset.

\footnote{The data are publicly available at http://www.commsp.ee.ic.ac.uk/mandic/research/wind.htm}
Figure 3-7. The figures show A) the predicted time series, and B) the resulting MSE of prediction, for three methods MxKLMS, MKNLMS and MKLMS in the task of long-term prediction of wind data.
Table 3-5. The table shows the parameter values used in the three methods for both tasks, along with the resulting prediction gains and normalized mean square errors averaged over 10 different trials.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>MSE</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKLMS</td>
<td>$\mu = 0.2, \eta = 2$</td>
<td>0.25±0.03</td>
<td>7.507±0.93</td>
</tr>
<tr>
<td>MKNLMS</td>
<td>$\rho = 0.01, \eta = 0.9$</td>
<td>0.27±0.05</td>
<td>6.72±1.14</td>
</tr>
<tr>
<td>MKLMS</td>
<td>$\hat{\eta} = 0.5, \eta = 0.2$</td>
<td>0.27±0.05</td>
<td>6.57±1.11</td>
</tr>
</tbody>
</table>

3.7 Summary

In this chapter, we proposed the MxKLMS algorithm which is simply a combination of stochastic gradient approximations to LS problems intrinsically solved in multiple RKHSs simultaneously. The final solution exists in the sum of RKHSs and can be represented as the weighted sum of inner products in each space, giving the final multi-kernel adaptive filtering formulation in (3–26). Hence, theoretical strength of MxKLMS stems from the fact that we are able to establish the multi-kernel formulation as an inherent optimization in RKHSs. Moreover, this paper presents multi-kernel formulation as an alternative to the multiple kernel learning methods established for batch methods, pointing out that those methods cannot be applied to the sequential learning methods.

We have presented MxKLMS as a novel multikernel adaptive filtering method, by explaining it in terms of the sum of spaces framework and the competitive gated model framework. The sum of spaces framework establishes MxKLMS as an inherent optimization in RKHSs, where as the gated competitive model framework establishes it as a competitive combination of multiple KLMSs.

As a significant theoretical contribution in the paper, we were able to establish the sufficient criteria for mean square convergence of the MxKLMS filter. In addition to that, we also performed steady state analysis of the filter to obtain an upper bound on the EMSE of filter that was equal to the EMSE of KLMS. This suggests that the MxKLMS has an ability to perform better than KLMS, which is also supported by the empirical results presented.
We showed empirically that MxKLMS is not only able to select the most appropriate kernels from a large pool of predefined kernels but can also select different kernels for different regions of the input data, which is a possible scenario in case of non-stationary data. Moreover, we showed theoretically and empirically that regardless the initial pool of kernels provided, MxKLMS is able to reach the best solution. Finally, by using multiple kernels and competitive gating, we show that MxKLMS is able to outperform the KLMS as well as the existing multi-kernel methods MKLMS and MKNLMS, even in the highly chaotic Lorenz and Santa Fe laser datasets.

In conclusion, this chapter has presented a novel multi-kernel adaptive filter, that has theoretical justification to its formulation and its working. This method stands out as a good method for solving the kernel size selection problem by not just efficiently selecting the appropriate kernels for the problem at hand, but also showing a significant improvement in the overall performance.
CHAPTER 4
QUANTIZED MIXTURE KERNEL LEAST MEAN SQUARE

The mixture kernel least mean square (MxKLMS) successfully addresses the kernel selection problem by implementing multiple kernels in the mono-kernel KLMS thus, improving the performance. However, like any other existing multi-kernel adaptive filters, it suffers from a problem of linearly growing dictionary like in KLMS. In this chapter, we present the quantized MxKLMS (QMxKLMS) algorithm to achieve sub linear dictionary growth. This method quantizes the input space based on the conventional criteria using Euclidean distance in input space as well as a new criteria using Euclidean distance in RKHS induced by the sum kernel. The empirical results suggest that QMxKLMS using the latter metric is suitable in a non-stationary environment with abruptly changing modes as it is able to utilize the information regarding the relative importance of kernels. Moreover, the QMxKLMS using both metrics are compared with the QKLMS and the existing multi-kernel methods MKLMS and MKNLMS-CS, showing an improved performance over these methods.

4.1 Background

In addition to tackling the issue of kernel selection, the use of multiple kernels with KAF have shown fast convergence compared to the mono-kernel methods along with an ability to perform better in non-stationary environment. Given the various advantages of using multiple kernels, this also adds a memory and computational burden. As it is well known that the dictionary size of KLMS grows linearly with the incoming samples, it is evident that the computational burden as well as memory requirement increases with the use of multiple kernels. There have been several work in the kernel adaptive filtering (KAF) literature to constrain the growth of the dictionary in KLMS. Some of them include, the use of novelty criterion (NC) [39] in [45] to check if the newly arrived datum is informative enough. Similar method called coherence criterion (CC) [49] has also been explored in [45]. The idea of approximate
linear dependency test (ALD) [13] was used with KLMS in [42]. Recently, the idea of quantizing the input space by representing a certain location with a fewer samples, was used with KLMS in [11], showing a significantly improved performance over the NC and CC. Some of these methods including NC and CC have also been used to constrain the linear dictionary growth in multiple kernel adaptive filtering, like in [67] and [75], turning it into a sub linear growth. However, when it comes to the multiple kernels, a criterion based on a metric defined in the input space might not always be justifiable. This is because, such a metric would disregard the information pertaining to the relative distances of the data in different spaces projected by different kernels. And since multiple kernel adaptive filtering weighs a pool of predefined kernels, it would therefore, be more sensible to build a criterion that gives a decision based on a metric defined in different spaces along with the the information about the importance of respective kernels. Keeping this in mind, we have proposed a new criterion suitable for MxKLMS, that implements the quantization method for KLMS [11]. This criteria uses the Euclidean distance in the RKHS induced by sum of kernels, which in fact, can also be interpreted as the weighted sum of distances in the multiple RKHSs. Since the Euclidean distance in an RKHS corresponds to the correntropy metric in the input space [30, 57], this procedure effectively is equivalent to a correntropy metric based in the sum space. We also implement quantization criteria for MxKLMS by computing euclidean distances in the input space like in the original method [11] and compare their performances for stationary and non-stationary data.

4.2 Quantized KLMS: Overview

Let, \( u_n \in U \subset \mathbb{R}^l \) be an input vector at time \( n \), and \( y_n \in Y \subset \mathbb{R} \) be the desired response, which is a non-linear function of input \( u_n \). The goal is to learn a continuous input-output mapping \( f : U \rightarrow Y \), based on the incoming input-output pair \( \{u_i, y_i\}_{i=1}^N \) in the reproducing kernel Hilbert space (RKHS) \( \mathcal{F} \), induced by the positive-definite kernel \( \kappa : U \times U \rightarrow \mathbb{R} \) [3].
The QKLMS algorithm can then be obtained by simply quantizing the feature vector \( \varphi(u_n) \) in the weight update equation (2–13) such that,

\[
\Omega_n = \Omega_{n-1} + \eta e_n \varphi Q[(u_n)]
\]  

(4–1)

where, \( Q[.] \) denotes a quantization operator in \( F \) [11]. However, since the feature space is very high dimensional and possibly infinite dimensional, the quantization is performed in the input space \( \mathbb{U} \) so that the learning rule for QKLMS is,

\[
\begin{align*}
    f_0 & = 0 \\
    e_n & = y_n - f_{n-1}(u_n) \\
    f_n & = f_{n-1} + \eta e_n \kappa(Q[u_n], \cdot)
\end{align*}
\]

(4–2)

where, \( Q[\cdot] \) is a quantization operator in \( \mathbb{U} \). Now, an online VQ method is implemented to quantize the input space \( \mathbb{U} \) in which the dictionary is obtained directly from online samples by computing Euclidean distance in \( \mathbb{U} \), given by,

\[
\|u_i - u_j\|_U = \left\{ (u_i - u_j)^T (u_i - u_j) \right\}^{\frac{1}{2}}
\]

(4–3)

Here, the quantization criteria is based on the Euclidean distance in \( \mathbb{U} \) and not in the RKHS \( F \), since the later is the monotonically increasing function of the former, given a Gaussian kernel, and hence, the quantization obtained in both the cases would be very similar.

### 4.3 Quantized MxKLMS

The quantized MxKLMS (QMxKLMS) algorithm can be obtained by quantization of the feature vector \( \varphi_k(u_n) \) in the weight-update equation \( \Omega_{k,n} = \Omega_{k,n-1} + \eta e_n \beta_k \varphi_k(u_n), \forall k \in \{1, \ldots, P\} \), in (3–17) which can be expressed as
\[
\begin{align*}
\Omega_{k,0} &= 0, e_n = y_n - \sum_{m=1}^{P} \beta_m (\Omega_{m,n-1} \varphi_m(u_n)) \\
\Omega_{k,n} &= \Omega_{k,n-1} + \eta e_n \beta_k Q[\varphi_k(u_n)] \\
\Omega_n &= \sum_{m=1}^{P} \beta_m \alpha_{m,i} \kappa_m (Q[u_n], \cdot)
\end{align*}
\]

where \(Q[\cdot]\) is a quantization operator in \(F_{\kappa_m}\). Also, \(\beta_m\) is learned along with \(\Omega_m\), for which the learning rule is same as described in the section 3.4. Following the same reasoning as in section 4.2, the quantization is performed in the input space \(U\) such that the learning rule for QMxKLMS is

\[
\begin{align*}
f_{k,0} &= 0, e_n = y_n - f_{k,n-1}(u_n) \\
f_{k,n} &= f_{k,n-1} + \eta e_n \beta_k \kappa_k (Q[u_n], \cdot) \\
f_n &= \sum_{m=1}^{P} \beta_m \alpha_{m,i} \kappa_m (Q[u_n], \cdot)
\end{align*}
\]

where, \(Q[\cdot]\) is a quantization operator in \(U\). In the rest of the paper, the notations are simplified using \(\varphi_{m,n} = \varphi(u_n)\), \(\varphi_{m,n}^q = Q[\varphi(u_n)]\), and \(u_n^q = Q[u_n]\).

The online vector quantization (VQ) method required to quantize the input space \(U\), presented in [11] was based on building the dictionary directly from online samples by computing Euclidean distance in \(U\). However, when we have combination of multiple kernels incorporating certain weights \(\beta\), quantization based on the distance in the corresponding sum space might not always be same as one based on the distance in single kernel, depending upon how the \(\beta\) changes over time. Intuitively, it makes sense to consider the relevance of a kernel (given by \(\beta\)) along with the distance measured in its induced RKHS. Therefore, we propose to compute the Euclidean distance in the sum space \(F\) at every incoming sample, given by,

\[
\|\varphi_i - \varphi_j\|_F = \left\{ (\varphi_i - \varphi_j)^T (\varphi_i - \varphi_j) \right\}^{\frac{1}{2}} = \sqrt{2 - 2\kappa(u_i, u_j)}
\]
Algorithm 2 Online VQ in $\mathbb{U}$

**Input:** $\{u_i \in \mathbb{U}\} \forall i \in \{1, \ldots, n\}$

**Initialization:** Choose quantization size $\delta \geq 0$, and initialize dictionary $D_1 = \{u_1\}$

**Computation:**

while $\{u_i\}$ ($i > 1$) available do

1. Compute the distance between $u_i$ and $D_{i-1}$:
   $$dis(u_i, D_{i-1}) = \min_{1 \leq j \leq \text{size}(D_{i-1})} dis(u_i, u_{j, i-1})$$

2. If $dis(u_i, D_{i-1}) \leq \delta$, keep the dictionary unchanged:
   $$D_i = D_{i-1}, \quad \text{and quantize } u_i \text{ to the closest dictionary element vector } u_i^q = D_{j^*, i-1}$$
   where,
   $$j^* = \arg \min_{1 \leq j \leq \text{size}(D_{i-1})} dis(u_i, u_{j, i-1})$$

3. Else, update the dictionary:
   $$D_i = \{D_{i-1}, u_i\}, \quad \text{and quantize } u_i \text{ as itself: } u_i^q = u_i$$

end while

where, $\varphi : \mathbb{U} \to \mathbb{F}$ be the corresponding mapping function. We can notice that (4–6) is in fact, the correntropy induced metric [30, 57] defined in the RKHS $\mathbb{F}$.

