THESIS

SPARSE REPRESENTATIONS IN MULTI-KERNEL DICTIONARIES FOR IN-SITU CLASSIFICATION OF UNDERWATER OBJECTS

Submitted by

Somayeh Hosseini

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2017

Master's Committee:

Advisor: Ali Pezeshki
Co-Advisor: Mahmood R Azimi-Sadjadi

Edwin Chong
Jie Luo
Michael Kirby

ABSTRACT

SPARSE REPRESENTATIONS IN MULTI-KERNEL DICTIONARIES FOR IN-SITU CLASSIFICATION OF UNDERWATER OBJECTS

The performance of the kernel-based pattern classification algorithms depends highly on the selection of the kernel function and its parameters. Consequently in the recent years there has been a growing interest in machine learning algorithms to select kernel functions automatically from a predefined dictionary of kernels. In this work we develop a general mathematical framework for multi-kernel classification that makes use of sparse representation theory for automatically selecting the kernel functions and their parameters that best represent a set of training samples. We construct a dictionary of different kernel functions with different parametrizations. Using a sparse approximation algorithm, we represent the ideal score of each training sample as a sparse linear combination of the kernel functions in the dictionary evaluated at all training samples. Moreover, we incorporate the high-level operator's concepts into the learning by using the in-situ learning for the new unseen samples whose scores can not be represented suitably using the previously selected representative samples. Finally, we evaluate the viability of this method for in-situ classification of a database of underwater object images. Results are presented in terms of ROC curve, confusion matrix and correct classification rate measures.

ACKNOWLEDGEMENTS

I would like to express sincere thanks to my advisor Dr. Ali Pezeshki and my co-advisor Dr. Mahmood Azimi for giving me the opportunity to work on this project. Without their continuous advice and support the completion of this work would have never been accomplished.

I am also thankful to my committee members, Dr. Edwin Chong, Dr. Rockey Luo and Dr. Michael Kirby, for their time and assistance.

I would like to thank Information System Technologies Inc. (ISTI) for providing us with the dataset and specially to its staff members Nick Klausner and Neil Wachowski for supporting us throughout this work.

Finally I would like to thank my family and friends for their emotional support.

This thesis is typset in LaTeX using a document class designed by Leif Anderson.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. PROBLEM STATEMENT AND MOTIVATION

Underwater object classification is problem of classifying images of underwater objects into two categories of mine-like and non-mine-like objects. There are various causes that make this classification problem difficult to be tackled including different target (mine-like object) shapes and orientations, different conditions in the environments from which the images are collected, presence of spatial-varying clutter, different depths in which the targets are buried in the seafloor and the obstruction of the targets by seafloor sand formations and vegetation that will result in high false alarm rates. A great deal of research has been devoted [9] in recent years to develop computerized methods to address such issues for underwater applications.

Many of the existing state-of-the-art methods for pattern classification rely on using the kernel machines [10] that are useful in solving different machine learning problems. The idea behind kernel machines is to non-linearly map the original input space into a higher dimensional feature space and find the similarity measure (inner product) between the points in this higher dimensional feature space. Using the kernel trick, we can simply replace the inner product in the original space with a kernel function representing the similarity function in the higher dimensional space, without having to find the explicit non-linear mapping function.

Although the advances in the kernel-based methods have significantly impacted on the machine learning field, the selection of the kernel functions and their corresponding feature spaces, that plays an important role in the performance of the kernel-based methods, is still challenging for the users. In most of the kernel-based learning algorithms a parametrized class of kernels is selected and the parameters of the kernels are adjusted using cross validation. However, algorithms

that rely on using a single kernel function [9] are not flexible enough for classification due to their low degrees of freedom. Consequently, the methods such as [1]- [8] that allow for automatic selection of kernel functions based on training data (multiple kernel learning methods) have recently become more desirable. Using the multiple kernel functions, the number of available degrees of freedom is increased and as a result the classes can be better represented while the outliers are discarded for a more reliable classification. In the current work we have proposed a new multiple kernel learning method that relies on using a kernel dictionary and sparse representation theory as opposed to the single kernel methods. The proposed method also makes use of the expert operator's feedback through in-situ learning in order to maintain the classification performance in new environmental conditions as this is one of the problems encountered in real mine-hunting scenarios. In the following sections we first give a literature review on kernel learning methods and then introduce our multi-kernel method.

1.2. LITERATURE REVIEW ON KERNEL LEARNING METHODS

In [1] the authors develop a multiple kernel learning approach using semidefinite programming (SDP) [11] for binary classification. They applied the method to a partially-labeled dataset in order to predict the labels for the unlabeled portion. The dataset is implicitly embedded into a higher-dimensional feature space in which the pairwise inner products between the data points are calculated (implicitly using a kernel function) and contained in a symmetric positive semidefinite matrix called kernel or Gram matrix. The kernel matrix in fact defines the relative positions of all data points in the embedding space. As a measure of separation between the labeled points, they used an optimization criterion (cost function) such as support vector machines (SVM) hard margin or 1 and 2-norm soft margin over the labeled samples. However, instead of using a fixed kernel matrix, they used a library of known kernel matrices to design a kernel matrix (corresponding

to the whole dataset) that optimizes the cost function. For this reason, they restricted the kernel matrix to the set of positive semidefinite matrices with bounded trace that can be written as a linear combination of the kernel matrices in the library. They posed the problem of joint optimization of the coefficients of such combination and the coefficients of the classifier and showed that it is a semidefinite programming problem. By constraining the linear coefficients to be positive, this optimization problem can be reduced further to a quadratically-constrained quadratic program (QCQP) that is a special case of the SDP. Moreover, they considered the optimization problem that searches for the kernel matrix which has the maximum alignment with a set of labels among a set of linear combinations of known kernel matrices. Here, the alignment is the cosine of the angle between the kernel matrix and the rank one matrix built using the labels vector (ideal kernel matrix).

Although the QCQP problem in [1] is a convex optimization problem, it becomes intractable as the number of learning samples or kernels grow. This allows the existing convex optimization toolboxes to solve this problem only for small number of data samples and kernels. Therefore, applying the more advanced solutions such as sequential minimal optimization (SMO) techniques is essential for the large-scale problems. However, the multiple kernel learning, while being a convex problem, is non-smooth; that means the cost function in this problem is non-differentiable. The non-smoothness of the cost function causes the simple local descent algorithms such as SMO to either not converge or converge to a wrong value. This new problem is addressed in [2] in which the authors proposed a novel dual formulation of the QCQP as a second-order cone programming (SOCP) problem. They called this new formulation, that is similar to the classical maximum margin SVM formulation, the *support kernel machine (SKM)* due to the fact that the Karush-Kuhn-Tucker (KKT) conditions in this formulation detect not only the support vectors but also the so called ”support kernels” which are active in the linear combination of kernels. The SKM

formulation that represents exactly the multiple kernel learning problem of [1], enables them to derive a smoothed formulation of the problem through the *Moreau-Yosida (MY) regularization* [21] that is consistent with the SMO techniques. Their evaluation results illustrated that their proposed algorithm is more efficient compared to general-purpose interior-point methods.

An alternative mathematical framework for kernel learning is developed in [3] using the ideas in [1]. Given a kernel function and a traning dataset, the authors of [3] first defined a function called empirical quality functional that measures how well the kernel function matches to a specific dataset. Although optimizing over a large enough class of kernels can minimize this quality functional, it leads to a poor generalization performance due to optimizing too much and overfitting. Therefore, to address this problem they regularized the empirical quality functional by introducing a so called Hyper reproducing kernel Hilbert space (Hyper-RKHS) of functions on the kernels itself and optimized the regularized quality functional over this Hyper-RKHS. In fact the only things that tell the Hyper-RKHS and a normal RKHS apart are the compounded index set of Hyper-RKHS and the additional condition of symmetry in the first two arguments of the kernel that generates it, namely hyperkernel. They provided examples of hyperkernels and general instructions for constructing them. They defined the regularized quality functional by adding the regularization term, that is the squared RKHS norm of the kernel in Hyper-RKHS space weighted by a regularization constant, to the empirical quality functional. They selected the specific example of regularized risk functional (commonly used in SVMs) as the empirical quality functional and considered only positive semidefinite kernel matrices to confirm the positive definiteness of the kernel function. They proved the representer theorem for RKHS/Hyper-RKHS which says the optimal kernel minimizing the regularized risk functional/ regularized quality functional can be expressed as a finite kernel/hyperkernel expansion on the input data. Using this theorem, they wrote the regularization terms in the regularized quality functional in quadratic form. They posed

the optimization problem based on new formulation of the objective function, and derived its dual formulation that can be represented as a SDP. They posed several hyperkernel optimization problems derived from some popular machine learning problems such as binary classification (linear or quadratic SVM), novelty detection (single class SVM) and regression. For each of the mentioned problems they defined a suitable loss function in the regularized quality functional and derived the corresponding SDP.

In [4] the authors studied the problem of searching through a compact and convex set of kernels for an optimal kernel that estimates a real-valued function given a training dataset. A commonly used approach of solving this problem is to optimize a regularization functional that trades off the learning error quantized by a loss function and the smoothness of the solution measured by the regularization term in a Hilbert space of functions. They let the Hilbert space to be a RKHS to be able to use the result of representer theorem [3] for the representation of regularization functional. They used this representation of the functional as a design criterion to find the optimal kernel function. They also showed that the minimizer kernel exists if the loss function is continuous and the set of kernels is a compact and convex subset of kernel functions corresponding to positive definite kernel matrices. The two useful examples of a convex and compact set of kernels are the set of convex combinations of a finite number of kernels belonging to set of kernels with positive definite kernel matrices and the convex hull of a predefined set of kernels parametrized by a compact set. They proved that this problem is a convex optimization problem for a variety of regularization functionals and they provided the explicit function expression and the optimal solution for the specific case of square loss regularization functional. Moreover, they proved that the optimal kernel is always expressed as a convex combination of at most $m + 2$ basic kernels (m is the number of training samples), even though the primary search span for the kernels is an uncountable set.

The authors of [5] started with the multiple kernel learning problem for binary classification initially proposed in [1] and reformulated it as a semi-infinite linear program (SILP). They then extended this problem to a general setting by formulating the SILP for an arbitrary strictly convex and differentiable loss function. They tried different loss functions such as soft-margin loss, the ϵ-insensitive loss and the quadratic loss in this general SILP and analyzed them. They developed an algorithm for solving SILPs called *wrapper* algorithm that iteratively solves a single kernel SVM problem, that can be solved with many existing toolboxes, while the number of constraints grows at each iteration. The advantage of this algorithm is that it is valid for every kernel function and for a large class of loss functions. They improved this algorithm by developing a significantly more efficient algorithm called *chunking* algorithm that is able to optimize the SVM multipliers and the kernel coefficients simultaneously. Finally, they showed how the SVM training (including multiple kernel learning) can be sped up when the mapping into the kernel feature space is known and sparse.

The authors of [6] proposed a Difference of Convex functions Programming (DC-programming) algorithm for kernel learning. Similar to [3] they optimized a convex regularized functional over the set of convex combinations of some basic kernels. What distinguishes this approach from other related works is that it uses a continuous parametrized family of kernels as the basic kernels. In other words, the objective function is optimized over a continuously parametrized set of basic kernels such as a Gaussian family whose variance is an arbitrary positive value or a family of polynomial kernels of arbitrary degree. Their resulting formulation for kernel learning includes a minimax optimization problem and a greedy algorithm. While this optimization problem is not convex, it can be categorized as the larger class of DC programs and be solved using the recent results of DC optimization theory.

