THESIS

A MULTI-TASK LEARNING METHOD USING GRADIENT DESCENT WITH APPLICATIONS

Submitted by

Nathan Dean Larson

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 2021

ABSTRACT

A MULTI-TASK LEARNING METHOD USING GRADIENT DESCENT WITH APPLICATIONS

There is a critical need to develop classification methods that can robustly and accurately classify different objects in varying environments. Each environment in a classification problem can contain its own unique challenges which prevent traditional classifiers from performing well. To solve classification problems in different environments, multi-task learning (MTL) models have been applied that define each environment as a separate task. We discuss two existing MTL algorithms and explain how they are inefficient for situations involving high-dimensional data. A gradient descent-based MTL algorithm is proposed which allows for high-dimensional data while providing accurate classification results. Additionally, we introduce a kernelized MTL algorithm which may allow us to generate nonlinear classifiers.

We compared our proposed MTL method with an existing method, Efficient Lifelong Learning Algorithm (ELLA), by using them to train classifiers on the underwater unexploded ordnance (UXO) and extended modified National Institute of Standards and Technology (EMNIST) datasets. The UXO dataset contained acoustic color features of low-frequency sonar data. Both real data collected from physical experiments as well as synthetic data were used forming separate environments. The EMNIST digits dataset contains grayscale images of handwritten digits. We used this dataset to show how our proposed MTL algorithm performs when used with more tasks than are in the UXO dataset.

Our classification experiments showed that our gradient descent-based algorithm resulted in improved performance over those of the traditional methods. The UXO dataset had a small improvement while the EMNIST dataset had a much larger improvement when using our MTL algorithm compared to ELLA and the single task learning method.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Dr. Mahmood R. Azimi-Sadjadi, and my committee members, Dr. Ali Pezeshki and Dr. Iuliana Oprea, for their guidance through my graduate research. I would also like to thank my friends in the Digital Signal and Image Processing Lab for their continued support and for helping me with my research and coursework.

I would like to thank the National Park Service (NPS) for their partial funding under cooperative agreement number P14AC00728 and the Strategic Environmental Research and Development Program (SERDP) under contract number W912HQ-17-C-0002. Without their funding, this research may not have been possible.

Finally, I would like to thank my family for supporting and encouraging me through all my graduate work.

DEDICATION

To my parents, Ron and Linda, and my brother Brian.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

Many classification problems include multiple related tasks that cannot use the same classifier. If a task does not contain a sufficient amount of training data by itself, training tasks individually will not provide accurate classification results. For these problems, it is critical to develop a multi-task learning (MTL) method [1,2] to train all tasks together using all of their training data. This thesis uses two problems to test the effectiveness of different MTL methods including the one developed in this work.

The main problem we focused on in this thesis was the classification of underwater targets. The problem of classifying underwater objects such as mines or UXOs or other objects must be performed in many different environments which may have significant variations between each other such as water density, sediment material, and various types of interference. These variations can cause differences in the received sonar signals of the same objects when observed in different environments. Thus, a classifier trained for one environment may perform poorly when used in others. This problem creates the need to develop a classification model that works across environments. Training models for each environment individually does not take advantage of the fact that observations of similar objects in different environments still share many features.

The second problem we used to test our MTL algorithm was the EMNIST [3] digits dataset containing images of handwritten digits. Instead of being different environments, the tasks in this dataset were the 10 different digits. This dataset showed us how our MTL algorithm performed against others when used to classify different classes in a single environment.

We also introduce kernel MTL (KMTL) to solve nonlinear classification problems. We suggest a way to modify our gradient descent-based MTL algorithm so that it can be applied to this KMTL problem. The implementation and testing of this method will be accomplished in future work.

In this thesis, we describe existing algorithms for the training of multi-task learning models and introduce an alternative method to train based on the gradient descent algorithm. To show the effectiveness of this algorithm, we use it to train classifiers with the datasets described above. The goal of this thesis is to determine if MTL is a good method to use for the problem of underwater object classification as well as any similar problem.