Among the various possible ways in which $\kappa$ can be decomposed as the weighted sum of $\kappa_m$, where the weights are convex [3, 56], we choose to represent $\kappa$ as $\sum_m \beta_m \kappa_m$ in order to be consistent with the functional decomposition $f = \sum_{m=1}^P \beta_m f_m$. Moreover, this allows us to use the relative importance of spaces given directly by $\beta$. Therefore, the distance in $\mathbb{F}$ can be written as,

$$\|\varphi_i - \varphi_j\|_x = \sqrt{2 \left( 1 - \sum_{m=1}^P \beta_m \kappa_m(u_i, u_j) \right)}$$

$$= \sqrt{\sum_{m=1}^P \beta_m \left( 1 - \exp \left( -\frac{\|u_i - u_j\|^2}{2\sigma_m^2} \right) \right)}$$

(4–7)

where, the constant $\sqrt{2}$ is ignored. Clearly, the distance in $\mathbb{F}$ can be represented as the weighted sum of distances in $\mathbb{F}_m$. Now, a general online VQ method is presented in
Algorithm 2, that can perform online quantization of input space based on the distance in $\mathbb{U}$ or $\mathbb{F}$. In the algorithm, $D_{j,i-1}$ denotes the $j^{th}$ element of the dictionary $D$ at $(i-1)^{th}$ iteration and $\text{dis}(.)$ operator can be the Euclidean distance in $\mathbb{U}$ or $\mathbb{F}$. The Euclidean distance in $\mathbb{U}$ is given by (4–3) and in $\mathbb{F}$ is given by (4–7). Clearly, the distance in $\mathbb{F}$ can be represented as the weighted sum of distances in $\mathbb{F}_m$.

It is true that (4–7) is also a monotonically increasing function of distance in the input space $\mathbb{U}$ and using either might not make a difference in quantization when it comes to static signal leading to static values of $\beta$. But, if there is variability in the values of $\beta_m$, for example, in case of abruptly changing data, where the relative importance of kernels would change with time, this distance will make a difference which we will see later in the section 4.4. In this paper, we present the QMxKLMS results based on the euclidean distances in both $\mathbb{U}$ and $\mathbb{F}$ and call them QMxKLMS-Eu and QMxKLMS-KEu respectively.

**4.4 Experiments and Results**

In this section, we present experiments to demonstrate the working and robustness of QMxKLMS based on two datasets: synthetic non-stationary data and Lorenz chaotic series. We will also compare the method with the two existing multi-kernel adaptive filtering techniques MKLMS and MKNLMS, that implement NC and CC respectively, to constrain the dictionary growth.

**4.4.1 Non-Stationary Signal Prediction**

The purpose of this experiment is to demonstrate the difference in the working of QMxKLMS-Eu and QMxKLMS-KEu in learning the non-stationary system with a constrained dictionary. The synthetic data used for this experiment is same as described in section 3.6.2, and is shown in Fig. 3-3A. With this setup, we hope to attain variability in the values of $\beta$ due to the changing modes and thus change in the relative importance of kernels over time, as mentioned in the previous section. This setup is also useful in testing the ability of QMxKLMS to effectively learn the changes in a
Algorithm 3 Quantized MxKLMS Algorithm

Input:
\{u_i \in U, y_i\} \forall i \in \{1, \cdots, n\}
P Kernels

Initialization:
\widehat{y}_1 = 0,
Gate parameter \nu_{m,1} = \frac{1}{P}, \beta_{m,1} = \frac{1}{P} \forall m \in \{1, \cdots, P\},
Learning rates: \eta, \mu > 0,
Quantization threshold \delta \geq 0,
Initialize dictionary \textbf{D}_1 = \{u_1\}

Computation:
while \{u_i\} (i > 1) available do
1. Evaluate: \widehat{y}_i = \sum_{m=1}^{P} \sum_{j=1}^{\text{size} (\textbf{D}_{i-1})} \beta_{m,i} \alpha_{m,i} \kappa_m (\textbf{D}_{j,i-1}, u_i)
2. Compute error: \(e_i = y_i - \widehat{y}_i\)
3. Update gate parameter: \(\nu_{m,i} = \nu_{m,i-1} - \mu \nabla \nu_m\)
4. Compute \(\beta_{m,i}\)
5. Compute the distance between \(u_i\) and \textbf{D}_{i-1}:
\(\text{dis}(u_i, \textbf{D}_{i-1}) = \min_{1 \leq j \leq \text{size}(\textbf{D}_{i-1})} \text{dis}(u_i, \textbf{D}_{j,i-1})\)
6. If \(\text{dis}(u_i, \textbf{D}_{i-1}) \leq \delta\), keep the dictionary unchanged: \(\textbf{D}_i = \textbf{D}_{i-1}\), and quantize \(u_i\) to the closest dictionary element vector through updating the coefficient of that vector:
\(\alpha_{m,i} = \alpha_{m,i-1} + \eta \beta_{m,i} e_i\) where, \(j^* = \arg \min_{1 \leq j \leq \text{size}(\textbf{D}_{i-1})} \text{dis}(u_i, \textbf{D}_{j,i-1})\)
7. Else, add a new center and corresponding new coefficient:
\(\textbf{D}_i = \{\textbf{D}_{i-1}, u_i\}\), and \(\alpha_{m,i} = [\alpha_{m,i-1}, \eta \beta_{m,i}]\)
end while

non-stationary system, by competitively combining the kernel weights. Such a unique ability of creating competition among kernels adds to its robustness compared to the mono-kernel methods, thus, giving a better overall performance.

The experiment is such that, the past 10 samples are used to predict the next sample. The multiple kernels used are Gaussian kernels with different kernel sizes,

65
Figure 4-1. The figure shows the effect of quantization threshold ($\delta$) on the final normalized MSE and final network size on synthetic non-stationary A) QMxKLMS-Eu B) QMxKLMS-KEu.
Table 4-1. Comparison of QMxKLMS-Eu, QMxKLMS-KEu, MxKLMS and KLMS on prediction of synthetic non-stationary signal, to obtain (I)smallest network size (centers) (II)minimum NMSE.

<table>
<thead>
<tr>
<th>Method</th>
<th>Centers</th>
<th>NMSE</th>
<th>Gain</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMxKLMS-Eu</td>
<td>9</td>
<td>1.082±3.25</td>
<td>-0.34</td>
<td>η = 2, µ = 0.5, δ = 0.73</td>
</tr>
<tr>
<td>QMxKLMS-KEu</td>
<td>7</td>
<td>0.071±0.24</td>
<td>11.15</td>
<td>η = 2, µ = 0.5, δ = 0.81</td>
</tr>
<tr>
<td>QKLMS, σ = 0.1</td>
<td>9</td>
<td>2.195±7.78</td>
<td>-3.45</td>
<td>δ = 0.73, η = 0.5</td>
</tr>
<tr>
<td>QKLMS, σ = 1</td>
<td>9</td>
<td>0.105±0.26</td>
<td>9.76</td>
<td>δ = 0.73, η = 0.5</td>
</tr>
<tr>
<td>QMxKLMS-Eu</td>
<td>1660</td>
<td>0.001±0.004</td>
<td>29.30</td>
<td>η = 2, µ = 0.5, δ = 0.001</td>
</tr>
<tr>
<td>QMxKLMS-KEu</td>
<td>1285</td>
<td>0.001±0.004</td>
<td>29.30</td>
<td>η = 2, µ = 0.5, δ = 0.25</td>
</tr>
<tr>
<td>QKLMS, σ = 0.1</td>
<td>1650</td>
<td>0.020±0.059</td>
<td>16.86</td>
<td>δ = 0.001, η = 0.5</td>
</tr>
<tr>
<td>QKLMS, σ = 1</td>
<td>1650</td>
<td>0.004±0.01</td>
<td>23.82</td>
<td>δ = 0.001, η = 0.5</td>
</tr>
<tr>
<td>MxKLMS</td>
<td>8000</td>
<td>0.001±0.0004</td>
<td>29.61</td>
<td>η = 2, µ = 0.5</td>
</tr>
</tbody>
</table>

σ = {0.1, 1}. First, the effect of quantization threshold (δ) is evaluated on the normalized MSE (NMSE) and the network size at the final stage of adaptation for QMxKLMS-Eu and QMxKLMS-KEu. These effects can be seen in Fig. 4-1. As expected, the final NMSE increases with increasing value of δ, and thus, the decreasing final network size for both the methods. However, in case of QMxKLMS-KEu, the final NMSE obtained even with minimum network size is much smaller than in case of QMxKLMS-Eu. Therefore, for a fair comparison, the two methods have been evaluated for two cases: (I) small network size (II) minimum NMSE.

The result of prediction for case (I) and case (II) are presented in Fig. 4-2. The figure compares the network size evolution as well as the normalized squared error across samples for QMxKLMS-Eu and QMxKLMS-KEu. The results have been quantified in Table. 4-1 for both the cases (I) and (II) in terms of mean NMSE, prediction gain and the final network size. The prediction gain is defined as,

\[ G_p = 10 \log_{10} \left( \frac{\sum_{i=-T_0}^{T_0} ||y_i||^2}{\sum_{i=-T_0}^{T_0} ||y_i - \hat{y}_i||^2} \right). \]

In case (I), QMxKLMS-Eu is clearly unable to learn, while QMxKLMS-KEu still gives a reasonable performance with only 7 samples. Where as in case (II), both the methods are able to attain same minimum NMSE but QMxKLMS-KEu still attains smaller network size than QMxKLMS-Eu.
Figure 4-2. The figures compare the performances of QMxKLMS-Eu, QMxKLMS-KEu and QKLMS ($\sigma = 0.1, 1$) on prediction of synthetic non-stationary signal in terms of A) Network size evolution across samples (QMxKLMS-Eu and QKLMS ($\sigma = 0.1, 1$) overlap) B) normalized squared error across samples to obtain smallest network size and C) Network size evolution across samples, QMxKLMS-Eu and QKLMS ($\sigma = 0.1, 1$) overlap D) normalized squared error across samples, to obtain minimum NMSE. Figure continued.
Figure 4-2 continued.

We can notice that both QMxKLMS are able to attain the same NMSE as MxKLMS with a network size that is as small as \( \frac{1}{6} \) the total, without any compromise in the performance, showing a significant advantage over MxKLMS. Moreover, even with reduced network size QMxKLMS-Eu and QMxKLMS-KEu depict similar kernel weight
Figure 4-3. QMxKLMS Kernel weight evolution across samples for synthetic non-stationary signal.

Evolution as in MxKLMS as shown in Fig. 4-3. It is also evident from the results that QMxKLMS is able to show effective prediction performance, better than mono kernel QMxKLMS, even when the system switches between two modes, by selecting the best kernel at each region.

In Fig. 4.4.1, QKLMS with $\sigma = 0.1$ gives less prediction error until the 800th sample, compared to QKLMS with $\sigma = 1$. This is reflected in the kernel weights for QMxKLMS and MxKLMS in Fig. 4-3, where emphasis is given to the kernel with $\sigma = 0.1$ compared to kernel with $\sigma = 1$. As the performance of QKLMS with $\sigma = 0.1$ starts to degrade compared to $\sigma = 1$, the kernel weights start to switch. Thus, QMxKLMS, like MxKLMS, is able to select different kernels for different regions. Similar behavior can be noticed in the regions around 2000, 4000, 7000 samples.

4.4.2 Short-term Prediction of Lorenz Chaotic Time Series

This experiment compares the QMxKLMS-Eu and QMxKLMS-KEu with the two existing multi-kernel methods: MKLMS and MKNLMS-CS for short-term prediction of
Lorenz chaotic time series, in terms of the convergence speed, accuracy of prediction and their network size evaluated for the two cases: (I) small network size (II) minimum NMSE. The sample data for Lorenz chaotic system was generated as in section 3.6.5, with same preprocessing to obtain input data shown in Fig. 3-6A. We utilize the $x$-dimension of the time series, and predict the current value $u(i)$ from past 5 values $\mathbf{u}_i = [u(i - 5), u(i - 4), ..., u(i - 1)]^T$.

Same set of Gaussian kernels with kernel sizes $\sigma \in \{0.01, 0.1, 0.5, 1\}$ are used for all methods. The MKLMS and MKNLMS-CS have been implemented with the NC and CC sparsification criteria that they have been presented with in [67] and [75], respectively. The sparsification parameters for these methods include the input distance threshold $\delta$ and the error threshold $\delta_e$ for MKLMS and the input distance threshold $\delta$ for MKNLMS-CS. These parameters, along with the threshold parameter $\delta$ for both QMxKLMS were assigned different values to obtain the two cases (I) and (II). The other parameters including, learning rates $\eta$, $\hat{\eta}$ and the regularization parameter $\rho$ for MKLMS, the learning rate $\eta$ for MKNLMS-CS and the learning rates $\eta$ and $\mu$ for QMxKLMS, MxKLMS, were fixed for both the cases. The parameters are listed in Table. 4-2.