Motivated by the works in [1], [2], [5], the authors of [7] introduced another formulation for multiple kernel learning. They started by changing the mixed-norm regularization in the primal formulation of [2] with a weighted ℓ_2-norm regularization. They also managed the sparsity of the linear combination of the kernels through an ℓ_1-norm constraint on the kernel coefficients. This leads to a new formulation for multiple kernel learning that is now smooth and convex and they developed a simple gradient-descent-based algorithm to solve it iteratively. They extended their algorithm that has been initially developed for binary classification to some other popular SVM-based problems such as SVM regression, one-class SVM and pairwise multiclass SVM. They also tried to generalize further by considering any other convex loss function that can be fitted within their algorithm. They proved through empirical results that their algorithm is more efficient compared to the similar approach in [5]. Moreover, by constraining the kernel coefficients to be sparse they can interpret the resulting decision function more easily.

The authors of [8] showed that for a variety of kernel-based learning algorithms the problem of learning the kernel and the parameters of the algorithm at the same time can be posed as a tractable convex optimization problem that can be solved efficiently using interior-point methods. They first wrote a very general optimization problem that formulates a wide variety of kernel-based learning algorithms and consists of an objective function and some constraints. Using an extension of the representer theorem [3], they reformulated the objective and constraints to be a function of the kernel (Gram) matrix obtained from kernel function at training samples. They wrote the general kernel learning problem corresponding to this new formulation and optimized it over a set of Gram matrices obtained from a convex set of positive semidefinite kernel functions that should be selected properly for a good generalization performance. Using the change of variables, they showed that this optimization problem is convex and can be solved efficiently since the first and second derivatives of its objective and constraints can be evaluated efficiently. They considered some special

cases of this general formulation of the problem such as the regularized loss functional and hard margin SVM and derived their equivalent convex kernel learning problems.

The authors of [9] used an adaptable multi-class single-kernel machine for the purpose of image retrieval and classification. Similar to probabilistic neural network [12], their classification system consists of pools of neurons each representing a class or a sub-class. Each pool of neurons consists of a single kernel function that is centered around a set of feature vectors belonging to the corresponding class. A linear combination of the kernel functions at each pool represents the score function for that pool (class). They trained their system using either model-reference or relevance feedback learning. In both cases the network parameters are obtained implicitly using regularized least squares approach. In the model-reference learning they trained the samples for each pool in a batch-mode to find the hidden information of classes. In addition, they introduced a single-round relevance feedback learning as an alternative in order to adaptively modify the score functions. The relevance feedback learning brings the high-level expert operator's feedback into the learning process by acquiring the operator to visually provide the desired scores for the retrieved images. They derived a recusrsive equation to update the score functions for the images that recieve the operator's feedback. They also proved that their proposed relevance feedback learning is stable in the sense that it does not change the score functions for the previously-trained images. Moreover, to avoid over-fitting problems and reduce the computational complexity, they developed an unsupervised selective sampling method based on the Fisher information matrix that selects the most informative samples for each relevance feedback learning iteration.

1.3. MULTIPLE-KERNEL IN-SITU CLASSIFICATION USING SPARSE REPRESENTATION THEORY

In the current work, we propose a general kernel-based multi-class in-situ classification system using an expanded dictionary of kernel functions and sparse representation theory and apply it to the specific problem of classifying underwater objects images. As depicted in Figure 1.1, the block diagram of this system consists of several units. In the feature extraction unit we extract the entropy-based feature vectors from the localized snippet images of underwater objects. In the baseline training unit we first construct a vector-valued score function as a linear combination of various nonlinear kernel functions (selected from a kernel dictionary), each of which measures the similarity between a data sample to be classified and the representative samples from a class. The representative samples for different classes are selected automatically through a sparse approximation algorithm such as [19], by forcing the representation of the vector-valued score function to be sparse in a dictionary of kernel functions with respect to all data samples from different classes. After finding the sparse weight matrix, we use it to compute the vector-valued score function for a new data sample $\underline{q}$ with unknown class label. When the entries in the score vector do not assign the sample $\underline{q}$ to any of the classes with high enough confidence, then it is the time for the in-situ learning unit to come into play. In the case of in-situ learning we need to update the score function by learning the new training samples one by one or in groups using the labels provided by an expert user. However, the updating must be done such that the scores for all previously learned data samples are not influenced at all. This is enabled by updating the representation vectors in the null space of previously selected kernel functions in a sparse fashion. Additionally, we need to decide whether the new samples need to be added to the representative sample sets for different classes. For the first case, we consider to learn the new samples by simply updating the weight matrix in the score function in a sparse fashion without changing the dictionary while maintaining

the previously learned scores. For the second case, we go further by adding the new samples to the dictionary while maintaining the previously learned scores, that simply means we expand the kernel dictionary.

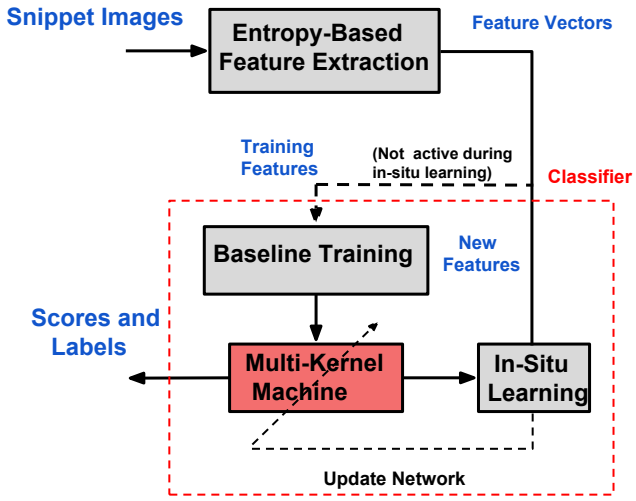

FIGURE 1.1. Multi-kernel in-situ classification system

This system is similar to [9] due to its general multi-class structure and bringing in-situ learning into play. Moreover, it is comparable to the kernel learning methods proposed in [1]- [8] in the sense that it does not rely only on a single kernel.

The remainder of this thesis is organized as follows. In Chapter 2 we describe the proposed theoretical framework for our classification algorithm in three sections: multi-kernel score function, sparse representation of the score function using training data and finally updating the sparse representation using the in-situ data. In Chapter 3 we present the results of evaluating the proposed theoretical framework for classification on a dataset of underwater images. Finally, we give a summary of our work and the evaluation results along with the possible improvements or future work in Chapter 4.

CHAPTER 2

MULTIPLE-KERNEL CLASSIFICATION SYSTEM USING SPARSE REPRESENTATION FRAMEWORK

2.1. INTRODUCTION

In machine learning, classification is the problem of identifying a new sample based on a training set of samples whose class membership is known. Prior to each classification process, the feature vectors are extracted from the original data samples in order to make that classification problem easier to be tackled. The feature vectors are the lower dimensional representations of the data to be classified. As the dimensionality of the data used to fit a classifier increases, the quality of the fitted model deteriorates and as a result the generalization error increases. This phenomenon is known as *curse of dimensionality* [22]. Therefore, by removing the redundant (noise) features and extracting the most salient features of the raw image data we can lower the risk of curse of dimensionality. In addition, this enable us to deal with a smaller-sized data and as a result perform the same task but with a lower amount of time and complexity. A classifier can now be fitted to the feature vectors extracted from the training samples by setting a decision rule that divides those feature vectors into p decision regions or classes. The classification decision for an unseen observation vector is then made by comparing p scalar functions known as score functions. The values of the score functions for a given observation vector decide to which class it should be assigned. The observation vector belongs to the class for which it yields the maximum value of the score function.

Different types of classifiers can be designed by generating different types of score functions. For a variety of classification problems the kernel-based score functions are used. The kernel-based classification system has been one of the most promising classification methods to this date.

However, the selection of kernel functions and their parameters still remains to be a challenging issue of the kernel-based methods. This issue has motivated many researchers to develop multiple kernel approaches. Most of the current multiple kernel learning methods such as [1]- [8] construct a kernel model that is a linear or nonlinear combination of some predifined base kernel functions. The kernel learning algorithm then learns the weighting coefficients of each base kernel, rather than finding the optimal parameters of a single kernel. A multiple kernel learning model benefits from more degrees of freedom and flexibility and is specifically suitable for multi-modal data such as sonar images. We may require a different notion of similarity (kernel) function to represent each set of features in multimodal data. Therefore, we can represent each mode of the data by a different kernel function and then linearly combine these functions to represent the whole data. Although the current multiple kernel learning approaches automate the kernel selection process, they are computationally expensive and less interpretable than single kernel approaches.

In the current work, we have developed a classifier with the multi-kernel-based score functions that is simpler and more interpretable than the other multiple kernel approaches. The proposed multi-kernel classification system that is shown in Figure 1.1 builds a large dictionary of base kernel functions and makes use of sparse representation framework to automatically select the most representative kernel functions for representing the subspace spanned by our dataset. The sparse representation of score functions of the training samples leads to a more interpretable model and enables us to manage the complexity of the model and consequently lower the overfitting risk. Another capability of our classification system is to take advantage of an expert user's feedback through in-situ (online) learning. During the in-situ learning the parameters of the model are changed in order to be able to represent the scores of a new set of samples in addition to the previously-learned samples. The remainder of this chapter is divided into 3 parts in which we

describe: the score function design, baseline training and in-situ training of our proposed classification system, respectively.

2.2. MULTI-KERNEL SCORE FUNCTIONS

In this section we describe how we construct the score functions for our multi-kernel classifier. Let $\mathcal{X} = \{\underline{x}_j \in \mathbb{R}^L, j \in [1, M]\}$ be the database of our training samples, $\underline{x}_j$'s, belonging to p different classes $C_1, \ldots, C_p$. Without loss of generality, we assume that the data samples are arranged in blocks, with the first block of data samples corresponding to all samples from C_1, the second block corresponding to all samples from C_2, and so on. Our goal is to build a vector-valued score function $\underline{s} : \mathbb{R}^L \to \mathbb{R}^p$ as a linear combination of some similarity matching functions between a data sample $\underline{q}$ and every sample in the database $\mathcal{X}$. A typical similarity function is the Euclidean distance measure $k(\underline{q}, \underline{x}_j) = (\underline{q} - \underline{x}_j)^T(\underline{q} - \underline{x}_j)$. Equivalently, we can use the inner product function $k(\underline{x}_j, \underline{q}) = \underline{x}_j^T \underline{q}$ that is a decreasing similarity measure. If the data samples are linearly separable in the original input space and an optimal hyperplane can separate them we calculate the inner product in the original input space. However, there are the cases where the data samples are not linearly separable in the original input space. In such cases we use a nonlinear mapping function $\Phi : \mathbb{R}^L \to \mathbb{R}^{N_f}$ with $N_f > L$ to map the data samples to the higher dimensional feature space. The non-linear mapping function adds some high-level features to the original input space. We hope that by mapping to the higher dimensional feature space the samples can be separated linearly using an optimal hyperplane, because we will be able to catch higher-level concepts in this feature space. The inner product in the higher dimensional feature space becomes

$$k(\underline{x}_j, \underline{q}) = \Phi^T(\underline{x}_j)\Phi(\underline{q}).$$

By making use of the kernel trick we do not even need to explicitly find the mapping function Φ, that can be infinite-dimensional sometimes. There are many symmetric functions $k(\underline{x}_j, \underline{q})$ namely

kernel functions [10] that can be formulated as an inner product in the higher dimensional feature space. The kernel concept allows us to replace the simple inner product function with some other choice of kernel function in the input space. In the current work we use a dictionary of these kernel functions to construct our scoring function for each class.