1.2 Literature Review

Multi-task learning has been the topic of many recent works [1, 2, 4, 5]. Grouping and Overlap in Multi-Task Learning (GO-MTL) [1] is a popular method that represents task parameters as sparse combinations of atoms, or columns, in a single shared dictionary. The proposed algorithm finds a dictionary and determines the sparse coefficients which atoms, or columns, should be used by each task. The idea behind this method is to automatically group similar tasks together during training by making them share dictionary atoms. The optimization problem in GO-MTL is not convex, so there may be many local minima. Additionally, solving this problem can become very computationally expensive for high-dimensional datasets or with a large number of tasks. The Efficient Lifelong Learning Algorithm (ELLA) [2] is an MTL algorithm designed to be used for lifelong learning which requires the ability for the MTL model to be updated after initial training by adding training data to the existing tasks or adding new tasks. ELLA uses the second-order Taylor approximation of the optimization problem in GO-MTL to update the dictionary every time a task is updated or added. The updates to the dictionary affect all tasks and can improve their performances. However, there is no guarantee that other tasks will improve and instead may result in reduced performance. More detailed summaries of GO-MTL and ELLA are in Chapter 3. The authors of [4] introduce Collective Lifelong Learning Algorithm (CoLLA) as an extension of ELLA to allow multiple agents each performing MTL. Each agent has its own set of unique tasks with a shared dictionary and communicates with connected agents by making their dictionaries equal when it receives an update. This method has the same dimensionality problems as ELLA. In [5], the authors introduce a convex MTL model using the hinge loss used in many support vector machines [6]. Because it is convex, there is no risk of finding a local minimum. As with the previously explained MTL models, the task parameters are described using a dictionary matrix. However, this model clusters tasks together by assigning each task to a single column of the dictionary without allowing overlap between groups.

The authors of [7] define three different categories of transfer learning: inductive transfer learning, transductive transfer learning, and unsupervised transfer learning. Multi-task learning is described an example of inductive transfer learning which is the setting where the source task and target task of knowledge transfer are different but related tasks, and the target task contains labeled data. Unlike other transfer learning methods, MTL does not have designated source and target tasks but instead allows all tasks to transfer knowledge to all other tasks. In [8], the authors contrast transfer learning and MTL by explaining that transfer learning provides larger benefits to target tasks than to source tasks while MTL treats all tasks equally. These learning strategies both provide knowledge transfer between tasks. However, they differ in terms of the direction of this transfer.

1.3 Contribution of this Work

Current MTL algorithms provide effective transfer learning between tasks. However, they are very inefficient while dealing with high-dimensional data. This thesis introduces a new method of training MTL model based on the gradient descent algorithm which has iterations with linear time complexity with respect to the dimension of the data. Our algorithm is generally slower when using low-dimensional data but much faster with high-dimensional data. This thesis also introduces a differentiable approximation of the ℓ_1-norm for the purpose of enforcing sparsity while having a defined gradient.

We tested our gradient descent-based algorithm for classification of the UXO targets as well as digits in the EMNIST datasets and compared the results to those found by using ELLA and a single task learner (STL). The proposed method resulted in the better performance on both datasets. Although the gained improvement over the other two methods was small on the UXO dataset, it led to a larger improvement on the EMNIST dataset.

For nonlinear classification problems, we introduce a kernelized version of the gradient descent-based MTL algorithm. This kernel MTL algorithm is used for non-linear classifiers and can be used on data that is not linearly separable. We do not present any testing results of this kernel algorithm. Improvement and testing of the kernel MTL is included in future works.