The results of prediction are generated by averaging over 10 independent simulations, each using different input segments of 3500 sample length. Fig. 4-5 compares the average network size, average test and training normalized MSE (NMSE) of QMxKLMS-Eu, QMxKLMS-KEu, MKLMS and MKNLMS-CS across training samples, for the case (I). Each ensemble learning curve is obtained by plotting the ensemble MSE on test data of 100 sample length (keeping the filter fixed during testing phase) at each training instance. Notice that, for a network size as small as 25 centers, the test NMSE for the four methods are similar although they are much less compared to that of MxKLMS which uses all the available centers. Similarly, Fig. 4-6 compares the average network size and average test and training normalized MSE (NMSE) of QMxKLMS-Eu, QMxKLMS-KEu, MKLMS and MKNLMS-CS across training samples, for the case (II).
Figure 4-4. The figure shows the effect of quantization threshold ($\delta$) on the final normalized MSE and final network size on Lorenz chaotic time series A) QMxKLMS-Eu B) QMxKLMS-KEu.
Figure 4-5. The figures compare performances of QMxKLMS-Eu, QMxKLMS-KEu, MKNLMS-CS, and MKNLMS on short term prediction of Lorenz chaotic time series based on the A) mean network size B) mean training NMSE C) mean test NMSE (all methods except MxKLMS overlap), obtained for case (I) small network size. Figure continued.
Notice that, MKNLMS-CS gives minimum test NMSE when the network size is as small as 25 centers, as in case (I) but the other methods require around 250 samples to give their best performances in terms of test NMSE. However, the best performances of QMxKLMS-Eu and QMxKLMS-KEu out perform the best performances of MKLMS and MKNLMS. In fact, QMxKLMS-Eu and QMxKLMS-KEu are able to give as good a performance as MxKLMS that uses all the centers.

These results are quantified in Table. 4-2 in terms of final network sizes, final test and training NMSE and the final test gain for both the cases (I) and (II). The final network size is calculated as an average of final network size obtained at the end of training for 10 different simulations. The final normalized test MSE is calculated as an average over last 1000 samples in the average ensemble learning curves for 10 simulations. Also, the final normalized training MSE is calculated as an average over last 1000 samples in the average normalized training MSE curves for 10 different simulations. Finally, the test gain is obtained as an average prediction gain of test data.
Figure 4-6. The figures compare performances of QMxKLMS-Eu, QMxKLMS-KEu, MKNLMS-CS, and MKNLMS on short term prediction of Lorenz chaotic time series based on the A) mean network size B) mean training NMSE C) mean test NMSE (MxKLMS, QMxKLMS-Eu and QMxKLMS-KEu overlap), obtained for case (II) minimum test NMSE. Figure continued.
Figure 4-6 continued.

from 10 different simulations. As we can see, QMxKLMS-KEu and QMxKLMS-Eu are able to outperform all the other methods. Also, it should be noted that, both QMxKLMS are able to attain as good a performance as MxKLMS with as less as close to 250 centers. Hence, we are able to cut down the memory requirement and computational complexity by a very large amount without compromising in the performance of the system.

4.5 Summary

In this paper, we presented MxKLMS with a method to obtain sub linear growth in dictionary by quantizing the input space, and called it quantized MxKLMS (QMxKLMS). We quantized the input space based on the conventional criteria using the Euclidean distance in the input space, as well as using a new criteria based on the Euclidean distance in the final sum space and called them QMxKLMS-Eu and QMxKLMS-KEu respectively. We showed that QMxKLMS-KEu is able to incorporate the distances in different spaces based on their relative importance given by the $\beta$. We demonstrated
Table 4-2. Comparison of QMxKLMS-Eu, QMxKLMS-KEu, MxKLMS, MKNLMS-CS, MKLMS, and QKLMS on prediction of Lorenz chaotic time series to obtain (I) smallest network size (II) minimum MSE.

<table>
<thead>
<tr>
<th>Case</th>
<th>Method</th>
<th>Network Size</th>
<th>Training MSE</th>
<th>Test MSE</th>
<th>Test Gain</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>QMxKLMS-Eu</td>
<td>23.14±1.79</td>
<td>0.007±0.003</td>
<td>0.124±0.102</td>
<td>14.75±5.89</td>
<td>𝜷 = 1, 𝜹 = 1</td>
</tr>
<tr>
<td></td>
<td>QMxKLMS-KEu</td>
<td>23.31±2.54</td>
<td>0.008±0.004</td>
<td>0.150±0.144</td>
<td>14.82±7.55</td>
<td>𝜷 = 1, 𝜹 = 1</td>
</tr>
<tr>
<td></td>
<td>MKLMS</td>
<td>25.90±1.66</td>
<td>0.014±0.006</td>
<td>0.126±0.122</td>
<td>14.99±4.21</td>
<td>𝜷 = 0.05, 𝜱 = 0.08</td>
</tr>
<tr>
<td></td>
<td>MKNLMS-CS</td>
<td>22.05±1.63</td>
<td>0.033±0.004</td>
<td>0.110±0.082</td>
<td>13.79±2.61</td>
<td>𝜷 = 10⁻⁴, 𝑘 = 0.1</td>
</tr>
<tr>
<td>II</td>
<td>QMxKLMS-Eu</td>
<td>239.10±3.98</td>
<td>0.001±0.001</td>
<td>0.013±0.011</td>
<td>25.38±3.44</td>
<td>𝜷 = 1, 𝜹 = 1</td>
</tr>
<tr>
<td></td>
<td>QMxKLMS-KEu</td>
<td>253.10±20.34</td>
<td>0.001±0.001</td>
<td>0.013±0.009</td>
<td>25.27±3.48</td>
<td>𝜷 = 1, 𝜹 = 0.0031</td>
</tr>
<tr>
<td></td>
<td>MKLMS</td>
<td>264.9±17.85</td>
<td>0.006±0.004</td>
<td>0.044±0.057</td>
<td>22.43±4.99</td>
<td>𝜷 = 0.05, 𝑘 = 0.08</td>
</tr>
<tr>
<td></td>
<td>MKNLMS-CS</td>
<td>22.05±1.63</td>
<td>0.033±0.004</td>
<td>0.110±0.082</td>
<td>13.79±2.61</td>
<td>𝜷 = 10⁻⁴, 𝑘 = 0.1</td>
</tr>
<tr>
<td></td>
<td>MxKLMS</td>
<td>3500</td>
<td>0.001±0.001</td>
<td>0.012±0.01</td>
<td>25.47±3.51</td>
<td>𝜷 = 1, 𝜹 = 1</td>
</tr>
</tbody>
</table>
its usefulness over QMxKLMS-Eu in a non-stationary environment containing rapidly switching modes. In addition, we also demonstrated that both QMxKLMS-Eu and QMxKLMS-KEu perform equally well compared to each other and much better compared to the existing multi-kernel adaptive filtering methods MKLMS and MKNLMS-CS based in the short term prediction of Lorenz chaotic time series. Therefore, QMxKLMS over all, is a much efficient method for addressing the issue of kernel selection and improving the performance of a system. Moreover, when it comes to non-stationary system, QMxKLMS-KEu is more suitable than QMxKLMS-Eu. The convergence behavior of QMxKLMS and its steady state analysis and tracking properties can be possible future avenues for research.
5.1 Background

5.1.1 Maximum Correntropy Criteria

MCC is a derivative of correntropy, that has been effectively implemented as a robust cost for linear adaptive filtering in [60]. Here, the interest is to maximize the similarity between the desired response \( d_n \) and the predicted response \( \hat{y}_n \) in correntropy sense:

\[
\hat{\nu} = \frac{1}{N} \sum_{i=1}^{N} \kappa_{\sigma_c}(e_i) = \frac{1}{N} \sum_{i=1}^{N} \exp \left( -\frac{e_i^2}{2\sigma_c^2} \right)
\]  

(5–1)

where, \( e_i = d_i - \hat{y}_i \). The quantity in (5–1) is the expected risk and gives an estimate of error pdf evaluated at 0 from the Parzen density estimation principle. Therefore, maximizing the correntropy of error maximizes the error pdf at 0. This can be achieved by the gradient update rule on the cost:

\[
J_c = \kappa_{\sigma_c}(e_i) = \exp \left( -\frac{e_i^2}{2\sigma_c^2} \right)
\]  

(5–2)

5.1.2 Generalized Convexity of Correntropy Cost Function

Definition: Convex: A twice differentiable function \( f : S \to \mathbb{R} \), where \( S \) is a non-empty subset of \( \mathbb{R}^n \), is said to be convex if and only if the Hessian matrix of \( f \) is positive semidefinite at each point in \( S \). Alternatively, the function \( f \) is said to be convex over \( S \) if and only if:

\[
f(x_2) \geq f(x_1) + (x_1 - x_2)^{\top} \nabla f(x_1) \quad \forall x_1, x_2 \in S
\]

Convexity adds advantages such as strong duality, tractability and extendability in a learning environment. Strong duality provides an alternate mechanism for finding optimal solution, tractability suggests that local minimum in the performance surface is global minimum and extendability allows the non-weighted sum of function to hold the same convexity property as the function itself [64]. However, convexity is difficult
to attain in real world and therefore, convexity has been generalized to new class of function called pseudoconvex [33].

**Definition: Pseudoconvex:** Let \( f : S \rightarrow \mathbb{R} \) be a differentiable function, where \( S \) is a non-empty subset of \( \mathbb{R}^n \). The function \( f \) is said to be pseudoconvex if:

\[
\nabla f(x_1)^\top (x_2 - x_1) \geq 0 \implies f(x_2) \geq f(x_1) \quad \forall x_1, x_2 \in S
\]

i.e., a differentiable function is pseudoconvex if it is increasing in any direction where it has a positive directional derivative. In the convexity analysis of the correntropy cost presented in [64], pseudoconvexity has been established for correntropy cost for one dimensional error, which occurs in online method. It establishes that gobal optimal solution can be obtained with sufficiently small value of stepsize for gradient update rule, although with slow rate of convergence. However, fast convergence can be attained by using the solution of quadratic cost function at a starting point. This suggests that the MCC for one dimensional error is pseudoconvex and therefore, the following is true:

**global optimal solution at** \( x^* \in \mathbb{R} \iff \nabla J_c(x^*) = 0 \)

**Definition: Invex:** Let \( f : S \rightarrow \mathbb{R} \) be a differentiable function, where \( S \) is a non-empty subset of \( \mathbb{R}^n \). The function \( f \) is said to be invex if and only if:

\[
f(x_2) \geq f(x_1) + \psi(x_1, x_2)^\top \nabla f(x_1) \quad \forall x_1, x_2 \in S
\]

where \( \psi : S \times S \rightarrow \mathbb{R}^n \) is some arbitrary vector function [6]. Invexity preserves tractability and extendability. Correntropy cost has been established as invex in [64] for higher dimension with cumulative error of \( n \) samples. This would correspond to an expected risk function and therefore, risk associated with correntropy cost is invex and therefore, every local minima is a global minima.
5.1.3 Kernel MCC

Kernel MCC (KMC) [76] extends MCC to kernel methods to show robust non-linear filtering in presence of non-Gaussian noise. Following the same notation as in section 2.4, given the latest input-output pair \( \{ \varphi(u_n), y_n \} \), the current \( \Omega_n \) is updated using gradient ascent update rule on the cost defined by 5–2 as follows:

\[
\begin{align*}
\Omega_0 &= 0 \\
e_n &= y_n - \hat{y}_n = y_n - \Omega_{n-1}^T \varphi(u_n) \\
\Omega_n &= \Omega_{n-1} + \eta \frac{\partial}{\partial \Omega_n} [\kappa_{\sigma_c}(e_n)] \\
&= \Omega_{n-1} + \frac{\eta}{\sigma_c^2} \exp \left( \frac{-e_n^2}{2\sigma_c^2} \right) e_n \varphi(u_n) \\
\Omega_n &= \tilde{\gamma} \sum_{i=1}^{n} \left[ \exp \left( \frac{-e_i^2}{2\sigma_c^2} \right) e_i \varphi(u_i) \right]
\end{align*}
\]

(5–3)

where, \( \tilde{\gamma} = \frac{\eta}{\sigma_c^2} \) is the step-size parameter. Then, the output at \( n + 1 \) is,

\[
\begin{align*}
\hat{y}_{n+1} &= \Omega_n^T \varphi(u_{n+1}) = \sum_{i=1}^{n} \tilde{\gamma} \exp \left( \frac{-e_i^2}{2\sigma_c^2} \right) e_i \varphi(u_i)^T \varphi(u_{n+1}) \\
&= \sum_{i=1}^{n} \tilde{\gamma} \exp \left( \frac{-e_i^2}{2\sigma_c^2} \right) e_i \kappa(u_i, u_{n+1})
\end{align*}
\]

(5–4)

where \( \kappa \) is the kernel for the mapper. KMC has been trained using MSE in the initial step to obtain faster convergence. A sufficient condition for convergence of KMC is established in [76], and has also been shown to maintain the self-regularization property of KLMS.