Let $\mathcal{K} = \{k(\cdot, \underline{x}; \theta) \mid k \in \mathcal{F},\ \theta \in \Theta, \underline{x} \in \mathcal{X}\}$ denote our dictionary of kernel functions, where $\mathcal{F}$ is a set of kernel families and Θ is a set of parameters for the kernel families in $\mathcal{F}$. Each kernel element is a function $k(\cdot, \underline{x}; \theta) :\ \mathbb{R}^L \to \mathbb{R}^+$ which measures the similarity between a point in $\mathbb{R}^L$ and a data point in $\mathcal{X}$. For example, suppose $\mathcal{F}$ is the set of RBF Gaussian, Quartic and Laplacian kernel families. Then, the elements of the kernel dictionary $\mathcal{K}$ consist of RBF Gaussian kernel functions

$$k(\underline{q}, \underline{x}; \sigma^2) = e^{-\|\underline{q}-\underline{x}\|_2^2/2\sigma^2},$$

with mean vectors $\underline{x} \in \mathcal{X}$ and variances $\sigma^2 \in \Theta$, Quartic kernel functions

$$k(\underline{q}, \underline{x}; \sigma^2) = (1 - \|\underline{q} - \underline{x}\|_2^2/\sigma^2)^2,$$

with varying parameters $\sigma^2 \in \Theta$ and Laplacian kernel functions

$$k(\underline{q}, \underline{x}; \sigma^2) = e^{-\|\underline{q}-\underline{x}\|_2/\sigma}$$

with centers $\underline{x} \in \mathcal{X}$ and scale parameters $\sigma \in \Theta$.

We construct a vector-valued score function as a linear combination of the dictionary elements in $\mathcal{K}$ as the following form:

$$\underline{s}(\underline{q})^T = [s_1(\underline{q}), \ldots, s_p(\underline{q})] = \underline{k}(\underline{q})^T W$$

where W is an $N \times p$ weight matrix, $\underline{k}(\underline{q})^T$ is a ($N = |\mathcal{F}| \times |\mathcal{X}|$)-dimensional row vector of all elements in $\mathcal{K}$ and the entries in the kernel vector $\underline{k}(\underline{q})^T$ are arranged in blocks in the same

fashion that data samples are arranged in $\mathcal{X}$. That is, the first block has all the kernel functions that are centered around samples from class C_1, the second block has all the kernel functions that are centered around samples from class C_2, and finally the last block has all the kernels that are centered around samples from class C_p. The i^{th} entry $s_i(\underline{q})$ in $\underline{s}(\underline{q})^T$ is the score with which the classifier believes the data sample $\underline{q}$ belongs to class C_i, $i = 1, 2, \ldots, p$. Here the score function for each class is built using the kernel functions centered around samples from all classes as opposed to the method proposed in [9] where the score function for each class is built using only the kernel functions centered around the samples from the same class.

Using the constructed score function $\underline{s}(\underline{q})^T$, our goal is now to represent the ideal scores of our training samples in $\mathcal{X}$ as a *sparse* linear combination of kernel functions in the kernel vector $\underline{k}(\underline{q})^T$. The sparse representation of the score functions automatically identifies the sample elements in $\mathcal{X}$ and the kernel types and parameters that best represent different classes in a compact and parsimonious fashion. The sparsity enables the classifier to use only a partial number of samples for representing each class and as a result reduces the required memory size and computing load, and avoids overfitting problems as well. In addition, the interpretation of the score function for each sample becomes easier using the sparsity concept. In the next section, we will introduce how a sparse representation of the score function can be obtained from a set of training samples with known class labels.

2.3. SPARSE REPRESENTATION OF SCORE FUNCTION USING TRAINING DATA

In this section we design our classifier by training it with the training samples in the database $\mathcal{X}$. Our goal is to represent the known labels of the training samples as a sparse linear combination of the kernel functions that we have calculated by evaluating the dictionary elements at the same training samples. Figure 2.1 shows a block diagram of our classification system that shares some

similarities to [9] and can be interpreted as a more general form of it. The system consists of p classes to each a subset of training samples belong. For a given sample $\underline{q}$ the score created by each class is obtained by calculating the weighted sum of kernel functions in the dictionary $\mathcal{K}$ evaluated at the sample $\underline{q}$. The unique weight vector for each class determines which kernel functios are active in building the score function for that class. However, the kernel functions from all classes can potentially participate in building the score function for a given class because the dictionary elements are centered around training samples from all classes. This is illustrated in Figure 2.1 where all of the classes are taking part in building the score function for a given class. Moreover, by designing the weight vectors to be sparse, only a small number of kernel functions that best represent the scores of classes are selected automatically. The term sparse means that the number of non-zero entries in a vector is so small. The sparse approximation algorithm [13] finds the weights for the kernel functions that are centered around selected representative samples while it returns zero for the other weights. As a result of dealing with sparse vectors, we will need less time for calculations, have less data to store in the memory as only the positions and the values of non-zero entries need to be recorded and we will not encounter the consequences of over-training of the classification system. Using the designed sparse weight vectors we can hopefully calculate the scores of a new sample that has not been trained and find out to which class it belongs. In the rest of this section we show how to design the weight vector for each class such that the desired lables for training samples are met and the resulting weight vector will be sparse as well.

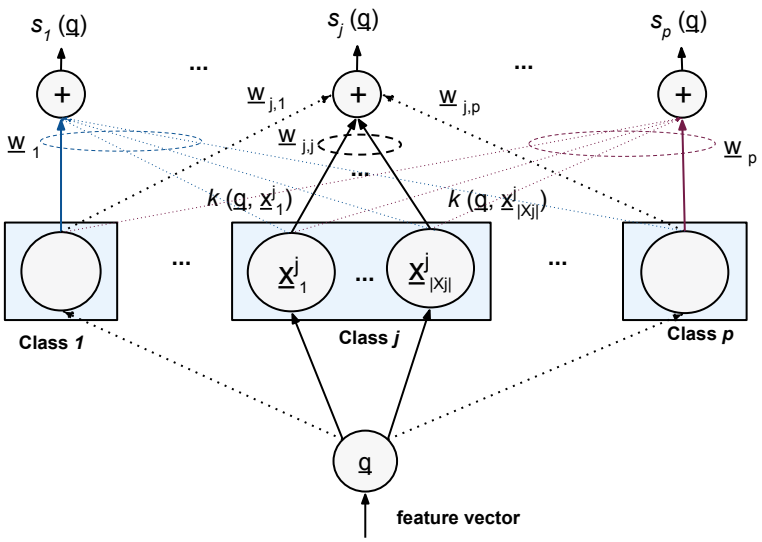

FIGURE 2.1. The multi-kernel classification system

Let $\mathcal{X}_i$ denote a subset of training samples that represent class C_i, $i = 1, 2, \ldots, p$ such that $\mathcal{X} = \bigcup_{i=1}^{p} \mathcal{X}_i$ and $M = |\mathcal{X}| = \sum_{i=1}^{p} |\mathcal{X}_i|$. Let $\underline{\ell}_i^T$ denote the $1 \times p$ label (ideal score) vector for class C_i, with a unit entry at position i and zero entries at all other positions. If a sample $\underline{q}_i \in \mathcal{X}_i$ from class C_i is given to the score function, then we expect the score function to return exactly $\underline{s}(\underline{q}_i)^T = \underline{\ell}_i^T$ or a close approximation to it, that is

$$\underline{\ell}_i^T = \underline{s}(\underline{q}_i)^T + \underline{e}_i^T,$$

where $\underline{e}_i^T$ is a small error vector. Therefore, if we plug all the training samples in $\mathcal{X} = \bigcup_{i=1}^{p} \mathcal{X}_i$ in the score function $\underline{s}^T(\cdot)$, we will have

$$L_0 = K_{00} W_0 + E_0. \tag{2.1}$$

where the rows represent the score equations for the training samples.

Here

$$L_0 = \begin{bmatrix} \underline{\ell}_1^T \\ \vdots \\ \underline{\ell}_1^T \\ \vdots \\ \underline{\ell}_p^T \\ \vdots \\ \underline{\ell}_p^T \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \in \mathbb{R}^{M \times p}$$

is the matrix of desired labels,

$$K_{00} = \begin{bmatrix} \underline{k}(\underline{x}_{1,1})^T \\ \vdots \\ \underline{k}(\underline{x}_{1,|\mathcal{X}_1|})^T \\ \vdots \\ \underline{k}(\underline{x}_{p,1})^T \\ \vdots \\ \underline{k}(\underline{x}_{p,|\mathcal{X}_p|})^T \end{bmatrix} \in \mathbb{R}^{M \times N}$$

is the matrix of kernel vectors evaluated at data samples in $\mathcal{X}$, where $W_0 \in \mathbb{R}^{N \times p}$ is a weight matrix, and $E_0 \in \mathbb{R}^{M \times p}$ is a fitting error matrix, which we wish to keep its elements small and $\underline{x}_{i,j} \in \mathcal{X}_i$ is the j^{th} sample from class C_i. For example, if we only use the Gaussian Kernel family with variances σ_1^2 and σ_2^2, then

$$\underline{k}(\underline{x}_{1,1})^T = [1 \quad 1 \quad e^{-\|\underline{x}_1 - \underline{x}_2\|^2/2\sigma_1^2} \quad e^{-\|\underline{x}_1 - \underline{x}_2\|^2/2\sigma_2^2} \quad \cdots \quad e^{-\|\underline{x}_1 - \underline{x}_M\|^2/2\sigma_1^2} \quad e^{-\|\underline{x}_1 - \underline{x}_M\|^2/2\sigma_2^2}].$$

The objective is to design the weight matrix $W_0 \in \mathbb{R}^{N\times p}$ that minimizes the lengths of the columns of error matrix E_0 while keeping the columns of W_0 sparse. However, (2.1) consists of the following p columns that we need to solve separately in order to design each column of W_0

$$\underline{l}_{0i} = K_{00}\underline{w}_{0i} + \underline{e}_{0i}, \text{ for } i = 1, 2, \ldots, p, \tag{2.2}$$

where $\underline{w}_{0i}$, $\underline{l}_{0i}$ and $\underline{e}_{0i}$ are the i^{th} columns of W_0, L_0 and E_0 respectively. The i^{th} column in (2.2) consists of M equations with N unknowns and since $M < N$ it is an underdetermined system of equations with many possible solutions. However, we are interested in the sparsest solution where the total number of non-zero entries of $\underline{w}_{0i}$ is only a fraction of the total number of measuments M. The sparsity constraint regularizes our initial ill-posed optimization problem so that we can achieve a unique solution . Therefore, the new problem becomes to recover high dimensional sparse vectors $\underline{w}_{0i}, i = 1, 2, \ldots, p$, using only a small number of noisy linear measurements. By writing the equations in (2.2) as quadratic constraints and adding the sparsity requirement as an ℓ_1- norm, we can pose our problem as the following set of ℓ_1- norm minimizations each for a column of W_0

$$\begin{aligned} &\min \ \|\underline{w}_{0i}\|_1 \\ &\text{s.t.} \\ &\|\underline{l}_{0i} - K_{00}\underline{w}_{0i}\|_2 \leq \epsilon, \text{ for } i = 1, 2, \ldots, p, \end{aligned} \tag{2.3}$$

where $\epsilon > 0$ is some prespecified error margin. The problems of this sort are called sparse approximation problems [13] and can be solved using one of several available algorithms, e.g., orthogonal matching pursuit (OMP) [19],[20], basis pursuit denoising (BPDN) [16],[17], or LASSO [14],[15]. The sparse approximation algorithms have other applications including but not limited to feature

extraction and denoising. The orthogonal matching pursuit algorithm that is used in this work is described in Appendix A. The sparse solution to (2.3), W_0^*, if exists, is called the sparse approximation of L_0. The locations of the nonzero entries in the columns of W_0^* determine indices of the representative data samples in $\mathcal{X}$, whose kernel functions are chosen to express the ideal training scores.