1.4 Organization of the Thesis

Organization of the thesis is as follows. Chapter 2 explains how the fast ray model (FRM) is used to simulate the propagation of acoustic waves underwater and generate synthetic acoustic color features. The physical setup of the target and reverberation experiment 2013 (TREX13) dataset and acoustic color extraction were also covered in this chapter. The chapter finishes by describing the EMNIST dataset of handwritten characters. In Chapter 3, we review two existing MTL methods, namely GO-MTL [1] and ELLA [2], and discuss both their strengths and weaknesses. Chapter 4 introduces our gradient descent-based approach to the MTL problem. We then discuss the strengths and weaknesses of this method and compare them to those of GO-MTL and ELLA. Chapter 5 explains how the MTL objective function can be kernelized and presents an algorithm to solve this kernel MTL problem. Chapter 6 gives the setups of the experiments for classification of UXOs and handwritten digits and presents classification results with an analysis. The thesis ends with Chapter 7 by reviewing the covered materials and suggesting potential future work.

CHAPTER 2

DATASETS AND DESCRIPTION

2.1 Introduction

In this chapter, we introduce the two different datasets that are used to demonstrate the effectiveness of the proposed method and conduct comparisons with other methods. The first dataset consists of feature vectors extracted from underwater sonar data collected for the purpose of classifying UXO vs. non-UXO objects. More specifically, we deal with the target and reverberation experiment 2013 (TREX13) [9] dataset which contains acoustic color [9] data collected from an experiment performed by placing different objects in the Gulf of Mexico to generate our in-situ and testing datasets. This experiment used different UXO and non-UXO targets but was conducted in a single environment. Collecting enough real data from different environments requires more experiments which can be difficult and costly. To get data needed to train classifiers, rather than performing many expansive data collection experiments, we can instead use a model that can generate realistic synthetic sonar data. The fast ray model (FRM) described in [9, 10] is used to generate large amounts of acoustic color training data for objects of different materials and shapes while simulating the responses in different environments. For a classification problem with more than two classes, our second dataset is the EMNIST [3] digits dataset. The full EMNIST dataset contains a large number of uppercase and lowercase letters along with digits. However, we only used the subset consisting of handwritten digits in this work. This dataset contains more samples of handwritten digits than the original MNIST dataset.

The chapter begins with Section 2.2 by reviewing the FRM and showing how it can be used to generate the target-in-the-environment response (TIER) [9] of each target. Section 2.2.1 shows how the responses are used to generate the acoustic color images. Section 2.3

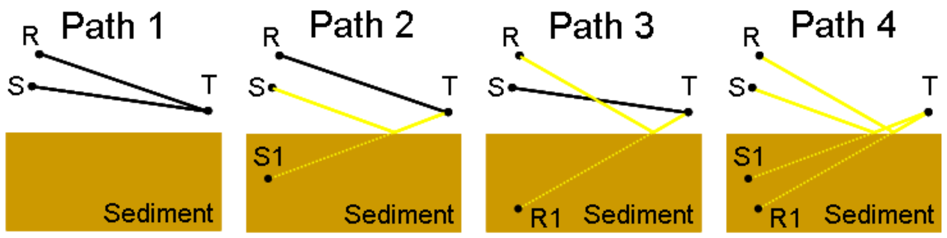


Figure 2.1: Four Ray Paths.

describes the physical setup of TREX13 test set and how the data was collected, and Section 2.3.1 describes the process of generating the acoustic color images of the targets using the collected data. Section 2.4 describes how the EMNIST dataset was generated and used in this work. Finally, Section 2.5 gives concluding remarks on this chapter.

2.2 Fast Ray Model - Generated Data - Training Set

The fast ray model (FRM) [9, 11] is used to simulate the propagation and scattering of underwater acoustic waves as they interact with underwater targets and the seafloor. This method uses a finite element (FE) model [12] of each target to generate lookup tables of scattering coefficients which can then be used to quickly perform these simulations in different environments. Each target has a FE model which is used to simulate the scattering of acoustic waves. The interaction of waves with the surface of the water can be ignored, and the direct path from source to receiver is short, so the signal travelling this path can be removed. The four propagation paths used are shown in Figure 2.1.