5.2 Mixture Kernel Maximum Correntropy Criteria (MxKMC)

Let, \( \varphi_m : U \rightarrow \mathbb{F}_{\kappa_m} \) be the corresponding mapping functions to \( P \) different RKHSs \( \mathbb{F}_{\kappa_m} \) induced by kernels \( \kappa_m, m \in \{1, \ldots, P\} \). MxKMC learns functions \( f_m \) in individual RKHS \( \mathbb{F}_{\kappa_m} \) and linearly combines the functions, thus, \( f = \sum_{m=1}^{P} \beta_m f_m \). This allows sequentially learning \( \beta \) together with \( f_m \in \mathbb{F}_{\kappa_m} \) in their respective RKHSs. Now, \( f \) can be obtained as the stochastic approximation to the following ERM problem associated with
MCC in each RKHS $\mathbb{F}_{\kappa_m}$:

$$\min_{f \in \mathbb{F}} \sum_{i=1}^{N} \left[ \kappa \left( y_i, f(u_i) \right) \right] = \min_{f_m \in \mathbb{F}_{\kappa_m}, \sum_{m=1}^{p} \beta_m > 0, \sum_{m=1}^{p} \beta_m = 1} \sum_{i=1}^{N} \left[ \kappa_{\sigma_c} \left( y_i, \sum_{m=1}^{p} \beta_m f_m(u_i) \right) \right]$$  \hspace{1cm} (5–5)

which requires learning an additional parameter $\beta_m$. Here, the ERM is invex, suggesting that it has a global solution. Therefore, (5–5) can be solve by alternating minimization with respect to $f_m$ and $\beta_m$.

For certain $\beta_m$, using reproducing property as in (2–9), the ERM problem using the KAF notations is

$$\min_{\Omega_m \in \mathbb{F}_{\kappa_m}} \sum_{i=1}^{N} \kappa_{\sigma_c} \left( y_i, \sum_{m=1}^{p} \beta_m \Omega_m \varphi_m(u_i) \right)$$  \hspace{1cm} (5–6)

Given the input-output pair $\{ \varphi_k(u_n), y_n \}$, at $n^{th}$ iteration, $\Omega_{k,n}$ is given by the following gradient update rule:

$$\begin{align*}
\Omega_{k,0} &= 0 \\
e_n &= y_n - \sum_{m=1}^{p} \beta_m \Omega_{m,n-1} \varphi_m(u_n) \\
\Omega_{k,n} &= \Omega_{k,n-1} + \gamma \exp \left( \frac{-e_n^2}{2\sigma_c^2} \right) e_n \beta_k \varphi_k(u_n) \\
\Omega_{k,n} &= \sum_{i=1}^{n} \alpha_{k,i} \varphi_k(u_i)
\end{align*}$$  \hspace{1cm} (5–7)

where, $\alpha_{k,i} = \gamma \exp \left( \frac{-e_i^2}{2\sigma_c^2} \right) e_i \beta_k$ $\alpha_{k,1}, \ldots, \alpha_{k,n}$ is the coefficient vector for linear combination and $\gamma$ is the step-size for gradient update. In addition to learning $\Omega_k$, the stochastic weights $\beta_k$ are learned using a non-linear gating function,

$$\beta_k = \frac{\exp(v_k)}{\sum_{j=1}^{p} \exp(v_j)}$$  \hspace{1cm} (5–8)

where, the gate parameter $v_k$ is the intermediate weight at the $n^{th}$ iteration. The gating function (7–6) maintains convexity in $\beta$. This requires learning $v_k$ using an additional
Algorithm 4 MxKMC

Input:
P Kernels

Initialization

\( \hat{y}_o = 0, \)

Gate parameter \( v_{m,0} = \frac{1}{P}, \forall m \in \{1, \ldots, P\}, \)

Learning rates: \( \bar{\gamma}, \bar{\mu} > 0, \)

Repeat \( \forall \ i \in \{1, \ldots, n\}: \)

1. Evaluate: \( \hat{y}_i \)
2. Compute error: \( e_i = y_i \) − \( \hat{y}_i \)
3. Update gate parameter: \( v_{m,i} = v_{m,i-1} - \bar{\mu} \nabla v_{m,i} \)
4. Compute gate output: \( \beta_{m,i} \)
5. Update filter coefficient \( \alpha_{m,i} = \bar{\gamma} \beta_{m,i} \exp \left( \frac{-e_i^2}{2\sigma_c^2} \right) e_i \)

Until convergence

update rule:

\[ v_{k,n+1} = v_{k,n} - \mu \frac{\partial \kappa_{\sigma_c}(e_n)}{\partial v_{k,n}} \] (5–9)

where, \( \mu \) is the step-size for gradient update. The gradient can be solved as follows:

\[ \nabla v_{k,n} = \frac{\partial \kappa_{\sigma_c}(e_n)}{\partial v_{k,n}} = \frac{\partial \kappa_{\sigma_c}(e_n)}{\partial \beta_{k,n}} \frac{\partial \beta_{k,n}}{\partial v_{k,n}} = \frac{1}{\sigma_c^2} \exp \left( \frac{-e_n^2}{2\sigma_c^2} \right) e_n \nabla y \nabla s \] (5–10)

where, \( \nabla y = \frac{\partial}{\partial y_{k,n}} (y - \hat{y}_n) \), and \( \nabla s = \frac{\partial}{\partial s_{k,n}} \beta_{k,n} \). Following similar steps from (3–21) to (3–25), we get,

\[ \nabla v_{k,n} = -\frac{1}{\sigma_c^2} \exp \left( \frac{-e_n^2}{2\sigma_c^2} \right) e_n \left[ \hat{y}_{k,n} - \sum_{l=1}^{P} \hat{y}_{l,n} \right] \beta_{k,n} [1 - \beta_{k,n}] \] (5–11)
where, $\hat{y}_{k,n} = \Omega_{k,n}^T \varphi_k(u_n)$. Then, the update (5–9) becomes,

$$v_{k,n+1} = v_{k,n} + \tilde{\mu} \exp \left( \frac{-e_n^2}{2\sigma_c^2} \right) \epsilon_n \left[ \hat{y}_{k,n} - \sum_{i=1 \atop i \neq k}^P \hat{y}_{i,n} \right] \beta_{k,n} [1 - \beta_{k,n}]$$  \hspace{1cm} (5–12)

where $\tilde{\mu} = \frac{1}{\sigma_c}$ is the new step-size. Now, the prediction $\hat{y}_{n+1}$ is given by,

$$\hat{y}_{n+1} = \Omega(u_{n+1}) = \sum_{m=1}^P \beta_m \Omega_m(u_{n+1})$$

$$= \sum_{m=1}^P \beta_m \Omega_m^T \varphi_m(u_{n+1})$$

$$= \sum_{m=1}^P \sum_{i=1}^n \beta_m \alpha_m \kappa_m(u_i, u_{n+1})$$  \hspace{1cm} (5–13)

### 5.3 Experiments and Results

#### 5.3.1 Synthetic Data

In this experiment, a simple frequency doubling problem is studied in presence of impulsive noise by creating two sine waves of frequencies $f_o$ and $2f_o$. Learning the desired system output $2f_o$ given $f_o$ is non-linear problem and presence of non-Gaussian noise in the output complicates it further. An impulsive noise was created using mixture of two Poisson models:

$$N = 0.7 P(m_1, \bar{\sigma}) + 0.3 P(m_2, \bar{\sigma})$$

where, $m_1 = 0$ and $m_2 = 2$ are the means and $\bar{\sigma} = 0.01$ is the variance for the distributions. The sine waves with two frequencies are shown in Fig. 5-1. For the purpose of training, 1500 noise free samples of signal with $f_o$ is used to predict 1500 noisy samples of signal with $2f_o$ and 200 noise free samples of the former are used to reproduce 200 noise free samples of the latter. In this experiment, the performance of MxKMC and MxKLMS using kernels with $\sigma = \{0.1, 0.5, 1, 1.5\}$ is compared with that of KMC using each of the kernels. The experiment was performed with 50 random
Figure 5-1. Frequency doubling initializations. The average normalized test MSE (NMSE) obtained at each training instance is shown in Fig. 5-2 for these methods. It is clear that MxKMC outperforms MxKLMS and even the KMC. As expected, MxKLMS using LMS is not able to overcome the effect of impulse noise, whereas, MxKMC using correntropy cost cleanly rejects the outlier noise and produces a noise-free output. KMC with $\sigma = 0.5$ also performs well but its speed of convergence is slower than that of MxKMC. The step-sizes used for MxKLMS are $\eta = 0.5$, $\mu = 0.5$, for MxKLMC are $\bar{\gamma} = 0.8$, $\bar{\mu} = 0.5$ and for KMC is $\bar{\gamma} = 0.3$. The parameters are selected to obtain optimal solution for each method. The difference in performance can also be noticed based on the kernel weight tracks of MxKMC and MxKLMS in Fig. 5-3. Notice that the weight track of MxKMC eventually stabilizes whereas that of MxKLMS continues to be unstable. The results are quantified in Table 5-1 for all the methods.
Table 5-1. Performance comparison of MxKMC, MxKLMS and KMC for noisy prediction of synthetic data.

<table>
<thead>
<tr>
<th>Method</th>
<th>Gaussian Kernel($\sigma$)</th>
<th>NMSE</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKMC</td>
<td>[0.1, 0.5, 1, 1.5]</td>
<td>0.005±0.001</td>
<td>37.66±1.91</td>
</tr>
<tr>
<td>MxKLMS</td>
<td>[0.1, 0.5, 1, 1.5]</td>
<td>0.691±0.009</td>
<td>-1.36±0.11</td>
</tr>
<tr>
<td>KMC</td>
<td>0.1</td>
<td>0.051±0.020</td>
<td>19.48±2.36</td>
</tr>
<tr>
<td>KMC</td>
<td>0.5</td>
<td>0.301±0.026</td>
<td>6.09±0.80</td>
</tr>
<tr>
<td>KMC</td>
<td>1.5</td>
<td>2.576±0.822</td>
<td>-11.75±3.03</td>
</tr>
<tr>
<td>KMC</td>
<td>2</td>
<td>2.942±0.623</td>
<td>-13.22±1.97</td>
</tr>
</tbody>
</table>

5.3.2 Noisy Short-term Prediction of Lorenz Chaotic Time Series

In this section, we compare the performance of MxKAC with MxKLMS along with the other existing multi-kernel adaptive filtering methods MKLMS and MKNLMS described in chapter 3, that use the LMS cost function like MxKLMS. The sample data for Lorenz chaotic system was generated as in section 3.6.5, with same preprocessing to obtain input data shown in Fig. 3-6A. We utilize the $x$-dimension of the time series, and predict the current value $u(i)$ from past 5 values $u_i = [u(i - 5), u(i - 4), ..., u(i - 1)]^T$. Same impulsive noise from section 5.3.1 was added to the desired response and
Figure 5-3. Kernel weights across samples for synthetic data prediction for A) MxKMC B) MxKLMS.

noisy prediction was performed. We performed 50 Monte-Carlo runs with random initializations. The average mean test NMSE obtained on 100 sample length test data
after each training instance is shown in Fig. 5-4. As expected, all the methods using LMS perform very poorly compared to MxKMC. The performance is also reflected in the stable kernel weight tracks of MxKMC in Fig. 5-3A compared to very unstable the kernel weight tracks of MxKLMS in Fig. 5-3B. The results are quantified in Table. 5-2 along with the corresponding parameters used. MKLMS and MKNLMS have been implemented without any dictionary pruning techniques.