After the weight matrix W_0^* is obtained, we can use it to calculate the score vector $\underline{s}(\underline{q})^T$ of a new sample $\underline{q}$ out of the dataset in order to assign it to a certain class. However, the case that some entries in the score vector $\underline{s}(\underline{q})^T$ are highly close to each other may happen. In such cases, we can not decide to which class the new sample with unknown label belongs, which means we have to use the feedback of a user to update the score functions in order to assign the new sample to a class with high enough confidence. In the following section we show how the sparse representation matrix W_0^* and the dictionary matrix K_{00} can be updated for in-situ learning as new data samples become available, in a sparse fashion without changing the scores for samples which have already been learned.

2.4. UPDATING SPARSE REPRESENTATIONS FOR IN-SITU LEARNING

The classification system designed in the previous section returns almost perfectly the ideal scores of the training samples. These training samples are extracted from a certain environment with specific properties. For example, in underwater target classification case the samples are the sonar images taken from a patricular ocean floor. However, there are cases where we are given samples from a totally different (geographical) environment with different properties (images from another ocean floor with different water depth and background) to classify. Due to the change in the environment, the feature vectors extracted from the new samples might be too different from the training feature vectors that they do not fit in the trained system. Consequently, the classifier will

not be able to make a correct decision about the new sample. In this case, either the new sample is assigned to a wrong class or the score values are so close that the system can not classify the sample with a high enough confidence. One of the solutions in this case is to somehow incorporate an operator's feedback namely relevance feedback into the learning process by asking an operator (who is already familiar with the new objects) to provide the learning system with the true scores and update the system based on these new information. We call this method of learning that makes use of relevance feedback, in-situ learning. We should keep in mind that only a limited number of samples can be used for in-situ learning. One of the reasons is that the operators will probably lose patience if they are queried for feedback for many times. Another reason is to avoid the overfitting of the system with too many (possibly redundant) in-situ samples. Therefore, if an in-situ sample is classified correctly and with a high enough confidence, we do not need to query the operator about that in-situ sample to update the classification system. In the rest of this section we show how we applied the in-situ learning to our classification system.

Suppose that training is done and a sparse weight matrix W_0^* is found. Let $\underline{q}$ denote a new data sample (not in $\mathcal{X}$) with unknown class label. The score vector for $\underline{q}$ is given by

$$\underline{s}(\underline{q})^T = \underline{k}(\underline{q})^T W_0^*.$$

If one entry $s_i(\mathbf{q})$ in the score vector is larger enough than the others, then we assume that the new sample $\underline{q}$ is adequately represented by the representative data samples selected by the sparse weight matrix W_0^*. In this case, we do not need to change the weight matrix W_0^* or the kernel dictionary $\mathcal{K}$. However, if the entries in the score vector $\underline{s}(\underline{q})^T$ do not assign the sample $\underline{q}$ to any of the classes with high enough confidence, then the operator needs to be queried for the true label of the sample and the score function $\mathbf{s}^T(\cdot)$ needs to be updated.

Updating the score function can be performed for new samples one-by-one or in groups. However, it must be done such that the scores for all previously learned data samples are not affected at all. We also need to decide whether or not the new samples need to be added to the representative sample sets for different classes. That is, we need to decide whether or not to add new elements to the kernel dictionary $\mathcal{K}$, where these elements are centered around the new data samples. We consider two cases. In the first case, the new sample can be learned by simply updating the weight matrix in the score function in a sparse fashion (while maintaining the previously learned scores) without changing the dictionary. In the second case, learning the labels for the new samples in a sparse fashion (while maintaining the previously learned scores) requires expanding the kernel dictionary.

2.4.1. CASE 1: IN-SITU LEARNING WITHOUT SAMPLE ADDITION. Let L_1 denote the label matrix for new samples for which the score function needs to be updated. Each row of L_1 is a label vector with zeros at all entries, except for a one at the entry corresponding to the class to which the operator believes the new sample $\underline{q}$ belongs. The simplest way to formulate the score equations for training samples and new samples is as follows

$$\left[\begin{array}{c} L_0 \\ \hline L_1 \end{array}\right] = \left[\begin{array}{c} K_{00} \\ \hline K_{10} \end{array}\right] W + \left[\begin{array}{c} E_0 \\ \hline E_1 \end{array}\right], \tag{2.4}$$

where W is the new weight matrix for learning the labels of the previous and new samples, K_{10} is the kernel vector evaluated at new samples and $[E_0^T \;\; E_1^T]^T$ is the fitting error matrix. We wish to design a sparse weight matrix W that keeps the error matrix $[E_0^T \;\; E_1^T]^T$ small. This problem also consists of p ℓ_1- norm minimizations for each column of weight matrix

$$\begin{aligned} &\min \ \|\underline{w}_i\|_1 \\ &\text{s.t.} \\ &\|\underline{l}_i - \left[\begin{array}{c|c} K_{00}^T & K_{10}^T \end{array}\right]^T \underline{w}_i\|_2 \leq \epsilon, \text{ for } \ i = 1, 2, \ldots, p, \end{aligned} \tag{2.5}$$

where $\underline{w}_i$ is the i^{th} column of W and $\underline{l}_i$ is the i^{th} column of $\left[\begin{array}{c|c} L_0^T & L_1^T \end{array}\right]^T$.
The problems in (2.5) can be solved for sparse columns of weight matrix $\underline{w}_i^*, i = 1, 2, \ldots, p$ using a sparse approximation algorithm such as OMP [19],[20].

Formulating the problem as in (2.4) means that for every group of trained samples augmented by new samples, the sparse algorithm solves recursively for new weight matrix. Although this solution satisfies the scores for the new and previous samples, it lacks the ability to maintain the support of the previously trained weight matrix while in practice we are interested in keeping this support at each iteration of in-situ learning. In other words, we would rather to keep the previously selected representative samples in the memory of the learning system.

One way to maintain the support of the weights from previous learning step is to somehow enforce this support to the sparse approximation algorithm. This might requires us to modify part of the sparse approximation algorithm. For instance, we can modify the OMP algorithm such that it uses the support of weight matrix from the past learning steps as its initial atoms indices.

Alternatively we can update the weight matrix to learn the scores of the new samples by posing the problem as

$$\left[\frac{L_0}{L_1}\right] = \left[\frac{K_{00}}{K_{10}}\right](W_0^* + \Delta W_0) + \left[\frac{E_0}{E_1}\right], \tag{2.6}$$

where ΔW_0 is the update weight matrix for learning the labels of the new samples and $K_{10}(W_0^* + \Delta W_0)$ are the scores for the new samples. We wish to design ΔW_0 such that the updated weight matrix $W_0^* + \Delta W_0$ is sparse and the scores for previously learned samples do not change, that is

$$K_{00}(W_0^* + \Delta W_0) = K_{00}W_0^*. \tag{2.7}$$

The constraint in (2.7) gives

$$K_{00}\Delta W_0 = \mathbf{0},$$

which means that the columns of ΔW_0 must be in the null space of K_{00}.

Since, L_1, W_0^*, K_{00} and K_{10} are known, we can reformulate (2.6) as

$$\left[\frac{L_0 - K_{00}W_0^*}{L_1 - K_{10}W_0^*}\right] = \left[\frac{K_{00}}{K_{10}}\right]\Delta W_0 + \left[\frac{E_0}{E_1}\right], \tag{2.8}$$

Ideally, $L_0 - K_{00}W_0^* = 0$ because W_0^* is a solution to (2.1). Therefore, if the error E_0 is negligible, the top portion of (2.8) assures that the columns of ΔW_0 lie in the null space of K_{00}. This in turn assures that the scores of the previously learned samples do not fluctuate much. The bottom portion is the score equations for the new samples. Now the objective is to find columns of ΔW_0 that keep the fitting error $[E_0^T \;\; E_1^T]^T$ small enough and are sparse at the same time. So the problem can be

posed as the following ℓ_1- norm minimizations that are also solved using a sparse approximation algorithm

$$\begin{aligned} &\min \ \|\Delta \underline{w}_{0i}\|_1 \\ &\text{s.t.} \\ &\|\underline{l}_{2i} - \left[\begin{array}{c|c} K_{00}^T & K_{10}^T \end{array}\right]^T \Delta \underline{w}_{0i}\|_2 \leq \epsilon, \text{ for } i = 1, 2, \ldots, p, \end{aligned} \tag{2.9}$$

where $\Delta \underline{w}_{0i}$ is the i^{th} column of ΔW_0, $\underline{l}_{2i}$ is the i^{th} column of the left side of (2.8) and $\epsilon > 0$ is an error threshold that is determined by the user.

Let $W^* = [\underline{w}_1^* \ldots \underline{w}_p^*]$ and $\Delta W_0^* = [\Delta \underline{w}_{01}^* \ldots \Delta \underline{w}_{0p}^*]$ be solutions to (2.5) and (2.9) respectively. If the updated weight matrices W^* and $W_0^* + \Delta W_0^*$ are sufficiently sparse, then the new samples are sparsely represented by the dictionary $\mathcal{K}$ and there is no need to update the dictionary by adding the new samples. However, if W^* or $W_0^* + \Delta W_0^*$ is no longer sparse or the loss of sparsity relative to W_0^* is considerable, then simply updating the weight matrix is not sufficient and we have to update both the dictionary and the weight matrix, as explained in the next section.

2.4.2. CASE 2: IN-SITU LEARNING WITH SAMPLE ADDITION. Suppose that W^* in (2.4) or $W_0^* + \Delta W_0^*$ in (2.6) for Case 1 are not sufficiently sparse. This means that the scores for the new samples can not be represented sparsely by the current kernel functions without affecting the scores of the previously learned samples. In this case, we need to expand the dictionary by adding new kernel functions.

Let $\Delta\mathcal{K}$ denote a set of kernel functions with different parameters each of which is centered around one of the new data samples. If we add these kernels to the dictionary $\mathcal{K}$, then we can write the score equations for both previous training samples and new samples in the simplest matrix form as

$$\left[\begin{array}{c} L_0 \\ \hline L_1 \end{array}\right] = \left[\begin{array}{c|c} K_{00} & K_{01} \\ \hline K_{10} & K_{11} \end{array}\right] W + \left[\begin{array}{c} E_0 \\ \hline E_1 \end{array}\right]. \tag{2.10}$$

In this equation, K_{01} and K_{11} are matrices of kernel functions in $\Delta\mathcal{K}$. Each column of K_{01} and K_{11} correspond to a kernel function that is centered at a new sample point for some choice of kernel parameters. As we move from column to column the centers and/or the kernel types and their parameters change. The rows of K_{01} and K_{11} are obtained by plugging in the training samples and the new samples in these kernels, respectively.