The source (S), receiver (R), and target (T) are located at $\mathbf{r}_S$, $\mathbf{r}_R$, and $\mathbf{r}_T$ with the image source (S1) and image receiver (R1) located at $\mathbf{r}_{S1}$, and $\mathbf{r}_{R1}$, respectively. The relevant distances are $d_1 = \|\mathbf{r}_S - \mathbf{r}_T\|$, $d_2 = \|\mathbf{r}_T - \mathbf{r}_R\|$, $d_3 = \|\mathbf{r}_{S1} - \mathbf{r}_T\|$, and $d_4 = \|\mathbf{r}_T - \mathbf{r}_{R1}\|$. The time delays for the four paths are $t_1 = (d_1 + d_2)/c_1$, $t_2 = (d_3 + d_2)/c_1$, $t_3 = (d_1 + d_4)/c_1$, and $t_4 = (d_3 + d_4)/c_1$ where c_1 is the speed of sound in water.

The far field scattered pressure can be written [9] as

$$p_s = p_0 A(\mathbf{k}_s, \mathbf{k}_i, \omega)\frac{\exp(ikr)}{r} \tag{2.1}$$

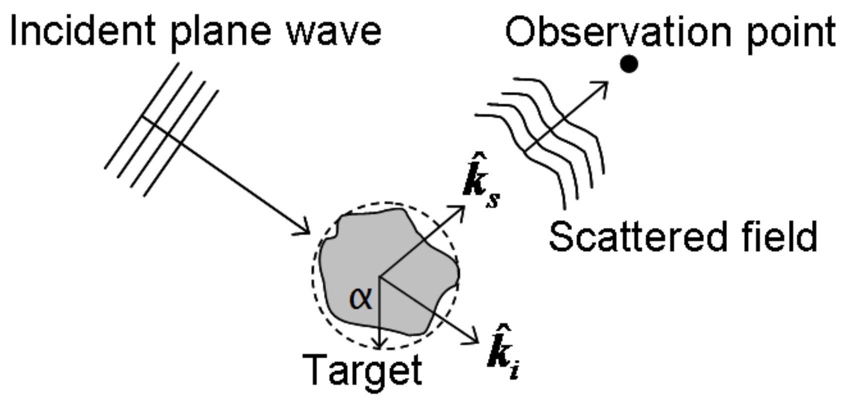


Figure 2.2: Sonar Wave Scattering

where p_0 is the incident pressure and r is the distance from a field point to the target. The scattering amplitude $A(\mathbf{k}_s, \mathbf{k}_i, \omega)$ is defined by the direction of the incident $\mathbf{k}_i$, the direction of scattered field $\mathbf{k}_s$, and the angular frequency ω, and $k = \omega/c_1$ is the angular wavenumber. The scattering amplitude contains information about the target's shape and material.

This model assumes there are no waves that interact with the target multiple times such as the path that hits the target, seafloor, and target again. It also assumes there is no scattering caused by reflection off the sediment. The authors in [13] ran a simulation without these two assumptions and compared the results to a physical experiment that included these paths. It was shown that the differences in the results caused by the extra scattering is negligible.

The spectrum of the scattered pressure is

$$P(\omega) = \left(\frac{A_1(\omega)}{d_1 d_2} e^{i\omega t_1} + \frac{V(\theta_g) A_2(\omega)}{d_2 d_3} e^{i\omega t_2} + \right. \\ \left. \frac{V(\theta_g) A_3(\omega)}{d_1 d_4} e^{i\omega t_3} + \frac{V^2(\theta_g) A_4(\omega)}{d_3 d_4} e^{i\omega t_4} \right) r_0 P_{src}(\omega) \tag{2.2}$$

where θ_g is the grazing angle, $P_{src}(\omega)$ is the spectrum of the transmitted pressure wave, $r_0 = 1m$ is the reference distance, and $A_k(\omega)$ is the scattering amplitude in path k that depends on the locations of the source, receiver, and target. The scattering amplitudes are generate from the results of simulations using FE models and are therefore not written as a mathematical expression. The reflection coefficient $V(\theta)$ is given by

$$V(\theta) = \frac{\rho \sin(\theta) - \sqrt{\kappa^2 - \cos^2(\theta)}}{\rho \sin(\theta) + \sqrt{\kappa^2 - \cos^2(\theta)}} \tag{2.3}$$

where $\rho = \rho_2/\rho_1$ and $\kappa = (1 + j\delta)/\nu$ with $\nu = c_2/c_1$. ρ_1 is the density of water, and ρ_2, c_2, and δ are the density, speed of sound, and loss parameter for the sediment, respectively. The pressure model in (2.2) contains the received spectrum from all four paths. Taking the inverse Fourier transform of (2.2) gives the received time domain signal.