![Graph showing comparison of test NMSE across training samples for Lorenz chaotic data.](image)

**Figure 5-4.** Comparison of test NMSE across training samples for Lorenz chaotic data.

<table>
<thead>
<tr>
<th>Method</th>
<th>NMSE</th>
<th>Gain</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKMC</td>
<td>0.08±0.05</td>
<td>22.49±2.66</td>
<td>$\tilde{\gamma} = 0.5$, $\tilde{\mu} = 0.5$</td>
</tr>
<tr>
<td>MxKLMS</td>
<td>35.10±33.92</td>
<td>-4.75±2.35</td>
<td>$\eta = 0.5$, $\mu = 0.5$</td>
</tr>
<tr>
<td>MKNLMS</td>
<td>20.43±18.64</td>
<td>-5.54±0.69</td>
<td>$\rho = 10^{-4}$, $\eta = 0.1$</td>
</tr>
<tr>
<td>MKLMS</td>
<td>27.65±24.78</td>
<td>-7.71±1.37</td>
<td>$\hat{\eta} = 0.05$, $\eta = 0.08$</td>
</tr>
</tbody>
</table>

### 5.4 Summary

In this chapter, we developed the multi-kernel adaptive filtering called MxKMC suitable for non-Gaussian noisy environment, using KMC criteria in the MxKLMS
framework. We empirically showed that MxKMC outperforms the mono-kernel KMC as well as MxKLMS and other existing multi-kernel adaptive filtering methods even in presence of impulse noise. Moreover, we showed empirically that it is able to select the appropriate kernel from the pool of kernels even in presence of noise.
Figure 5-5. Kernel weights across samples for Lorenz chaotic data prediction for A) MxKMC B) MxKLMS.
CHAPTER 6
ROBUST CLASSIFICATION USING KERNEL ADAPTIVE CLASSIFIER

6.1 Background

6.1.1 C-loss Function

The optimization goal in classification can be reframed as maximizing the similarity between the classifier output and the true label in the correntropy sense. Therefore, the correntropy induced loss function (c-loss) \[59\] was defined such that minimization of the empirical risk is equivalent to maximization of correntropy. C-loss is defined as:

\[
L_c(y, f(u)) = \zeta \left[ 1 - \kappa_{\sigma_c}(y, f(u)) \right]
\]

\[
= \zeta \left[ 1 - \kappa_{\sigma_c}(y - f(u)) \right]
\]

\[
= \zeta \left[ 1 - \kappa_{\sigma_c}(1 - yf(u)) \right]
\]

(6–1)

where \(\zeta\) is a positive scaling constant chosen such that, for classification margin \(\zeta = yf(u), L_c(\zeta = 0) = 1\), so that, \(\zeta = \left[ 1 - \exp\left(\frac{-1}{2\sigma_c^2}\right) \right]^{-1}\). Margin is a quantity that measures the distance of the sample \(u\) from the discriminant function \(f(u)\) and can be alternatively defined as \(e = y - f(u)\) in which case, \(L_c(e = 1) = 1\), so that, \(\zeta = \left[ 1 - \exp\left(\frac{-1}{2\sigma_c^2}\right) \right]^{-1}\). Also, \(\kappa_{\sigma_c}\) is a Gaussian kernel with the kernel size parameter \(\sigma_c\).

From the definition of correntropy, \(\sigma_c\) governs the region where similarity is measured and therefore, is a very crucial factor in case of c-loss as well \[28\]. Fig. 6-1A compares c-loss function for \(\sigma_c = 0.5\) and \(\sigma_c = 1\) with 0 - 1 loss and square loss, plotted against the margin \(\zeta\). C-loss with \(\sigma_c = 0.5\) gives a close approximation to 0 - 1 loss while c-loss with \(\sigma_c = 1\) is almost linear, and closely approximates the hinge loss \[59\] for \(\zeta \in [-1, 1]\).

The expected risk associated with the c-loss is,

---

Portions of this Chapter are published for in the following manuscript: Rosha Pokharel, Jose C. Principe, "Kernel Classifier with Correntropy loss", IJCNN, pp. 1-6, 2012.
Figure 6-1. Properties of C-loss: A) C-loss function for different values of \( \sigma_c \), plotted along with 0 - 1 loss and square loss (Sq-loss) across margin \( \varsigma = yf(u) \) B) Derivative of c-loss function plotted across \( |e| = |y - f(u)| \) for different values of \( \sigma_c \), compared with derivative of square loss. C-loss is less sensitive in the region of mis-classification, i.e., for \( 1 < |e| < 2 \).

\[
R_c(f) = \zeta \{ 1 - E[\kappa_{\sigma_c}(1 - yf(u))] \} \\
= \zeta \{ 1 - E[\kappa_{\sigma_c}(y - f(u))] \} \\
= \zeta \{ 1 - \nu(Y, f(U)) \} \quad (6-2)
\]

where \( \nu \) is the correntropy of error as defined in (5–1) and \( y \in Y, u \in U \). Clearly, minimizing the above risk is equivalent to maximizing the similarity (in correntropy sense) between predicted label \( f(u) \) and the desired label \( y \). Let, \( \xi = Y - f(U) \), then,

\[
\hat{\nu}(\xi) = \frac{1}{N} \sum \kappa_{\sigma_c}(e_i) \quad (6-3)
\]

Hence, this is an estimator of pdf of \( \xi \) evaluated at 0 and maximizing correntropy of errors \( \xi \) of a classifier maximizes \( p(\xi = 0) \), which corresponds to minimizing the risk. The contour of risk function for \( \sigma_c = 0.5 \) and \( \sigma_c = 1 \) in the 2D space of errors is shown in Fig. 6-2. It shows that the risk behaves like \( L_2 \) norm near the origin and slowly transitions to \( L_0 \) as the error increases. For larger errors, the function approaches the \( L_0 \)
Figure 6-2. The empirical risk function in a 2D space of errors $e = y - f(x)$ for c-loss with (A) $\sigma_c = 0.5$ (B) $\sigma_c = 1$. The risk behavior is similar to $L_2$ norm closer to origin and slowly transitions to $L_1$ norm. For the larger error, the risk function approaches $L_0$ or the counting norm. For a smaller kernel, the rate at which it transitions to $L_1$ norm is higher.

or counting norm. Also, notice that the rate of transition for $\sigma_c = 0.5$ is faster than that for $\sigma_c = 1$.

The derivative of c-loss function given by (6–4) also has a very desirable property, governed by $\sigma_c$ parameter.

$$\frac{\partial L_c(e_n)}{\partial e_n} = \frac{\zeta e_n}{\sigma_c^2} \exp \left( \frac{-e_n^2}{2\sigma_c^2} \right)$$ (6–4)

In the plot of the derivatives shown in Fig. 6-1B, we can notice that, for $\sigma_c = 0.5$, c-loss is more sensitive in the region of correct classification, i.e., $0 < |e_n| < 1$ and less sensitive in the region of miss-classification $|e_n| > 1$ and outliers. Also, the sensitivity is attenuated in the region of the boundary, i.e, when $|e_n| = 1$. Therefore, the noisy samples close to the boundary have less effect in the weight updates. This produces a smooth decision boundary and thus avoids overfitting. The sensitivity increases with increasing value of $\sigma_c$. It is clear that sensitivity for square-loss is linear with respect to the error, which is not a desirable property for a loss function, as it gives high emphasis
to the large errors and also to the samples closer to boundary, thus giving a poor classification accuracy and poor generalization.

### 6.1.2 Generalized Convexity of C-loss

With the advantage of having a tuning parameter $\sigma_c$, that governs the region of emphasis near the decision boundary, there is also a trade off associated. This is due to the fact that for $\sigma_c > 1$, the c-loss function is clearly convex, but its convexity cannot be guaranteed for $\sigma_c < 1$. However, recently the convexity analysis of the c-loss presented in [64] suggests that, c-loss is pseudoconvex for one dimension error and invex when it is defined as the cumulative error. Therefore, the risk associated with c-loss is invex. A detail description on generalized convexity of correntropy is presented in section 5.1.2.

### 6.2 Kernel Adaptive Classifier Algorithm

Given the pairwise sample, $\{u_n, y_n\}$, $u_n \in U \subseteq \mathbb{R}$, $y_n \in \mathbb{Y} = \{-1, 1\}$, the approximating function $f: U \rightarrow \mathbb{Y}$ of a learning machine, and thus, the KAC can be represented as:

$$f(u_n) = \sum_{i=1}^{n-1} \Omega_i \kappa(u_i, u_n)$$

(6–5)

where, $\varphi$ is the functional mapping that maps each input vector $u_i$ to the high dimensional feature space $F$ using mercer kernel $\kappa$. Without loss of generality, the Gaussian kernel is used because of its universal approximation property [34]. Thus, by placing a kernel at every sample, the classifier maps each incoming input sample to the desired response $y \in \{-1, 1\}$. With the use of the Gaussian kernel, such a network can be seen as a growing sum of radial basis functions, each associated with a different center $u_n$, and weighted by a coefficient vector $\Omega_n$ obtained from the training data. Then, the empirical risk in (6–2) can be minimized by using the stochastic gradient optimization of the c-loss function. For an incoming pairwise sample $\{\varphi(u_n), y_n\}$ at time $n$, the update rule for $\Omega$ is:
\[ \Omega_0 = 0 \]

\[ e_n = y_n - \hat{y}_n = y_n - \Omega_{n-1}^T \varphi(u_n) \]

\[ \Omega_n = \Omega_{n-1} - \eta \frac{\partial}{\partial \Omega_n} [\zeta \{1 - \kappa_{\sigma}(e_n)\}] \]

\[ = \Omega_{n-1} + \eta \frac{\zeta}{\sigma_c^2} \exp \left( \frac{-e_n^2}{2\sigma_c^2} \right) e_n \varphi(u_n) \]

\[ \Omega_n = \gamma \sum_{i=1}^{n} \left[ \exp \left( \frac{-e_i^2}{2\sigma_c^2} \right) e_i \varphi(u_i) \right] \]

where, the step-size \( \eta \) and constants \( \zeta \) and \( \sigma_c \) are dissolved into new step-size \( \gamma = \eta \frac{\zeta}{\sigma_c^2} \).

Then, the predicted output is given by,

\[ \hat{y}_n = \Omega_n^T \varphi(u_n) = \gamma \sum_{i=1}^{n-1} \left[ \exp \left( \frac{-e_i^2}{2\sigma_c^2} \right) e_i \kappa_{\sigma}(u_i, u_n) \right] \]

It should be noted that, both MCC and c-loss have same gradient and would give same update rule in KLMS framework as derived in (6–6) considering their root to correntropy. However, their interpretations are different as they they haven been proposed to address two different tasks: robust cost function for non-Gaussian noise filtering, and smooth robust loss function for classification.