Our goal is now to find the columns of W that are sparse and minimize the fitting error $[E_0^T \;\; E_1^T]^T$. This problem consists of the following p set of ℓ_1- norm minimizations that are solved using available sparse approximation algorithms for each column $\underline{w}_i$ of weight matrix separately

$$\begin{aligned} &\min \; \|\underline{w}_i\|_1 \\ &\text{s.t.} \\ &\|\underline{l}_i - K\underline{w}_i\|_2 \leq \epsilon, \text{ for } \; i = 1, 2, \ldots, p, \end{aligned} \tag{2.11}$$

where,

$$K = \left[\begin{array}{c|c} K_{00} & K_{01} \\ \hline K_{10} & K_{11} \end{array}\right]$$

is the expanded kernel matrix.

Similar to (2.4) in Case 1, the drawback of writing the problem as in (2.10) is that the representative samples from previous learning steps are not maintained in the memory of the learning system. Again, one way to solve this issue is to enforce the support of the columns of the weight matrix from previous learning iterations to the sparse approximation algorithm that solves (2.11).

Another way is to change the formulation of the problem so that we can update the weights instead of just solving for them recursively and independetly of the previous steps, that is

$$\left[\begin{array}{c} L_0 \\ \hline L_1 \end{array}\right] = \left[\begin{array}{c|c} K_{00} & K_{01} \\ \hline K_{10} & K_{11} \end{array}\right] \left(\left[\begin{array}{c} W_0^* \\ \hline 0 \end{array}\right] + \left[\begin{array}{c} \Delta W_0 \\ \hline \Delta W_1 \end{array}\right] \right) + \left[\begin{array}{c} E_0 \\ \hline E_1 \end{array}\right]. \tag{2.12}$$

Here, $\Delta W = \left[\begin{array}{c|c} \Delta W_0^T & \Delta W_1^T \end{array}\right]^T$ is the weight update matrix for learning the scores of the new samples that are now represented by the newly expanded kernel dictionary. To guarantee that the scores of the previously learned samples are not changed by updating the weights and the dictionary, we need to have

$$\left[\begin{array}{c|c} K_{00} & K_{01} \end{array}\right] \Delta W = 0. \tag{2.13}$$

This means that the update in the weight matrix must lie in the null space of $\left[\begin{array}{c|c} K_{00} & K_{01} \end{array}\right]$, which we denote by $\mathcal{N}(\left[\begin{array}{c|c} K_{00} & K_{01} \end{array}\right])$. We can rewrite (2.12) as

$$\left[\begin{array}{c} L_0 - K_{00}W_0^* \\ \hline L_1 - K_{10}W_0^* \end{array}\right] = \left[\begin{array}{c|c} K_{00} & K_{01} \\ \hline K_{10} & K_{11} \end{array}\right] \Delta W + \left[\begin{array}{c} E_0 \\ \hline E_1 \end{array}\right]. \tag{2.14}$$

where the left side and the matrix K are known. The top portion of (2.14) roughly satisfies the null-space condition in (2.13) because W_0^* is the solution to (2.1). Hence, by writing the score equations as in (2.14) we can learn the scores of the new samples represented by expanded dictionary while keeping the scores of previously learned samples almost unchanged. We wish to design ΔW so that it has sparse columns and minimizes the fitting error $[E_0^T \;\; E_1^T]^T$. This problem can be posed as

$$\begin{aligned} &\min \; \|\Delta \underline{w}_i\|_1 \\ &\text{s.t.} \\ &\|\underline{l}_{2i} - K\Delta \underline{w}_i\|_2 \leq \epsilon, \text{ for } \; i = 1, 2, \ldots, p, \end{aligned} \tag{2.15}$$

where $\Delta \underline{w}_i$ is the i^{th} column of ΔW. Again, these are p sets of ℓ_1-norm minimizations that can easily be solved using a number of algorithms.

Let $\Delta W^* = [\Delta \underline{w}_1^* \ldots \Delta \underline{w}_p^*]$ be the solution to (2.15). The resulting updated weight matrix $\left[\begin{array}{c|c} W_0^{*T} & 0^T \end{array}\right]^T + \Delta W$ should be sparse given that the dictionary is now expanded by the kernel functions centered about the new sample. However, it might be possible that it is not sparse. Therefore, we have come up with an alternative solution to (2.12) in which the sparsity is enforced to ΔW by setting the entries of ΔW_0 equal to zero. Therefore, (2.12) becomes

$$\left[\frac{L_0 - K_{00}W_0^*}{L_1 - K_{10}W_0^*}\right] = \left[\frac{K_{01}}{K_{11}}\right]\Delta W_1 + \left[\frac{E_0}{E_1}\right]. \tag{2.16}$$

Ideally $L_0 - K_{00}W_0^* = 0$, because W_0^* is a solution to (2.1), and if we enforce the columns of ΔW_1 to be in the null-space of K_{01}, then no more error than E_0 is introduced to the top portion of (2.16). This means

$$\Delta \underline{w}_{1i} = P_{K_{01}}^{\perp}\underline{\eta}_i, \text{ for } i = 1, 2, \ldots, p, \tag{2.17}$$

where $\Delta \underline{w}_{1i}$ is the i^{th} column of ΔW_1, $P_{K_{01}}^{\perp}$ is the orthogonal projection onto the null space of K_{01} and $\underline{\eta}_i$ is a vector whose projection onto the null-space of K_{01} becomes $\Delta \underline{w}_{1i}$ and also satisfies the bottom portion of (2.16).

Combining (2.17) and the bottom portion of (2.16) gives

$$\Delta \underline{w}_{1i}^* = P_{K_{01}}^{\perp}(K_{11}P_{K_{01}}^{\perp})^{\sharp}\underline{l}_{1i} \text{ for } i = 1, 2, \ldots, p, \tag{2.18}$$

where $\underline{l}_{1i}$ is the i^{th} column of $L_1 - K_{10}W_0^*$ and $(K_{11}P_{K_{01}}^{\perp})^{\sharp}$ is the pseudo-inverse of $K_{11}P_{K_{01}}^{\perp}$.

If the solution $\Delta W_1^* = [\Delta \underline{w}_{11}^* \ldots \Delta \underline{w}_{1p}^*]$ exists, the resulting updated weight matrix

$$\left[\frac{W_0^*}{0}\right] + \left[\frac{0}{\Delta W_1^*}\right]$$

will definitely be sparse and satisfies the scores of the new samples.

2.5. CONCLUSION

We started this chapter by giving an introduction about multiple kernel classification systems and then we gave a detailed description of our proposed multi-kernel classifier. We first described how we designed the score function of this classifier using a predefined dictionary of kernel functions and we continued by developing a mathematical framework for sparse representation of this score function using a training dataset. Finally, we described how the user feedback can contribute to the classifier's learning process through in-situ learning.

CHAPTER 3

EXPERIMENTAL RESULTS

3.1. INTRODUCTION

In this chapter we asses the performance of the general multi-kernel in-situ target classification method proposed in chapter 2. We evaluate the viability of the developed method on a specific feature set of underwater images called EOD dataset provided by Information System Technologies Inc (ISTI). Our specific goal is to classify the mine-like and non-mine-like (clutter) objects. We have used different performance measures such as receiver operating characteristic (ROC) curve, confusion matrices and recall plots for the evaluation of our system. In the following sections we first describe the EOD dataset and the feature vectors extracted from this dataset. Then, we explain how we designed our baseline system by training it with part of the dataset to obtain the baseline weights. Finally, we show how we used another part of the dataset for in-situ (online) training of our system to obtain the weight updates. The remaining part of the dataset is used as generalization (test) set for evaluation of the resulting classifier.

3.2. DATASET DESCRIPTION AND PREPROCESSING

We have generated the results of this chapter using the EOD dataset. This dataset consists of feature vectors extracted from pairs of high-frequency (HF) and broadband (BB) sonar images taken from the oceanfloor at one geographical environment with medium to hard difficulty levels in clutter density, so the HF and BB images contain actual targets and background clutter. In areas with higher clutter density the targets are obstacled by background clutter and as a result are harder to be detected and classified. The images are collected using the Small Synthetic Apertur Minehunter (SSAM) [24] that is a dual frequency band Synthetic Aperture Sonar (SAS) system. The dual channels of SSAM are capable of constructing the HF sonar with high spatial resolution

that captures more target details and the BB sonar with lower spatial resolution that has better clutter suppression ability. The targets in the images include mine-like objects such as bullet, cone, cylindrical and spherical shaped objects. The non-targets can be some typical objects (man-made or natural) such as tire, fish traps and pipes found in the image backgorund.

The image pairs are partitioned into smaller regions of interest called snippets with 50% overlap in both dimensions in order to ensure that target is not divided between different snippets. As a result of overlapping, the snippets with targets will contain the entire target not just part of it. The size of the snippets is selected based on the average target size in the images. As scanned horizontally, the image snippets containing an object have a background-highlight-shadow-background pixel distribution. The highlight-shadow pixel distributions vary depending on the type of the object and its range from the sonar. Thus, if we extract a distinct feature from each row of pixels in the image snippet, the resulting feature vector will represent the unique patterns of an object within the image snippet. Therefore, we extract the feature vectors based on the entropy of each row of pixels in a snippet. We calculate the entropies of all rows of the downsampled HF and BB image snippets and chain them together to form a 30-dimensional feature vector.

The resulting collection of 30-dimensional feature vectors extracted from the image snippets along with their labels constitute the EOD dataset. We should note that the non-targets in EOD dataset are so similar to the targets and are considered as difficult ones to be classified. We randomly choose 25% of the total 661 feature vectors for baseline training. We use half of the remaining feature vectors for in-situ learning and the other half for measuring the generalization error of our classification system. The generalization error measures how well a classifier generalizes to previously unseen data. Table 3.1 shows the proportions of the feature vectors used for baseline training, in-situ learning and generalization subsets of EOD dataset. It also shows the number of targets/non-targets in each subset. In the following sections we apply our classification

TABLE 3.1. EOD dataset

	Targets	Nontargets	Total
Baseline	91	74	165 (25 % of all samples)
In-situ	136	111	247 (37.5 % of all samples)
Generalization	137	112	249 (37.5 % of all samples)

TABLE 3.2. Correct classification rate

	Training	In-situ	Generalization
Recall	1	0.9190	0.9398

TABLE 3.3. Sparsity level of weight matrix after baseline training

Weight matrix	Column 1	Column 2
Distinct centers (out of 165)	25	22
Total non-zeros (out of 2805)	26	22

TABLE 3.4. Confusion matrix after baseline training: zero threshold / knee point of ROC curve

		Retrieved	
		Target	Nontarget
True	Target	133 / 127	4 / 10
True	Nontarget	11 / 6	101 / 106

method to these feature vectors and evaluate it using performance measures such as ROC curves and confusion matrices.