2.2.1 Acoustic Color Features of Synthetic Data

The acoustic color features of a given target contain the returned spectral power at all interrogated azimuthal angles (aspects) around the target. The features at each aspect are used to classify the target using the methods to be explained in Chapters 3 and 4. Generation of the acoustic color image of any given target starts by calculating the four scattering amplitudes $A_i(\omega)$ in (2.2) by using a FE model to simulate the scattering of low-frequency acoustic waves (1-30 kHz) for the target. Next, (2.2) generates the spectrum of the sonar return signals along a circular path or a linear path around the target. Taking the inverse Fourier Transform of these spectra gives the return signals. The received signals are then pulse-compressed using the transmitted signal. Finally, the magnitudes of the Fourier Transform of the pulse-compressed received signals were windowed to 1-30 kHz to get the acoustic color features at each aspect.

Table 2.1 shows the type and location of the objects used in the FRM simulation. The transmitted signal was a linear frequency modulated (LFM) chirp across the desired frequencies. Model parameters such as sonar interface elevation and water conditions were set to match those in the TREX13 dataset, and a linear path was used to match the TREX13's linear rail. These targets are all symmetric, so only the acoustic color features for half of the aspects needed to be calculated. The objects simulated using FRM are identical to some of those objects used in the TREX13 dataset, and most of the ranges of these targets are also

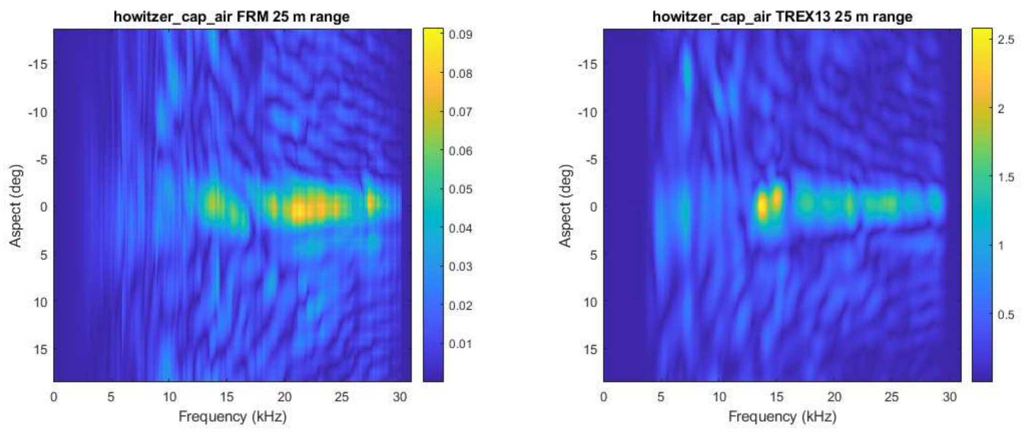


Figure 2.3: Comparison of acoustic color images from FRM (left) and TREX13 (right) for an air-filled howitzer cap at a range of 25m from the rail.

equal. A comparison of an acoustic color image generated using FRM to that of the experimental TREX13 is shown in Figure 2.3. As can be seen, the acoustic color image generated using the FRM accurately contains many of the spectral features seen in the acoustic color image of the same objects in the TREX13 dataset.

Table 2.1: Object Types and Ranges in FRM dataset.