### 6.3 Property of KAC Using C-loss: Self-Regularization

KLMS has been established as a self-regularized method [29] by showing that it is well posed in the sense of Hadamard. With the change of the cost function, the classification problem addressed by KAC using c-loss is still a mapping like filtering, and so the same learning rule as KLMS applies. Hence, the properties of regularization would still hold. In this context, this means that the algorithm does not need to be explicitly regularized by a penalty in the cost and that it is sufficient to properly select the step-size to achieve different degrees of regularization. Stability and solution norm constraints are very closely related to each other [40], see [29] for more details.
Under the $H^\infty$ stability condition, the prediction error for KAC can be shown to satisfy the following inequality, by following similar arguments as in [29]:

$$\|e\|^2 < \gamma^{-1}\|\Omega^o\|^2 + 2\|e^o\|$$

(6–8)

where $e = [e_1, ..., e_N]^T$ and $e^o = [e^o_1, ..., e^o_N]^T$, $\{e^o_i\}_{i=1}^N$ is modeling uncertainty. Then, the squared norm of $\Omega$ can be obtained as:

$$\|\Omega_N\|^2 = \langle \Omega_N, \Omega_N \rangle$$

$$= \|\gamma \sum_{i=1}^{N} e_i \exp \left(-\frac{e_i^2}{2\sigma_c^2}\right) \varphi(u_i) \|^2$$

$$= \gamma^2 N e^T C_e G_\varphi C_e e$$

$$= \gamma^2 N e^T C_e Q \text{diag}\{\tau_1, ..., \tau_N\} Q^T C_e e$$

$$\leq \gamma^2 N \tau_1 \|C_e e\|^2$$

(6–9)

where,

$$G_\varphi = \text{gram matrix}$$

$$\tau_1 = \text{largest eigenvalue of input autocorrelation matrix } R_\varphi$$

$$Q = \text{orthonormal matrix}$$

$$G_\varphi = Q \text{diag}\{\tau_1, ..., \tau_N\} Q^T$$

$$C_e = \text{diag}\left\{\exp\left(-\frac{e_1^2}{2\sigma_c^2}\right), \exp\left(-\frac{e_2^2}{2\sigma_c^2}\right), ..., \exp\left(-\frac{e_N^2}{2\sigma_c^2}\right)\right\}$$

Using (6–8),

$$\|\Omega\| < \sqrt{\gamma^2 N \tau_1 \|C_e\| (\gamma^{-1}\|\Omega^o\|^2 + 2\|v\|^2)}$$

(6–10)

The solution norm has an upper bound, which is governed by the number of training samples $N$, the step size $\gamma$, the largest eigen value $\tau_1$ and an extra factor $C_e$. Since $C_e$ is a monotonically decreasing function of $e_i^2$, the solution norm is clearly bounded from
above. Therefore, the algorithm KAC with c-loss is well posed in the sense of Hadamard and so is self-regularized, with the regularization governed by the step-size $\gamma$. Having such a property is one of the major advantages for a classifier, as this gives a much simpler way to select the regularization parameter to govern generalization, compared to the tedious and computationally expensive task of regularization parameter selection like in SVM. Due to the presence of $C_e$, the solution norm for KAC has a tighter bound compared to KLMS with square loss, in which the factor $C_e$ is not present. Later in this chapter, we show that KAC using c-loss is less dependent on the step-size, compared to KLMS, which indicates that the generalization is less affected by the step-size, due to the presence of $C_e$.

### 6.4 Experiments and Results

KAC can be implemented online as well as in batch mode, due to its iterative nature. In this section, the classification performance of KAC has been analyzed based on two synthetic datasets, one in online mode and one in batch mode, and on the benchmark datasets from UCIrvine machine learning repository [7] in batch mode. In online mode, the classifier learns from single sweep of the training samples while in batch mode, the training samples are passed through the classifier in *epochs*. Each epoch contains a fixed number of training samples, the corresponding coefficients are added at every epoch to finally obtain a coefficient vector. With the synthetic datasets, the classification performance is first compared qualitatively based on the decision boundaries or the discriminant functions obtained using c-loss as well as square loss for different initial conditions. The decision boundary obtained using soft margin (Gaussian) SVM is also presented for the best values of the box constraint parameter, $C$, selected by cross validation$^1$. In case of benchmark datasets, the generalization performance of KAC and the mean percentage accuracy obtained using c-loss and square loss

---

$^1$ C-SVM was implemented using libsvm [9]
are compared for mapper kernel size \( \sigma \), c-loss kernel size \( \sigma_c \) and step-size \( \gamma \). The performance is also compared with SVM and neural networks (NN) using c-loss and square loss. All the datasets used are normalized to have unit variance along each feature.

Figure 6-3. Decision boundaries obtained for Synthetic Data-1 using KAC with c-loss \((\sigma_c = 0.5)\), KAC with square loss, and C-SVM \((C = 0.8)\), for \( \sigma = 1 \) and A) \( \gamma = 0.01 \) B) \( \gamma = 0.05 \) C) \( \gamma = 0.1 \). Boundary from c-loss is similar to the best boundary obtained from SVM in all cases but that from square loss degrades gradually.
6.4.1 Synthetic Data-1

This experiment shows the classification performance when data is passed through once. The dataset consists of two classes, both drawn from two different Gaussian distributions, one with smaller variance than the other. Fig. 6-3 compares the discriminant functions obtained by using c-loss \((\sigma_c = 0.5)\) and square loss in KAC for different values of step-size \(\gamma\) and for mapper kernel size \(\sigma = 1\), along with that obtained from C- SVM \((C = 0.8)\). The ratio of training and test data for this experiment is 70:30. The decision boundary is optimal in case of smaller values of \(\gamma\) for both the loss functions. However, as the value of \(\gamma\) increases, generalization ability of square loss starts to decrease rapidly, in contrast to c-loss, which is able to generalize well even in worst case, and is still very similar to that of SVM.

6.4.2 Synthetic Data-2

This experiment compares the decision boundary obtained using c-loss \((\sigma_c = 0.5)\) function and square loss function for different values of step-size \(\gamma\) and mapper kernel size \(\sigma = 1\), along with C-SVM \((C = 0.2)\). In this case, the classifier is used in batch mode by repeating the samples for 50 epochs and the ratio of training and test data for this experiment is 70:30. Here, the class-1 data has crescent moon structure and class-2 data is a drawn from a Gaussian distribution. The decision boundary obtained using c-loss is optimal and very similar to that of SVM for all values of \(\gamma\), but the boundary degrades gradually in case of square loss.

6.4.3 Pima Indian Diabetes Dataset

With this dataset, the classifier decides if a test subject falls in a diabetic or a non-diabetic group, based on 8 physiological measurements provided as features. The dataset consists of 768 subjects or samples, out of which, 300 samples have been used for training the classifier while the rest have been used for testing. The classifier performance has been obtained after averaging over 50 Monte Carlo runs. The results thus obtained across training epochs for \(\gamma = 0.03, \sigma_c = 0.5\) and different values of \(\sigma\)
Figure 6-4. Decision boundaries obtained for Synthetic Data-1 using KAC with c-loss ($\sigma_c = 0.5$), KAC with square loss, and C-SVM ($C = 0.2$), for $\sigma = 1$ and A) $\gamma = 0.01$ B) $\gamma = 0.05$ C) $\gamma = 0.1$. Boundary from c-loss is similar to the best boundary obtained from SVM in all cases but that from square loss degrades gradually.

are presented in Fig. 6-5. The mean percentage accuracy of classification at the end of training is tabulated in Table. 6-2 for different values of $\sigma, \gamma$ and $\sigma_c$. C-loss with $\sigma_c = 0.5$ performs better than $\sigma_c = 1$. Results clearly show that c-loss performs better in terms of accuracy and robustness than square loss.
Figure 6-5. Performance of KAC with C-loss ($\sigma_c = 0.5$) and sq-loss on Pima Indian diabetes dataset across epochs for: A) $\sigma = 0.5$ B) $\sigma = 1$ C) $\sigma = 2$. Results are obtained as an average over 50 Monte Carlo runs. Figure continued.
6.4.4 Wisconsin Breast Cancer Dataset

The dataset consists of samples from 683 breast cancer patients and the classification problem in this dataset is to separate the malignant or benign sample based on the 9 features such as size, shape, clump thickness, and cell properties, etc. 300 samples have been chosen for training, the rest for testing the classifier. As earlier, the classifier is trained with c-loss and square loss, for different number of epochs and different values of σ, σ_c and γ. The result obtained from the average over 50 Monte Carlo runs for σ_c = 0.5, γ = 0.03 and different values of σ have been shown in Fig. 6-6 and tabulated in Table. 6-2. As described earlier, the performance of c-loss depends on choice of σ_c, and so the effect of σ_c on performance is clear from Table.6-2 along with that obtained with σ_c = 1. C-loss with σ_c = 0.5 performs better than σ_c = 1. The best performance for both the loss functions are similar but the robustness of c-loss and square loss to the increasing step-size follows similar trend as earlier. It is clear that c-loss is much more robust than square loss even after prolonged training.
Figure 6-6. Performance of KAC with C-loss ($\sigma_c = 0.5$) and sq-loss on Wisconsin breast cancer dataset across epochs for: A) $\sigma = 0.5$ B) $\sigma = 1$ C) $\sigma = 2$. Results are obtained as an average over 50 Monte Carlo runs. Figure continued.
6.4.5 Blood Transfusion Service Center Dataset

This dataset is taken from the Blood Transfusion Service Center in Hsin-Chu city in Taiwan. The problem is to classify the 748 blood donors as ones that donated blood in March 2007 and ones who did not, based on four features. 300 samples are used for training and rest for testing. The classifier is trained with c-loss for different number of epochs and different values of $\sigma$, $\sigma_c$ and $\gamma$. The result obtained from the average of 50 Monte Carlo runs for $\sigma_c = 0.5$, $\gamma = 0.03$ and different values of $\sigma$ is shown in Fig. 6-7 and tabulated in Table. 6-2 along with that obtained with $\sigma_c = 1$. The result follows similar trend as in previous experiments. C-loss with $\sigma_c = 0.5$ is robust to overfitting than square loss.

6.4.6 Comparison with SVM and NN classifiers

The classification results are compared with that of SVM and NN classifiers in Table. 6-1, for the above three datasets. The results are obtained from NN after training for 75 epochs, with single hidden layer and 20 processing elements using...
Figure 6-7. Performance of KAC with C-loss ($\sigma_c = 0.5$) and sq-loss on blood transfusion dataset across epochs for: A) $\sigma = 0.5$ B) $\sigma = 1$ C) $\sigma = 2$. Results are obtained as an average over 50 Monte Carlo runs. Figure continued.
Table 6-1. Performance comparison of KAC, neural network and SVM benchmark dataset. Best of results across epochs is presented for NN and KAC.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Datasets</th>
<th>Wisconsin Breast Cancer</th>
<th>Pima Diabetes</th>
<th>Blood Transfusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>KAC (c-loss)</td>
<td>97.21±0.62</td>
<td>76.25±1.80</td>
<td>78.02±1.71</td>
<td></td>
</tr>
<tr>
<td>KAC (sq-loss)</td>
<td>96.67±0.66</td>
<td>73.76±3.11</td>
<td>75.69±3.43</td>
<td></td>
</tr>
<tr>
<td>NN (sq-loss)</td>
<td>96.50±1.26</td>
<td>72.78±1.18</td>
<td>75.13±3.47</td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>97.10±0.52</td>
<td>75.16±0.65</td>
<td>77.26±1.80</td>
<td></td>
</tr>
</tbody>
</table>

back propagation. SVM and NN parameters are selected by cross validation. Table 6-1 shows the best percentage accuracy obtained after training over 25, 50 and 75 epochs for KAC and NN and that obtained using best value of C-parameter for SVM, averaged over 50 Monte Carlo runs. The three methods show comparable performance, however, KAC has some advantages over SVM and NN. KAC is a sequential method that can classify a sample as it comes, unlike batch SVM that has to consider all the samples at ones that increases computational burden. Also, from the invexity of c-loss and the well-posedness of KAC, it does not suffer from local minima like NN. Moreover,
using c-loss, KAC is less prone to overfitting even after prolonged training, and its
regularization can be controlled by carefully assigning step-size, which is a major
advantage over SVM that requires expensive cross validation method to select its c-
parameter.

6.5 Summary

We have successfully implemented the robust c-loss function in KLMS framework
to obtain an online classifier KAC. We showed that with c-loss, KAC is still a self
regularized framework where the regularization parameter is simply the step-size. We
argued that presence of c-loss provides a tighter bound to the solution norm, and thus is
less affected by the value of step-size, making the model robust. Moreover, KAC is less
prone to overfitting even after training through multiple epochs, which is an advantage
over expensive cross-validation methods required for SVM.
Table 6-2. Mean classification accuracy of KAC with C-loss and square loss on Pima Indian diabetes dataset, Wisconsin breast cancer (WBC) dataset and blood transfusion service center (BTS) dataset for different values of mapper kernel size $\sigma$, step-size $\gamma$ and c-loss kernel size $\sigma_c$.