3.3. BASELINE TRAINING

Using the samples in the baseline training subset we first build the labels matrix L_0 and kernel matrix K_{00} that consists of Gaussian kernel functions centered around these samples. Each Gaussian kernel function is parametrized by 17 different variances σ^2 selected from the interval $[0.05 \;\; 6]$. Therefore, the total number of columns in K_{00} is $165 \times 17 = 2805$. We then run OMP algorithm [19],[20] for $p = 2$ times to solve (2.3) for each column of W_0 separately. The OMP algorithm stops running when the normalized residual reaches a desired size or in other words when the error gets small enough. The desired size of the residual is adjusted so that both the desired training

scores as well as a reasonable sparsity level are achieved. In our case $\epsilon = 0.09$ satisfies both of these conditions. We use the resulting optimal weights W_0^* to evaluate the system on the baseline training, in-situ and generalization subsets of EOD dataset. The correct classification rates for these subsets are shown in Table 3.2. We can observe that the baseline training already does an almost perfect job on the generalization data set with a high value of correct classification rate. In Table 3.3 We can see the number of distinct centers that are selected by the algorithm and also the total number of non-zero entries for each column of W_0^*. This table shows that only 25 representative samples are needed to satisfy the scores of all 165 samples in the baseline training set hence indicating a high level of sparsity. We should point out that in addition to the OMP, we tried to solve the problem using a basis pursuit denoising approach [16], [17] such as log-barrier algorithm [18]. However, the result did not turn out to provide a reasonable level of sparsity. Figure 3.1 shows the ROC curve for the generalization set. Table 3.4 depicts the generalization set confusion matrix for two different threshold values of likelihood ratio test i.e. $\gamma = 0$ and the value of γ corresponding to the knee point in Figure 3.1. The results presented in Table 3.4 show a small misclassification rate while the false alarm rate is a bit higher. In other words, this method has a better ability in detecting the targets rather than non-targets. Finally Figure 3.2 shows the distribution of the Gaussian variances that have been selected by the OMP algorithm where all of the 17 available variances are selected at least once by OMP.

3.4. IN-SITU LEARNING

After the baseline training, we perform the in-situ learning using the samples in the in-situ subset. At each in-situ iteration, we select a sample from the in-situ subset randomly. Using the weight matrix from the previous learning iteration, we calculate the score vector for this randomly selected sample. If the score vector suggests that this sample is classified correctly (i.e. the sample

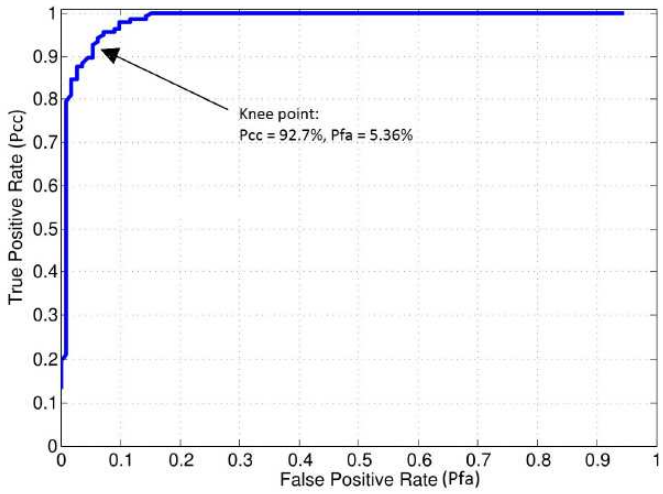

FIGURE 3.1. Generalization ROC curve after baseline training

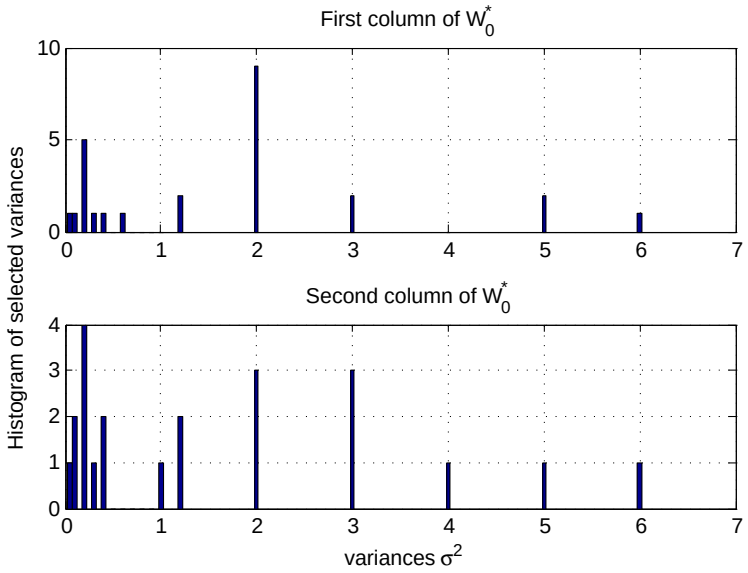

FIGURE 3.2. Histogram of the selected Gaussian variances after the baseline training

is located in the correct side of the current decision boundry) and the score values for 2 classes are well-separated (i.e. the sample is located far enough from the current decision boundry not inside it) , then we just evaluate the algorithm on the training, in-situ and generalization subsets, keep the results and move to the next iteration. If the scores imply that either the sample is incorrectly classified or is correctly classified but the score values for 2 classes are too close to each other, we learn the sample by changing the weights (i.e. changing the decision boundry). We first try the first

case to just update the weights without expanding the dictionary by solving one of the problems in section 2.3.1. We set a threshold for the number of non-zeros that can be added to the weight matrix after the update. If the number of non-zero entries that are added to the weight matrix after the update is smaller than the threshold, we keep the updated weights, evaluate the algorithm and continue to the next iteration. If this number exceeds the threshold, we expand the dictionary by adding new kernel functions centered around new sample to it and we solve one of the problems in section 2.3.2 to update the weights and again we evaluate the algorithm and move to the next iteration. If the resulting updated weight matrix is not sparse enough according to our threshold, even for case 2, we consider this sample as an outlier and remove it from the dictionary but we still evaluate the algorithm using the weights from the previous iteration and keep the result. We continue the learning until we run out of the in-situ samples.

Therefore, as the learning progresses, no matter a sample is learned or not, we evaluate the algorithm on the entire training, in-situ and generalization subsets and calculate the correct classification rates for them. At the end of the in-situ learning process, we plot the recall and the generalization ROC curves to see how well the in-situ learning performs and if it improves the correct classification rate or not.

First we try the simplest way of formulating the problem as stated in (2.4) and (2.10) and solve either (2.5) or (2.11) depending on the sparsity level that we can achieve. The results show that out of 247 in-situ samples 8 are learned without addition and 11 with addition to the kernel dictionary. The rest of the samples are not learned (i.e. do not contribute to the change of the weights) because they are considered correctly classified and their score values for 2 classes are well-separated. Table 3.5 represents the number of distinct centers and the total number of non-zero elements of the weight matrix after in-situ learning. The numbers still prove a high sparsity level for the weight matrix. Table 3.6 shows the generalization confusion matrices after the baseline and

TABLE 3.5. Sparsity of weight matrix before / after in-situ learning that solves (2.5) or (2.11) with OMP

	Column 1	Column 2
Distinct centers (out of 165/ 176)	25/ 39	22 / 39
Total non-zeros (out of 2805 / 2992)	26 / 42	22 / 42

TABLE 3.6. Confusion matrix before / after in-situ learning that solves (2.5) or (2.11) with OMP

		Retrieved	
		Target	Nontarget
True	Target	133 / 133	4 / 4
True	Nontarget	11 / 8	101 / 104

in-situ training when the threshold value for likelihood ratio test is set to $\gamma = 0$. The numbers show an improvement in the false alarm rate while it has no effect on the classification rate of the targets. The overall classification rate is only improved slightly after the in-situ learning. We can observe this also in Figure 3.3 where the correct classification rates for the whole training, in-situ and generalization subsets are plotted. The green curve shows a noticeable improvement in the classification rate of the in-situ subset while the red curve shows only a slight improvement in the classification rate of the generalization subset. One reason behind this might be that the baseline system perfectly learns the subspace spanned by the dataset and there is nothing left from this subspace to be learned through in-situ learning. In this case we can conclude that the wrongly-classified generalization samples are all outliers and we are not able to find a subspace that contains the baseline samples as well as these outliers no matter how good our in-situ learning system is. Another reason might be that the change in the subspace weights after each in-situ iteration is too small to represent the bigger subsace that contains the new in-situ sample. Figure 3.4 is the generalization ROC curve after baseline and in-situ training that shows no significant change after in-situ learning due to the mentioned possible reasons.

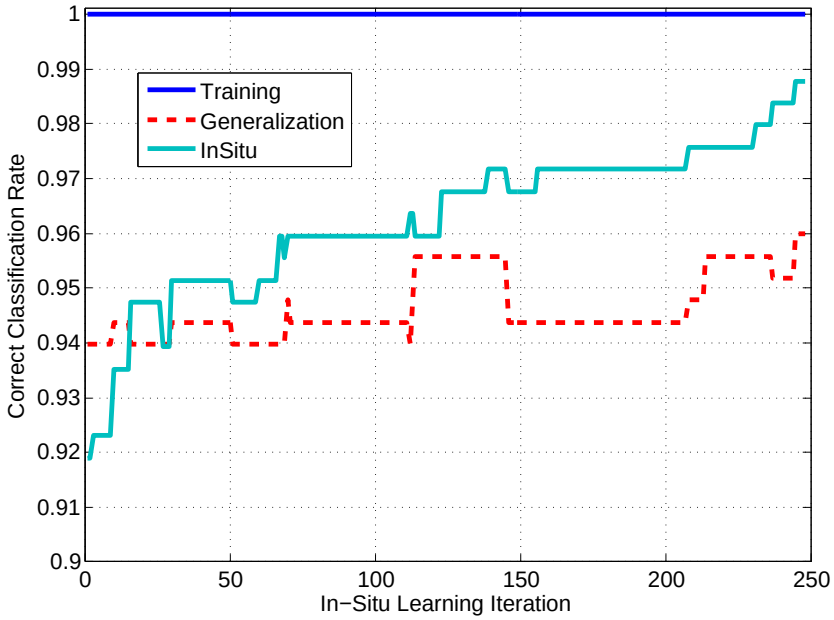

FIGURE 3.3. Correct classification ratio during in-situ learning that solves (2.5) or (2.11) with OMP

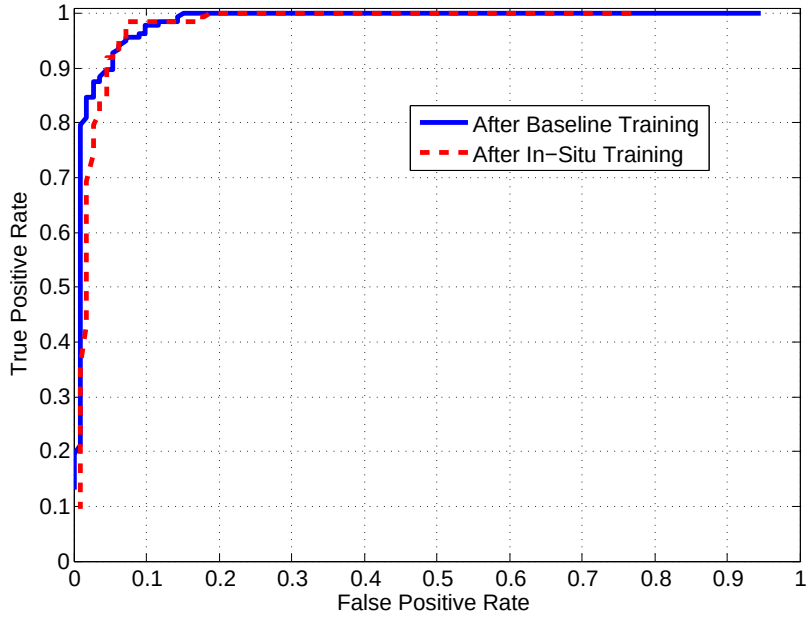

FIGURE 3.4. The generalization ROC curve before and after in-situ learning that solves (2.5) or (2.11) with OMP