Target	Class	Ranges(m)
3ft. Aluminum Cylinder	non-UXO	10,30,35,40
2ft. Aluminum Pipe	non-UXO	10,15,25,30
100mm. Aluminum Rocket Round	UXO	10,15,30
100mm. Solid Steel Rocket Round	UXO	10,15,25,30
105mm. Bullet (Air filled)	UXO	15,20,25
105mm. Bullet (H_2O filled)	UXO	15,20,35
155mm. Howitzer w/ Cap (H_2O filled)	UXO	15,25,30
155mm. Howitzer w/o Cap	UXO	25,30,40

2.3 TREX13 Dataset - Testing Set

TREX13 [9] was an experiment designed to detect and classify underwater targets using low-frequency sonar. The experiment placed a sonar tower on a straight 40m long rail with 30 targets each placed about 10m to 40m away from the rail on the seafloor. The sonar tower consisted of six hydrophones. However, only data collected from the third hydrophone was used to create the acoustic color features. As the tower traveled along the rail, a 6ms LFM chirp was transmitted every 0.025m, and the reflected signals were received by the hydrophones. The 30 targets were first placed with their tails pointed at $-80°$ to the rail, and the tower was run along the rail. The measurements were captured with the targets placed from $-80°$ to $80°$ in $20°$ increments orientation angles in 10 separate runs. Each run consisted of approximately 1600 pings sampled at 100kHz. These runs allowed for the construction of acoustic color images of all targets. A list of some of the targets along with their positions in the experiment is shown in Table 2.2. These correspond to the same objects in Table 2.1 for which we have the model-generated data.

Table 2.2: Object Types and Ranges in TREX13 dataset.

Target	Class	Ranges(m)
3ft. Aluminum Cylinder	non-UXO	30,35,40
2ft. Aluminum Pipe	non-UXO	15,25,30
100mm. Aluminum Rocket Round	UXO	10,15,30
100mm. Solid Steel Rocket Round	UXO	10,15,25,30
105mm. Bullet (Air filled)	UXO	15,20,25
105mm. Bullet (H_2O filled)	UXO	15,20,35
155mm. Howitzer w/ Cap (H_2O filled)	UXO	15,25,30
155mm. Howitzer w/o Cap	UXO	25,30,40

2.3.1 Acoustic Color Features of TREX13 Data

The data collected during the experiment is the received stave data, so the generation of acoustic color features of this data is similar to the process in Section 2.2.1 after calculating the received signal. That is, the generation of these features begins by pulse compressing the signal and isolating the object with a spatial filter [14]. The magnitudes of the Fourier Transform of the filtered signal were windowed to 1-30 kHz to get the acoustic color image. These features for each aspect contain 301 frequency bins with a frequency resolution of 100Hz. The aspect separation from of the acoustic color images is $0.5°$ with a total of 721 aspects per object. The dimensions of the acoustic color images of the TREX13 sonar data match those of the synthetic sonar data generated using the FRM model [9].

2.4 EMNIST Digits

The National Institute of Standards and Technology (NIST) maintains many datasets including ones containing handwritten digits and letters. In 1995, NIST created their Special Database 19 (SD-19) [15] which is a dataset of over 810,000 handwritten characters from 3600 writers. This dataset contains previously collected Special Databases 1 (SD-1), 3 (SD-3), and 7 (SD-7). SD-19 is organized in 5 separate data hierarchies: By_Page, By_Author, By_Field, By_Class, and By_Merge. The By_Page hierarchy contains the binary scans of all 3699 completed forms. The By_Author hierarchy contains the images of the individual characters separated by their authors. The By_Field hierarchy contains the individual characters separated by the field in which they appear on the forms. The By_Class hierarchy organizes the characters into 62 classes consisting of 10 digits, 26 lower-case letters, and 26 upper-case letters. Finally, the By_Merge hierarchy organizes the characters in the same way as in By_Class and merges the classes of letters that are similar when in lower-case and upper-case. These letters are C, I, J, K, L, M, O, P, S, U, V, W, X, Y, and Z.