<table>
<thead>
<tr>
<th>Data</th>
<th>Mapper kernel size, $\sigma$</th>
<th>0.5</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>50</td>
<td>75</td>
</tr>
<tr>
<td>Pima</td>
<td>C-Loss ($\sigma_c = 0.5$)</td>
<td>70.71±1.48</td>
<td>70.15±1.47</td>
<td>69.77±1.70</td>
</tr>
<tr>
<td></td>
<td>C-Loss ($\sigma_c = 1$)</td>
<td>70.71±1.80</td>
<td>71.02±1.80</td>
<td>69.29±1.77</td>
</tr>
<tr>
<td></td>
<td>Square-Loss</td>
<td>70.16±1.57</td>
<td>69.68±1.54</td>
<td>68.25±2.21</td>
</tr>
<tr>
<td>WBC</td>
<td>C-Loss ($\sigma_c = 0.5$)</td>
<td>95.83±0.41</td>
<td>95.64±0.56</td>
<td>94.49±0.82</td>
</tr>
<tr>
<td></td>
<td>C-Loss ($\sigma_c = 1$)</td>
<td>95.75±0.69</td>
<td>95.50±0.79</td>
<td>95.64±0.78</td>
</tr>
<tr>
<td></td>
<td>Square-Loss</td>
<td>95.72±0.81</td>
<td>95.44±0.84</td>
<td>93.63±4.26</td>
</tr>
<tr>
<td>BTS</td>
<td>C-Loss ($\sigma_c = 0.5$)</td>
<td>78.02±1.71</td>
<td>77.58±1.56</td>
<td>77.27±1.74</td>
</tr>
<tr>
<td></td>
<td>C-Loss ($\sigma_c = 1$)</td>
<td>77.02±1.81</td>
<td>75.18±1.76</td>
<td>75.01±1.92</td>
</tr>
<tr>
<td></td>
<td>Square-Loss</td>
<td>75.69±3.43</td>
<td>70.08±5.12</td>
<td>66.76±5.62</td>
</tr>
</tbody>
</table>
Like any other kernel method, kernel selection problem needs to be addressed in the proposed kernel adaptive classifier. The multiple kernel learning (MKL) techniques have been popularly implemented to solve classification tasks for batch kernel methods using SVM. For example, Ben-Hur et al. [5] suggest combining pairwise kernels using two methods: weighted and unweighted sum. Here the pairwise kernels express the similarity between pairs of objects, especially in bioinformatics applications, in terms of similarities between individual objects [16]. Lanckriet et al. [27] and Sonnenburg et al. [61] show that the simplest way is to use an unweighted sum of kernel functions. Tanabe et al. [65] use linear combination of kernels where kernel weights are selected based on the performance values obtained by each kernel separately, in order to avoid learning weights using some optimization techniques [16]. Using the notion of similarity between two kernels, called kernel alignment [24, 48], is also popular in the the field of MKL. However, all these various implementations of MKL techniques are only suitable for batch methods as mentioned earlier in chapter 3. Therefore, this section extends the MxKLMS framework to classification to produce mixture kernel adaptive classifier (MxKAC) using the robust loss function c-loss.

We have previously demonstrated a very good tracking ability of MxKLMS specially in case of non-stationary environment. In this chapter, we explore such advantage by focusing on binary classification problem in a non-stationary environment. In a non-stationary classification problem, the spatial geometry of the data changes over time, changing the decision boundary and therefore, the discriminant function. In such situation, contribution from past samples in classifying new sample is minimal, and might even degrade the prediction ability so we have to introduce a forgetting factor in the framework. Use of exponential forgetting factor to demonstrate the tracking ability of KLMS in non-stationary environment has also been introduced in [22].
7.1 Mixture Kernel Adaptive Classifier for Non-Stationary Classification

As in MxKLMS, let us start with a set of $P$ different positive definite kernels $\kappa_m, m \in \{1, \ldots, P\}$ inducing $P$ respective RKHSs $F_m$. MxKAC is based on learning function $f_m : U \rightarrow Y, U \subset \mathbb{R}, Y = \{-1, 1\}$, in individual RKHS $F_{\kappa_m}$ and linearly combining these functions, such that, $f = \sum_{m=1}^{P} \beta_m f_m$, which allows sequentially learning $\beta$ together with $f_m \in F_{\kappa_m}$ in their respective RKHSs. Notice that, unlike standard MKL algorithms where the hypothesis $f$ is learned in the RKHS induced by $\sum_m \beta_m \kappa_m$, here the hypothesis $f_m$ is learned in the RKHS induced by the kernels $\kappa_m$. However, the final function $f$ still exists in the sum of spaces,

$$F_{\kappa} = \{ f = \sum_{m=1}^{P} f_m | f_m \in F_{\kappa_m} \} \quad (7-1)$$

with its norm defined by $(3-14)$. The purpose of introducing the stochastic vector $\beta$ is to create competition among different RKHSs, and thus obtain higher weights to a few relevant kernels.

Now, $f$ can be obtained as the stochastic approximation to the following ERM problem in each RKHS $F_{\kappa_m}$:

$$\min_{f \in \mathbb{R}} \sum_{i=1}^{N} \zeta \left[ 1 - \kappa_{\sigma_c}(e_i) \right] = \min_{f_m \in F_{\kappa_m}} \sum_{i=1}^{N} \zeta \left[ 1 - \kappa_{\sigma_c} \left( y_i - \sum_{m=1}^{P} \beta_m f_m(u_i) \right) \right] \quad (7-2)$$

where we have utilized the expected risk associated with c-loss function defined in $(6-2)$. The ERM in $(7-2)$ requires learning an additional parameter $\beta_m$. Due to the invexity of c-loss this problem can still be solved using alternating minimization with respect to $f_m$ and $\beta_m$, using gradient descent.

For certain $\beta_m$, using reproducing property as in $(2-9)$, the ERM problem using the adaptive filtering notations is

$$\min_{\Omega_m \in F_{\kappa_m}} \sum_{i=1}^{N} \zeta \left[ 1 - \kappa_{\sigma_c} \left( y_i - \sum_{m=1}^{P} \beta_m \langle \Omega_m, \varphi_m(u_i) \rangle \right) \right] \quad (7-3)$$
Then, for an input-output pair \( \{ \varphi_m(u_n), y_n \} \) at the \( n \)th iteration, \( \Omega_{k,n} \) is given by the following stochastic gradient update update rule:

\[
\begin{align*}
\Omega_{k,0} &= 0 \\
e_n &= y_n - \sum_{m=1}^{P} \beta_m (\Omega_{m,n-1} \varphi_m(u_n)) \\
\Omega_{k,n} &= \Omega_{k,n-1} + \eta \zeta \exp \left( -\frac{e_n^2}{2\sigma_c^2} \right) e_n \beta_k \varphi_k(u_n) \\
\Omega_{k,n} &= \sum_{i=1}^{n} \alpha_{k,i} \varphi_k(u_i)
\end{align*}
\]

(7–4)

where, \( \alpha_{k,i} = \gamma \exp \left( \frac{e_i^2}{2\sigma_c^2} \right) \), \( \beta_k = [\alpha_{k,1}, \ldots, \alpha_{k,n}] \) is the coefficient vector for linear combination and \( \gamma = \frac{\eta \zeta}{\sigma_c^2} \) is the new learning rate. Now, we introduce a forgetting factor \( \lambda \) such that,

\[
\Omega_{k,n} = \sum_{i=1}^{n} \lambda^{n-i} \alpha_{k,i} \varphi_k(u_i)
\]

(7–5)

We can notice that the effective coefficient \( \lambda^{n-i} \alpha_{k,i} \) decays exponentially in the past. Thus, the current weight is less affected by the older observations. This factor is crucial in case of non-stationary classification where the past observation are irrelevant and may even deter the learning ability of the classifier. In addition to learning \( \Omega_k \), the stochastic weights \( \beta_k \) are learned using a non-linear gating function,

\[
\beta_k = \frac{\exp(v_k)}{\sum_{j=1}^{P} \exp(v_j)}
\]

(7–6)

where, the gate parameter \( v_k \) is the intermediate weight at the \( n \)th iteration. The gating function (7–6) maintains convexity in \( \beta \). This requires learning \( v_k \) using an additional update rule:

\[
v_{k,n+1} = v_{k,n} - \mu \frac{\partial \zeta \left[ 1 - \kappa_{\sigma_c}(e_n) \right]}{\partial v_k}
\]

(7–7)
where, $\mu$ is the learning rate. The gradient can be solved as follows:

$$\nabla v_{k,n} = \frac{\partial \zeta}{\partial v_{k,n}} [1 - \kappa_c(e_n)] - \frac{\partial \zeta}{\partial \beta_{k,n}} \frac{\partial \beta_{k,n}}{\partial v_{k,n}} = \frac{\zeta}{\sigma_c^2} \exp \left( -\frac{e_n^2}{2\sigma_c^2} \right) e_n \nabla_y \nabla_s$$

(7–8)

where, $\nabla y = \frac{\partial}{\partial \beta_{k,n}} (y - \hat{y}_n)$, and $\nabla_s = \frac{\partial}{\partial v_{k,n}} \beta_{k,n}$. From (3–21) - (3–25), we have,

$$\nabla v_{k,n} = -\frac{\zeta}{\sigma_c^2} \exp \left( -\frac{e_n^2}{2\sigma_c^2} \right) e_n \left[ \hat{y}_{k,n} - \sum_{l=1 \atop l \neq k}^P \hat{y}_{l,n} \right] \beta_{k,n} [1 - \beta_{k,n}]$$

(7–9)

where, $\hat{y}_{k,n} = \Omega_{k,n-1}^T \varphi_k(u_n)$. Then, the update (7–7) is,

$$\nabla v_{k,n} = v_{k,n-1} + \bar{\mu} \exp \left( -\frac{e_n^2}{2\sigma_c^2} \right) e_n \left[ \hat{y}_{k,n} - \sum_{l=1 \atop l \neq k}^P \hat{y}_{l,n} \right] \beta_{k,n} [1 - \beta_{k,n}]$$

(7–10)

where, $\bar{\mu} = \mu \frac{\zeta}{\sigma_c^2}$ is the step-size. Now, the prediction $\hat{y}_{n+1}$ is given by,

$$\hat{y}_{n+1} = \Omega(u_{n+1}) = \sum_{m=1}^P \beta_m \Omega_m(u_{n+1})$$

$$= \sum_{m=1}^P \beta_m \Omega_{m,n}^T \varphi_m(u_{n+1})$$

$$= \sum_{m=1}^P \sum_{i=1}^n \lambda^{i-1} \beta_{m,i} \alpha_{m,i} \kappa_{m,i}(u_i, u_{n+1})$$

(7–11)

Therefore, we developed an online classifier MxKAC for non-stationary environment by introducing a forgetting factor. By removing the forgetting factor, MxKAC can be easily implemented for stationary signals however, we only treat non-stationary problems in this chapter. Also, note that the use of forgetting factor can also be followed by pruning of past dictionary elements, thus significantly improving the memory requirement and computation burden.

### 7.2 Experiments

In this section, MxKAC is applied to solve non-stationary classification problem, where the underlying data structure changes or drifts over time. This can be caused by changes in some hidden context of the system [52, 73]. In such scenario, the classifier
should be able to learn the discriminant function sequentially from the consecutive batches of data, without any assumptions on the nature or rate of drift. We briefly explore the usefulness of MxKAC in such environment based on a synthetic data.

A synthetic data is generated using a drifting mixture of Gaussian. The experiment is conducted from time step \( t = 0 \) to \( t = 50 \), between which the data is drifting from one spatial location to another, in an unknown rate and pattern. Within this interval, 50 consecutive batches of data are presented for training in an online fashion (data are not repeated), where each batch has 400 training and 200 testing samples. Therefore, the classification performance is evaluated at every 400 samples based on percentage accuracy of classification.