As mentioned briefly in section 2.3.1, the drawback of this in-situ learning approach is that the previously selected centers are entirely cleared from the system's memory at each larning iteration. However, if we keep track of the weights as learning progresses, we can observe that there is only a

small difference in the supports of the weight matrices from iteration to iteration. That means only a few of the old centers are replaced by the new ones at each iteration. This can be observed in Figure 3.5 where the boxplots for the indices of the non-zero entries in both columns of the weight matrix are shown. The boxplots display the variation in the support of the columns of weight matrix during the in-situ learning iterations. The bottom and top of each box are the first and third quartiles of the indices, and the band inside each box is the median of the indices. The ends of the whiskers extended from each box represent the minimum and maximum index values. The plot does not show a significant change of the indices during iterations and also no outlier points are observed in the plot. Due to this fact we came up with the idea to modify the OMP algorithm so that at each iteration it selects all of the previously selected centers initially. In other words, we can enforce the suppot of weight matrix at iteration i to be a subset of its support at iteration $i+1$. A description of the modified OMP algorithm is give in Appendix A.

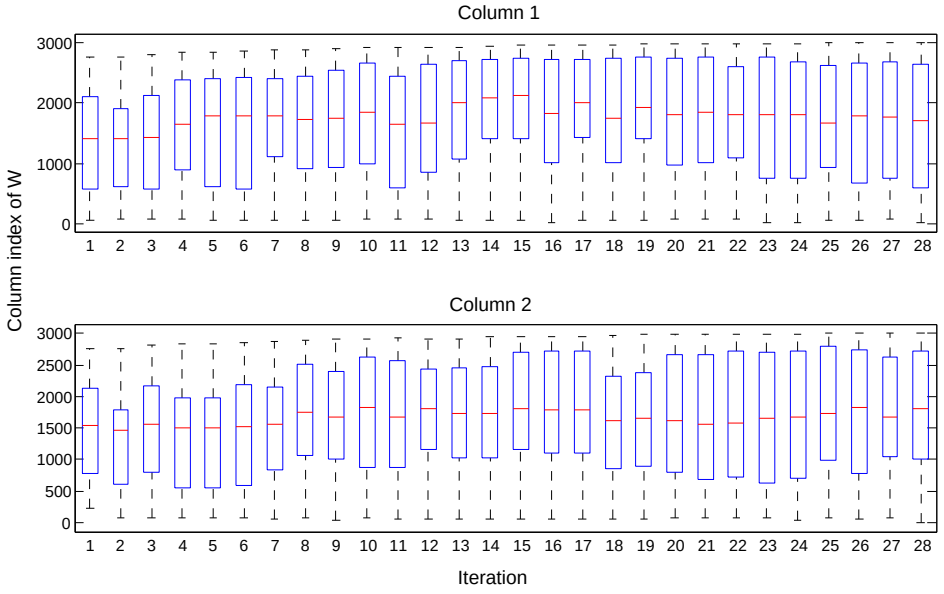

FIGURE 3.5. The sparsity pattern for the columns of weight matrix during learning iterations with OMP

So, this time we use the modified OMP algorithm to solve (2.5) or (2.11) depending on the level of sparsity that we obtain after the weight update without addition case. Here, the support of

TABLE 3.7. Sparsity of weight matrix before / after in-situ learning that solves (2.5) or (2.11) with modified OMP

	Column 1	Column 2
Distinct centers (out of 165/ 178)	25/ 66	22/ 61
Total non-zeros (out of 2805/ 3026)	26/ 76	22/ 68

TABLE 3.8. Confusion matrix before / after in-situ learning that solves (2.5) or (2.11) with modified OMP

		Retrieved	
		Target	Nontarget
True	Target	133 / 135	4 / 2
True	Nontarget	11 / 10	101 / 102

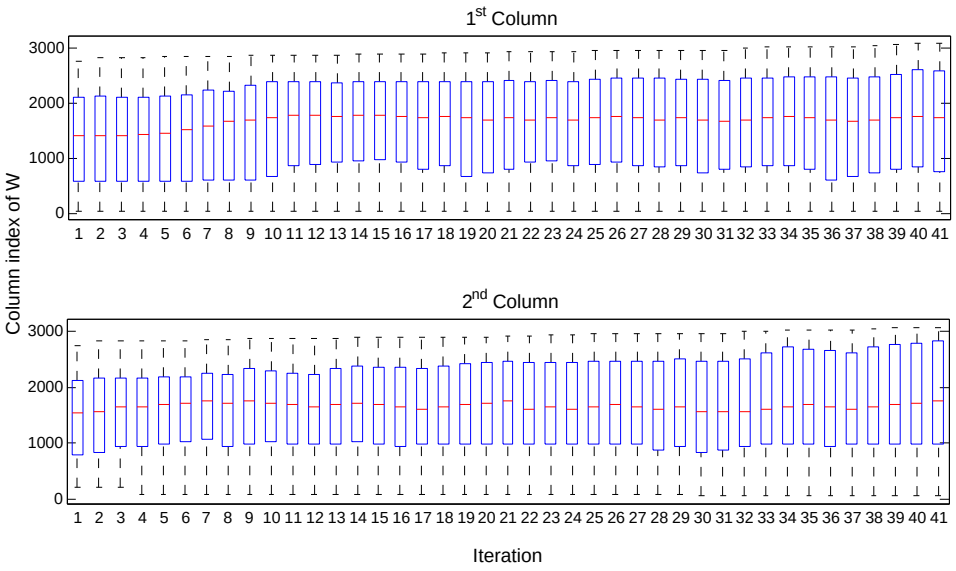

FIGURE 3.6. The sparsity pattern for the columns of weight matrix during learning iterations with modified OMP

weight matrix during each iteration is retained and used in the next iteration. Out of 247 in-situ training samples 18 are learned without addition and 13 with addition to the kernel dictionary. The resulting weight matrix after this in-situ learning experiment is still highly-sparse according to Table 3.7. In Figure 3.6 we can observe the sparsity pattern of this weight matrix during in-situ iterations where the boxplots show more similarity between the indices from one iteration to another.

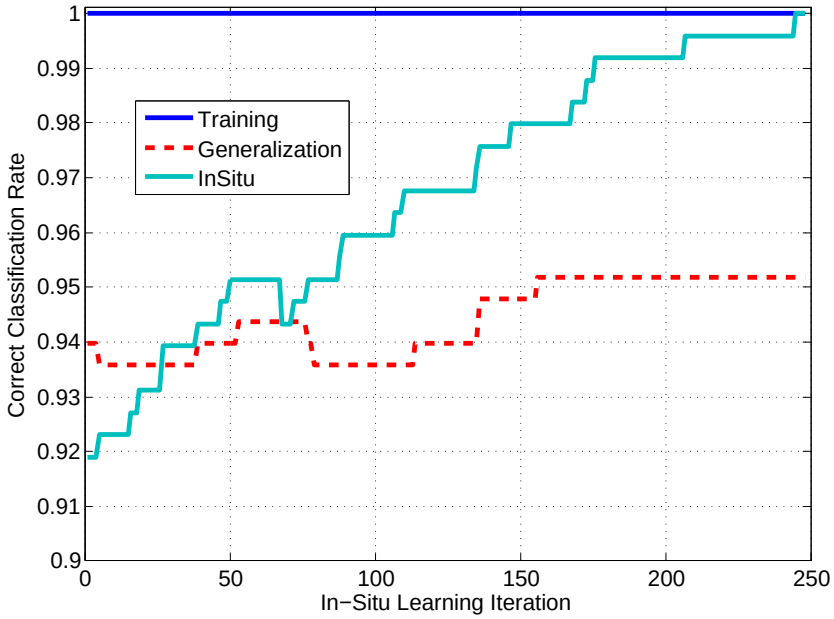

FIGURE 3.7. Correct classification ratio during in-situ learning that solves (2.5) or (2.11) with modified OMP

According to Table 3.8, there is a slight improvement in the classification rates of targets and non-targets after this in-situ experiment. The generalization ROC curve after this learning experiment is observed in Figure 3.8. The overall classification rate on the generalization set is again slightly improved as shown in Figure 3.7. Apparently, the only advantage of this approach to the previous one is that the representative samples are retained at each iteration as is required in the real sonar target classification scenarios.

Finally, we try to formulate the problem as in (2.6) and (2.12). We solve (2.9) or (2.15) depending on the sparsity to find the weight updates. The results of this approach are not satisfying. For each and every in-situ training sample, the updated weights for case 1 (without addition) and even case 2 (with addition) introduce many new non-zero entries and therefore the sparsity is lost significantly. As the final step we consider the second alternative for solving (2.12). We use this approach and solve for the weight updates with the sparsity enforcement according to (2.18). The weight updates in (2.18) are too small to make any change in the scores of generalization samples

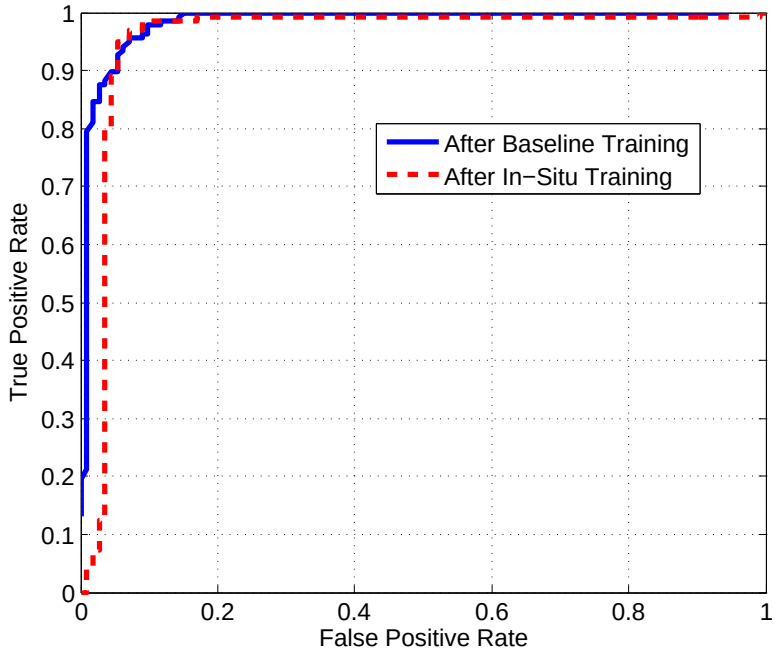

FIGURE 3.8. The generalization ROC curve before and after in-situ learning that solves (2.5) or (2.11) with modified OMP

due to very small entries in the null-space of the matrix K_{01}. On the other hand, if the variances are selected such that the entries are bigger, the null-space shrinks and the previously trained scores will change significantly. This issue can be related to the property of Gaussian kernel matrices. For the Gaussian kernels, as long as the points are distinct the kernel matrix will be full-rank and there will be no space left to put anything sparse in it. The solution might be to use a different kernel function that may even not satisfy the Mercer's theorem [25]. In addition to the RBF Gaussian kernel functions we have tried other kernel functions as well. Using the Laplacian kernel functions [10], we observed very similar results in terms of sparsity level as well as classification rate to the case where Gaussian kernel is used. However, the results obtained with the Quartic and Polynomial kernel functions [10] do not lead to a high level of sparsity as those with Gaussian and Laplacian kernels.