The modified NIST (MNIST) [16] dataset is derived from SD-1 and SD-3. SD-1 contains 58,527 digits written by 500 writers. The digits from 250 of the writers were placed in the training set, and the digits from the other 250 writers were placed in the testing set. Images from SD-3 were added to both the training and testing sets to increase them each to 60,000 examples. The final training set contains all 60,000 examples. However, the final testing set contains only 10,000 samples with 5,000 from SD-1 and 5,000 from SD-3. The original images in the NIST dataset are 128×128 binary images. The images were first down-sampled to 20×20 and became 8-bit grayscale because of the anti-aliasing effect of their normalization algorithm. Finally, each image was centered in a 28×28 image using the image's center of mass.

EMNIST [3] is an extension of the MNIST dataset of handwritten digits and is derived from the By_Class and By_Merge hierarchies of SD-19. SD-19 contains 128×128 binary images which were converted to 28×28 8-bit grayscale images by applying the following

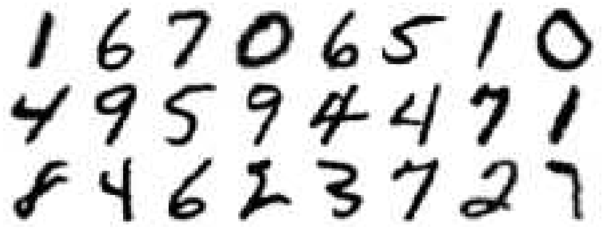


Figure 2.4: Examples of EMNIST digits.

process to each image. First, a Gaussian filter with standard deviation $\sigma = 1$ was used to soften the edges of the image. Next, the image was cropped to remove extra white space and contain only the region of interest. The extracted image was then centered in a square image and padded with a 2 pixel border. Finally, the image was resampled to 28×28 using bi-cubic interpolation and scaled to an 8-bit grayscale image.

EMNIST is organized into 6 different datasets. The first two datasets are By_Class and By_Merge with the processing described above. The EMNIST Balanced dataset is a subset of the By_Merge dataset which is balanced to contain an equal number of all 47 classes with 131,600 total images. The EMNIST Letters dataset is a subset of the By_Merge dataset which merges all upper-case and lower-case classes together into 26 balanced classes consisting of 103,600 letters. The EMNIST digits dataset is another subset which contains 10 balanced classes of 280,000 digits. Finally, the EMNIST MNIST dataset was created to match the size of the MNIST dataset with 70,000 total images. In this work, we used a subset of the EMNIST digits dataset containing 5000 samples for training and 4000 samples for testing.

2.5 Conclusion

The FRM explained in Section 2.2 is an effective tool to simulate the propagation of sonar signal and generate synthetic data used for training of the MTL algorithms. We showed the scattering amplitude generated from the target's FE model can be used to generate the spectrum of the scattered pressure of all four ray paths. We also covered the process to generate the acoustic color features of the synthetic data in Section 2.2.1 which can then be used to train classifiers along with a small portion of the TREX13 dataset for in-situ training. A summary of the physical setup and collection of data in TREX13 was also described together with the generation of the corresponding acoustic color features. The final UXO datasets used in this work included 4000 feature vectors from the FRM dataset and 400 feature vectors from the TREX13 dataset as the training dataset and 4000 feature vectors from the TREX13 dataset as the testing dataset.

The next dataset covered was the EMNIST dataset containing handwritten digits and letters. The EMNIST dataset was used since it allows us to test the developed methods for M-ary classification (i,e, for M-class problems). The final EMNIST datasets used in this work are the training dataset with 5000 images containing the digits only and the testing dataset with 4000 separate images also consisting only of digits. The following two chapters describe the algorithms used to perform classification on these datasets.

CHAPTER 3

A REVIEW OF MULTI-TASK LEARNING ALGORITHMS

3.1 Introduction

Multi-task learning (MTL) is a form of transfer learning that involves the training of multiple related classification or regression systems. In this context, a task is a supervised problem defined by a set of training data with corresponding labels and a function that maps the input data to their labels. The idea behind MTL methods is to learn similarities between tasks during training so that they can be used to improve the performance and generalization on all tasks. In this chapter, we explain two popular MTL methods that are used in this work. These methods are Grouping and Overlap in Multi-Task Learning (GO-MTL) [1] and the Efficient Lifelong Learning Algorithm (ELLA) [2]. Although these algorithms can be used for many different problems, here we are focusing on logistic regression for the purpose of classification. In either classification or regression problems, a set of parameters is found to fit some given training data. Normally, different tasks are trained independent of each other. However, in MTL all tasks are trained together by allowing them to have shared features. Doing this increases the number of samples used for each task by allowing them to use training data from similar tasks.