For this experiment, a drifting mixture of Gaussian structures are designed, where the samples are drawn from four different Gaussian distributions with changing means and variances. Let, the two dimensional sample sets \( \mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4 \) are drawn from four Gaussian distributions with means \( \vec{\mathbf{X}}_1, \vec{\mathbf{X}}_2, \vec{\mathbf{X}}_3, \vec{\mathbf{X}}_4 \) and variances \( \mathbf{\Sigma}_1, \mathbf{\Sigma}_2, \mathbf{\Sigma}_3, \mathbf{\Sigma}_4 \), respectively. The sample sets \( \mathbf{X}_1, \mathbf{X}_2 \) are in class 1 and \( \mathbf{X}_3, \mathbf{X}_4 \) are in class 2. Then, the parameters for these Gaussian structures at time \( 0 \leq \tilde{t} \leq \frac{1}{5} T \), where \( T > 0 \) is some time constant, are as follows:

\[
\begin{align*}
\vec{\mathbf{X}}_1 &= \begin{bmatrix} 2 - 10\tilde{t} \\ 5 - 10\tilde{t} \end{bmatrix}; \quad \vec{\mathbf{X}}_2 = \begin{bmatrix} 5 - 5\tilde{t} \\ 8 \end{bmatrix}; \quad \vec{\mathbf{X}}_3 = \begin{bmatrix} 6 + 0.5\tilde{t} \\ 2 + 13\tilde{t} \end{bmatrix}; \quad \vec{\mathbf{X}}_4 = \begin{bmatrix} 6 - 18\tilde{t} \\ 2 + 12\tilde{t} \end{bmatrix} \\
\mathbf{\Sigma}_1 &= \begin{bmatrix} 1 - 3\tilde{t} & 0 \\ 0 & 2 - 0.2\tilde{t} \end{bmatrix}; \quad \mathbf{\Sigma}_2 = \begin{bmatrix} 3 - 10\tilde{t} & 0 \\ 0 & 1 - \tilde{t} \end{bmatrix}; \quad \mathbf{\Sigma}_3 = \begin{bmatrix} 3 - 10t & 0 \\ 0 & 1 - t \end{bmatrix}; \quad \mathbf{\Sigma}_4 = \begin{bmatrix} 3 - 12\tilde{t} & 0 \\ 0 & 2 - 5\tilde{t} \end{bmatrix}
\end{align*}
\]

The result of classification is explained based on the decision boundary and the percentage accuracy. Fig. 7-1 shows the decision boundaries obtained using MxKAC at time \( t = 0 \) and \( t = 50 \). Notice that the data structure drifts from being almost linear separable into a complicated structure requiring non-linear boundary. Thus, the decision boundaries obtained using MxKAC with kernel sizes \( \sigma = \{0.5, 1, 1.5, 2\} \) change...
accordingly. Similarly, the decision boundaries obtained from mono kernel KAC for each of the kernel are shown in Fig. 7-2. It can be noticed that the difference in shapes of these decision boundaries depends on the size of the kernel, the larger the kernel size, the broader the boundary. Fig. 7-3B compares the percentage accuracy of classification averaged over 10 monte carlo trials, using MxKAC and mono kernel KAC. Here the monte carlo trials are obtained by randomly permuting the data set within each incoming batch of data. We can see that MxKAC significantly outperforms KAC as the structure gets more an more complicated. The kernel weights for MxKAC shown in Fig. 7-3A suggest that MxKAC adjusts to the changing data structure by changing the kernels. We can notice that the larger kernel is gradually weighted less and a smaller kernel is gradually weighted more, which is what we would expect. The mean classification accuracy over 10 monte carlo runs is tabulated in Table. 7-1 for different values of time samples. The learning rates for MxKAC are, $\gamma = 1.5$ and $\bar{\mu} = 0.5$ and for KAC is $\bar{\gamma} = 0.1$.

Figure 7-1. Decision boundary obtained for MxKAC using c-loss ($\sigma_c = 0.5$) and multiple kernels kernel sizes $\sigma = \{0.5, 1, 1.5, 2\}$ at time $t = 0$ and time $t = 50$ for A) MxKAC, $t = 0$ B) MxKAC, $t = 50$. 
Figure 7-2. Decision boundaries obtained for KAC using c-loss ($\sigma_c = 0.5$) at time $t = 0$ and time $t = 50$ for different kernel sizes A) $\sigma = 0.5$, $t = 0$ B) $\sigma = 0.5$, $t = 50$ C) $\sigma = 1$, $t = 0$ D) $\sigma = 1$, $t = 50$ E) $\sigma = 1.5$, $t = 0$ F) $\sigma = 1.5$, $t = 50$ G) $\sigma = 2$, $t = 0$ H) $\sigma = 2$, $t = 50$, figures continued in Fig.7-2.
Table 7-1. Mean percentage accuracy of classification of synthetic Gaussian dataset at different time steps.

<table>
<thead>
<tr>
<th>Method</th>
<th>t = 0</th>
<th>t = 10</th>
<th>t = 25</th>
<th>t = 40</th>
<th>t = 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>MxKAC</td>
<td>97.05±0.64</td>
<td>96.75±0.79</td>
<td>95.20±1.27</td>
<td>96.75±1.06</td>
<td>96.60±0.96</td>
</tr>
<tr>
<td>KAC, σ = 0.5</td>
<td>96.70±0.59</td>
<td>96.55±0.75</td>
<td>93.65±1.39</td>
<td>90.35±1.95</td>
<td>84.95±0.95</td>
</tr>
<tr>
<td>KAC, σ = 1</td>
<td>96.89±0.46</td>
<td>96.45±0.59</td>
<td>94.15±1.20</td>
<td>89.85±2.09</td>
<td>84.30±0.78</td>
</tr>
<tr>
<td>KAC, σ = 1.5</td>
<td>96.90±0.52</td>
<td>96.25±0.63</td>
<td>93.95±1.14</td>
<td>89.55±2.14</td>
<td>84.75±0.67</td>
</tr>
<tr>
<td>KAC, σ = 2</td>
<td>97.05±0.72</td>
<td>95.65±0.74</td>
<td>93.60±1.07</td>
<td>89.40±2.05</td>
<td>83.70±0.53</td>
</tr>
</tbody>
</table>

7.3 Summary

We extended KAC to MxKLMS framework using c-loss to achieve a robust classifier for online non-stationary classification, called MxKAC. Classification of streaming non-stationary data is a huge problem because its properties are always changing and also it’s accumulating over time which requires online methods that can classify it over a single sweep. In this chapter, we introduced MxKMC as a suitable solution to address this issue of big data. We showed empirically that given a pool of predefined kernels, MxKAC was able to select or combine kernels as needed in order to classify the changing data structure in an online fashion. The method can be further improved by using ways of constraining the dictionary and testing in more real world data.
Figure 7-3. Result on classification of synthetic non-stationary data using MxKAC showing A) Percentage accuracy of classification on 200 sample length test data obtained after training through every 400 samples from $t = 0$ to $t = 50$. Evolving decision boundaries are shown above the accuracy plot B) Kernel weights for MxKAC from $t = 0$ to $t = 50$. 

---

$\text{Percentage Accuracy}$

- **MxKAC**
- **KAC $\sigma = 0.5$**
- **KAC $\sigma = 1$**
- **KAC $\sigma = 1.5$**
- **KAC $\sigma = 2$**

$x \text{Total Samples} \times 10^4$

$\text{Kernel Weights}$

- **$\sigma = 0.5$**
- **$\sigma = 1$**
- **$\sigma = 1.5$**
- **$\sigma = 2$**

$x \text{Total Samples} \times 10^4$
CHAPTER 8
CONCLUSION

8.1 Summary

This dissertation presents a robust online kernel learning method suitable for supervised learning systems. The robustness was induced by focusing on two major aspects of a learning system that included the kernel map and the cost function. We presented the novel multi-kernel adaptive filtering called mixture kernel least mean square (MxKLMS) algorithm, that extended KLMS to the multiple kernel filtering framework. There are several multiple kernel learning techniques suitable for batch method and some suitable for online KAF but we have shown that MxKLMS is the most effective, theoretically well defined and computationally least expensive. Therefore, MxKLMS is different from the existing multiple kernel learning techniques suitable for batch learning methods, and the existing multi-kernel adaptive filtering methods in both scope and detail.

MxKLMS is a robust multiple kernel method for stationary and more specifically, non-stationary environments. In stationary environment, MxKLMS gives faster rate of convergence compared to mono-kernel KLMS where as in non-stationary environment, it shows an exceptional tracking behavior to adapt to the changing systems, which KLMS is not able to do as it requires a lot more samples before the system changes. A detail study on the convergence analysis and steady state behavior of the method presented in this work corroborates the strong theoretical standing of the method. We were able to establish a key relation between MxKLMS and KLMS steady state EMSE, which suggested that MxKLMS EMSE upper bounded by that of KLMS. To be more specific, it is always less than that of KLMS when more that one kernels are selected and is equal to that of KLMS when single kernel is selected from the pool of predefined kernels. This is a very meaningful theoretical finding that makes MxKLMS more desirable than KLMS. Moreover, we presented the empirical results
where MxKLMS out-performs the existing multi-kernel adaptive filtering methods in various datasets.

It is expected that use of multiple kernels in KLMS would increase the computational burden. To tackle this problem, we constrained the growing dictionary using the quantization technique implemented in quantized KLMS and developed quantized MxKLMS algorithms. We also introduced a new criteria for quantization that involved use of distances in multiple RKHSs rather than just distances in the input space. We were able to attain significant improvement in the computation and memory requirement without having to compromise the prediction accuracy.

By introducing the maximum correntropy criteria in MxKLMS to obtain mixture kernel MCC (MxKMC), we extended the limitation of MxKLMS to tackle more difficult problems where the usual assumptions of Gaussianity and stationarity might no longer be valid. Several examples on the robustness of the ITL based correntropy criteria have been presented in the past for signal processing and machine learning applications. MxKMC was able to outperform MxKLMS and mono kernel KMC in presence of impulsive noise. Such robust online method can have many areas of application such as communications, geosensing, financial data modelling, biomedical engineering, etc.

These filtering methods can be well translated for the classification problems, which we did using the robust correntropy induced loss function (c-loss). C-loss is a recently developed loss function that has been shown to be much more robust than square loss, which we showed in this work as well by presenting kernel adaptive classifier (KAC). Although the architecture of KLMS and KAC is same, it is important to use an appropriate loss function designed for classification thus, the use of c-loss. We extended this to the MxKLMS framework to obtained mixture KAC (MxKAC) showed that it can be applied in solving the non-stationary online classification problems. As it merges two desirable properties of KLMS and KAC, i.e., the tracking ability and the robust classification, it can have various application in data mining real world
problems such as transportation, social media [15], financial monitoring [62], biomedical applications [14, 26], economics [17], etc.

### 8.2 Avenues for Future Work

a) Partitioning an input space and learning a filter for certain part of input space is a very useful approach for dealing with non-stationarity. A particular example of this is regime detection. But an interesting question to answer would be, can this be done in an online fashion where we can locally model filters for different parts of input space such that it can be referred to when needed? This could be a possible future direction for MxKLMS algorithm because of the fact that multiple RKHS models are trained simultaneously with this and that a competitive approach is used to select the best RKHS models at a particular time. So coming up with an online method to generate local RKHS models for local regions in the input space such that it can be referred to when needed, could be an interesting future direction from here.

b) In order to address the problem of non-stationarity better, we can merge the minimum description length (MDL) criteria with both MxKLMS and MxKAC for application in filtering and classification. Use of MDL has been implemented with KLMS for compact dictionary building and robust non-stationary prediction in [77]. It would be an interesting step forward to merge these two concepts as an alternative to the forgetting factor used in MxKAC.

c) This work provides detail theoretical analysis on MxKLMS algorithm but there are still many things that need to be addressed theoretically. For example, given that MxKLMS has an excellent tracking ability, it would be meaningful to analyze its tracking behavior theoretically. The combination filters derived for linear adaptive filters in the past such as [2, 4, 36, 58] can be referred to proceed in this direction. Moreover, convergence analysis of the MxKMC still remains as a future work. Steady state analysis of KMC has been presented in [77] and using similar concepts as in MxKLMS, sufficient condition for convergence and steady state behavior of MxKMC
can be derived. For MxKAC, the tracking behavior of the classifier is another interesting theoretical study, that can be built up on the work presented in [22], which presents KLMS in the Bayesian framework.

d) As a next step forward for MxKLMS, a sparse multi-kernel formulation for KLMS can be achieved by using regularization by the $L_1$-norm for learning the kernel coefficient. The regularization by the $L_1$-norm is a method commonly referred to as the Lasso [66], which leads to sparse solutions. That is, the kernel weight vector with many zeros can be obtained, unlike the weight vector obtained from the competitive softmax gating used in MxKLMS, that has all non-zero elements. Moreover, the method can be further extended using $L_p$-norms regularization for learning the kernel coefficients to obtain various formulations of MxKLMS.

e) Finally, a very appealing step forward would be further study of MxKAC for application in different real world problems. This work develops MxKAC and briefly demonstrates its working on a synthetic data but MxKAC still needs to be implemented and tested in more problems. Given the method is online, is robust to noisy environment and is good in tracking changes in the data distribution, it can stand as a good alternative to the existing online or sequential methods for non-stationary classification. A lot of possible real problems related to data mining and big data can benefit from our approach of robust online kernel adaptive learning.
REFERENCES


BIOGRAPHICAL SKETCH

Rosha Pokharel was born in Minnesota, USA in 1987 and brought up in Kathmandu, Nepal after her family moved back in 1990. She completed her Bachelor of Engineering degree in electronics and communication engineering from Kathmandu University, Dhulikhel, in 2009. She was awarded the University of Florida Graduate School Fellowship in August 2010. She received her Master of Science and PhD in electrical and computer engineering from University of Florida in spring 2013 and spring 2014, respectively. Her research interests are machine learning, statistical signal processing and information theoretic learning.