3.5. CONCLUSION

In this chapter we showed the results of evaluaing the method proposed in chapter 2 on the EOD dataset given to us by Information System Technologies Inc (ISTI). This dataset consists of $30-$dimensional feature vectors extracted from underwater image snippets along with their labels (target and non-target). We first described the EOD dataset and then presented the evaluation results of our baseline system on this dataset using ROC curve and confusion matrix. We observed that the baseline system has a very good generalization performance. Finally, we evaluated our different in-situ learning approaches on EOD dataset and showed the results using recall plot and ROC curve. The results suggested that our in-situ learning approaches do not increase the generalization ability of our system despite the sparsity cost that they cause.

CHAPTER 4

CONCLUSION AND FUTURE WORK

In this work we have used sparse representation theory to develop a multi-kernel learning machine for in-situ learning and classification of underwater objects in sonar images. We applied the method to a SAS dataset of underwater objects to classify them into two groups of mine-like and non-mine-like objects. We first did the baseline training using a randomly selected subset of this dataset and found a sparse representation of the proposed multi-kernel score functions for this subset. We then performed the in-situ learning on another subset of the dataset to update the sparse score representations for the new unseen samples from this subset.

The evaluation results show that the baseline training system performs very well in terms of both generalization ability (with 93.98% correct classification rate) and sparsity level (99% sparsity). On the orher hand, we can observe from the in-situ learning results that the in-situ learning (specifically without addition case) fails to make a noticeable improvement in the generalization performance despite the sparsity level that it loses. This is possibly due to the incapability of the in-situ learning system to capture the new subspace that contains the new in-situ samples. I.e. the subspace weights are not changed sufficiently to capture the bigger subspace containing the new samples. Another possibility is that the baseline system performs so well in learning the subspace spanned by this dataset that it leaves no room for improvement during in-situ learning. In other words, the subspace spanned by the dataset is learned pefectly through the baseline training and as a result no significant change is made to the subspace weights during the in-situ iterations. In this case, the lack of improvement in the generalization ability after the in-situ learning iterations can be explained by the fact that the wrongly-classified generalization samples are just outlier samples. In the latter case, using a manifold learning approach as opposed to the current subspace learning

approach might improve the generalization performance through in-situ. Moreover, developing an unsupervised selective sampling procedure that allows the most informative samples to be selected for in-situ learning, might also contribute in improvement of the in-situ learning performance by reducing the number of in-situ samples and avoiding the overfitting problems.

BIBLIOGRAPHY

[1] G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. Jordan, "Learning the kernel matrix with semidefinite programming," *Journal of Machine Learning Research*, vol. 5, pp. 27-72, 2004.

[2] F. Bach, G. Lanckriet, and M. Jordan, "Multiple kernel learning, conic duality, and the SMO algorithm," *Proceedings of the 21st International Conference on Machine Learning*, pp. 41-48, 2004.

[3] A. S. C. Ong and R. Williamson, "Learning the kernel with hyperkernels," *Journal of Machine Learning Research*, vol. 6, pp. 1043-1071, 2005.

[4] C. Micchelli and M. Pontil, "Learning the kernel function via regularization," *Journal of Machine Learning Research*, vol. 6, pp. 1099-1125, 2005.

[5] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf, "Large scale multiple kernel learning," *Journal of Machine Learning Research*, vol. 7, pp. 1531-1565, 2006.

[6] C. M. A. Argyriou, M. Hauser and M. Pontil, "A DC-programming algorithm for kernel selection," *Proceedings of the 23rd International Conference on Machine Learning*, pp. 41-48, 2006.

[7] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet, "SimpleMKL," *Journal of Machine Learning Research*, vol. 9, pp. 2491-2521, 2008.

[8] S. J. Kim, A. Zymnis, A. Manani, K. Koh, and S. Boyd, "Learning the kernel via convex optimization," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2008*, pp. 1997-2000, 2008.

[9] M. Azimi-Sadjadi, J. Salazar, S. Srinivasan, "An adaptable image retrieval system with relevance feedback using kernel machine and selective sampling," *IEEE. Trans. Image Process.,* vol.18,no.7,1645-1659, July 2009.

[10] B. Schölkopf and A. J. Smola, *Learning with Kernels.* Upper Saddle River, NJ: The MIT Press, 2001, first edition.

[11] E. Chong and S. Źak, *An Introduction to Optimization.* Hoboken, NJ: John Wiley & Sons, 2008, third edition.

[12] D. F. Specht, "Probabilistic neural networks," *Neural Networks,* vol. 3, pp. 109118, 1990.

[13] A. Pinkus, "Sparse representations and approximation theory," *Journal of Approximation Theory,* vol. 163, no. 3, 388-412, 2011.

[14] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statist. Soc. Ser. B,* vol. 58, no. 1, pp. 267-288, 1996.

[15] Z. Ben-Haim, Y. C. Eldar, and M. Elad, "Coherence-based performance guarantees for estimating a sparse vector under random noise," *IEEE Trans. Signal Processing,* vol. 58, no. 10, pp. 5030-5043, 2010.

[16] S. S. Chen, D. L. Donoho, M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Scientific Comput.,* vol. 20, no, 1, pp. 33-61, 1998.

[17] D. L. Donoho, M. Elad, and V. N. Temlyakov, "Stable recovery of sparse overcomplete representations in the presence of noise," *IEEE Trans. Inform. Theory,* vol. 52, no. 1, pp. 6-18, 2006.

[18] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of Computational and Applied Mathematics,* vol. 124, no. 1-2, pp. 281-302, Dec. 2000.

[19] T. T. Cai and L. Wang, "Orthogonal matching pursuit for sparse signal recovery with noise," *IEEE Trans. Signal Processing,* vol. 57, no. 7, pp. 4680-4688, 2011.

[20] J. A. Tropp, "Just relax: Convex programming methods for identifying sparse signals in noise," *IEEE Trans. Inform. Theory,* vol. 52, no. 3, pp. 1030-1051, 2006.

[21] C. Lemarechal, and C. Sagastizabal, ”Practical aspects of the Moreau–Yosida regularization: Theoretical preliminaries,” *SIAM Journal on Optimization,* vol. 7, no. 2, pp. 367-385, 1997.

[22] C. M. Bishop, *Neural Networks for Pattern Recognition.* Clarendon Press, 1995.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms.* MIT Press, 2009, third edition.

[24] D. Brown, D. Cook, and J. Fernandez, ”Results from a small synthetic aperture sonar,” in *OCEANS 2006,* pp. 1-6, September 2006.

[25] J. Mercer, ”Functions of positive and negative type, and their connection with the theory of integral equations,” in *Proc. Roy. Soc. London Ser. A,* pp. 441-458, January 1909.

APPENDIX A

ORTHOGONAL MATCHING PURSUIT

Orthogonal matching pursuit (OMP) [19] is an iterative greedy algorithm [23] that is used to approximate a high-dimensional sparse vector from only a small number of linear measurements that are corrupted by noise. OMP is simpler and faster compared to alternative sparse approximation algorithms.

Consider the following linear model

$$\underline{l}_0 = K_{00}\underline{w}_0 + \underline{e}_0 \tag{A.1}$$

where $\underline{l}_0 \in \mathbb{R}^M$ is the observation vector, $K_{00} \in \mathbb{R}^{M\times N}$, $\underline{e}_0 \in \mathbb{R}^M$ is the vector containing the measurement errors, and the number of measurements M is much smaller than the dimension N of the unknown vector $\underline{w}_0$ $(M \ll N)$. The goal of OMP is to recover the support of sparse vector $\underline{w}_0$ under the model (A.1). To do this, OMP first initializes its residual (error at each iteration) and atom (selected column of K_{00} at each iteration). Then, at each iteration OMP selects a new atom which is the column of K_{00} that is most correlated with the current residual and adds it to the previous set of selected atoms. The algorithm then updates the residual by projecting the observations orthogonally onto the null-space of the previously selected atoms. As a result, the residual at each iteration will be orthogonal to the selected atoms (columns), so the algorithm does not select an atom (a column) twice and the set of selected atoms grows at each iteration. The algorithm then continues to iterate until a stopping criterion is met and it stops. In the following we first show the OMP steps and then how we modify OMP to retain the support of the weight matrix during in-situ learning iterations.

The OMP algorithm iterates as follows

- Step 1: Initialize the residual $\underline{r}^{(0)} = \underline{l}_0$ and selected atom $K^{(0)} = \emptyset$ and start the OMP iteration counter $i = 1$.
- Step 2: At iteration i find the selected atom

$$t_i = \arg\max_t \quad \underline{k}_t^T \underline{r}^{(i-1)}$$

  where $\underline{k}_t$ is the t^{th} column of K_{00}. Update the set of selected atoms $K^{(i)} = [K^{(i-1)} \ \underline{k}_{t_i}]$.
- Step 3: Update the residual

$$\underline{\hat{w}}_0 = ({K^{(i)}}^T K^{(i)})^{-1} {K^{(i)}}^T \underline{l}_0$$

$$\underline{r}^{(i)} = \underline{l}_0 - K^{(i)} \underline{\hat{w}}_0 = P^{\perp}_{K^{(i)}} \underline{l}_0$$

  where $P^{\perp}_{K^{(i)}}$ denotes the orthogonal projection onto null-space of $K^{(i)}$.
- Step 4: Stop the algorithm if the stopping criterion is met otherwise $i = i + 1$ and return to step 2.

Several stopping rules for the OMP have been suggested in [19] such as ℓ_2-bounded noise, ℓ_∞-bounded noise, and Gaussian noise. Our stopping rule for OMP follows the ℓ_2-bounded noise case and is achieved when the residual size (norm squared of the residual) reaches a desired value. In other words, in our work the OMP stops running as soon as the norm squared of the residual falls behind a prespecified small threshold value.

As required by one of our experiments, we need to change the OMP algorithm such that at each in-situ learning iteration it retains the support of the weight vectors from the previous in-situ iterations. In other words, we want the OMP to select all of the selected kernel functions from previous in-situ iterations first and then continues iterating until the stopping condition is achieved. Therefore, the only change that is needed to be applied to OMP occures in the initialization step of

the algorithm. The initial atoms in this case are selected to be the columns that are retained from previous in-situ learning step. The initial residual is then accordingly calculated by projecting the observations onto the null-space of the initial atoms. So, the first step of the OMP in this case becomes

- Step 1: Initialize the selected atoms $K^{(0)} = K_s$, where the columns of K_s are the columns of K_{00} that have been selected by the OMP at previous in-situ iteration.

  Initialize the residual $r^{(0)} = P_{K_s}^{\perp} \underline{l}_0$.

  Start the OMP iteration counter $i = 1$.

The rest of the algorithm stays unchanged.