GO-MTL [1] works by assuming the task parameters lie in a low-dimensional subspace and are separated into overlapping groups. The GO-MTL algorithm learns the subspace by finding its bases and determines which bases are used for each task. ELLA [2], on the other hand, is an extension of GO-MTL that allows for tasks and training data to be added after the initial training. This algorithm was developed to be used in an online environment and can be much faster than GO-MTL.

Organization of this chapter is as follows. Section 3.2 is a review of the GO-MTL algorithm and describes the process of two-class classification. Section 3.3 reviews ELLA and explains the strengths of this algorithm when compared to GO-MTL. Finally, Section 3.4 gives the conclusion of this chapter.

3.2 GO-MTL Method

A common way to define the problem of MTL is to assume that the tasks' parameter vectors lie in a shared low-dimensional subspace. This restriction forces the different tasks' parameters to share features while still being allowed to have some more task specific features. GO-MTL [1] uses this assumption to define the problem.

Suppose we have T tasks with each task $t \in [1, T]$ having N_t existing training samples $\mathbf{x}_i \in \mathbb{R}^d$ with corresponding outputs, or labels, $y_i \in \mathbb{R}$. The parameter vectors $\boldsymbol{\theta}^{(t)} \in \mathbb{R}^d$ all lie in a p-dimensional subspace defined by the dictionary matrix $\mathbf{L} \in \mathbb{R}^{d \times p}$ whose columns, or atoms, are the latent (unobservable) task parameters. These latent task parameters form a basis of the subspace that contains the parameters for each task t. Task t's parameter vector can be written as $\boldsymbol{\theta}^{(t)} = \mathbf{L}\mathbf{s}^{(t)}$ where $\mathbf{s}^{(t)} \in \mathbb{R}^p$ is the vector of sparse coefficients that determines which latent tasks are part of task t, and $\mathbf{S} = [\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(T)}]$ is the matrix of the sparse coefficients of all tasks. The sparsity of this vector $\mathbf{s}^{(t)}$ is needed to make sure that only a small number of latent atoms are used in each of the main tasks. This way, similar tasks will share many of the same latent atoms in $\mathbf{L}$, while unrelated tasks will have little to no overlap. To enforce sparsity, GO-MTL adds the ℓ_1-norm of $\mathbf{s}^{(t)}$ as a penalty to the objective function.

The problem in GO-MTL is then to minimize the objective function,

$$e_T(\mathbf{L}, \mathbf{S}) = \frac{1}{T} \sum_{t=1}^{T} \left\{ \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\mathbf{x}_i^{(t)\top} \mathbf{L}\mathbf{s}^{(t)}, y_i^{(t)}) + \mu \|\mathbf{s}^{(t)}\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2, \tag{3.1}$$

where $\mathcal{L}(\hat{y}_i^{(t)}, y_i^{(t)})$ is the loss function evaluated at the true output $y_i^{(t)}$ and estimated output $\hat{y}_i^{(t)} = \mathbf{x}_i^{(t)\top} \mathbf{L}\mathbf{s}^{(t)}$, $\|\mathbf{s}^{(t)}\|_1$ is the ℓ_1-norm of $\mathbf{s}^{(t)}$, $\|\mathbf{L}\|_F$ is the Frobenius norm of $\mathbf{L}$, and μ and

λ are the regularization parameters that are preselected. This optimization problem is not jointly convex over $\mathbf{S}$ and $\mathbf{L}$, but it is convex over each one when the other is held fixed. GO-MTL applies an alternating optimization method by repeatedly minimizing over $\mathbf{S}$ while keeping $\mathbf{L}$ constant and minimizing over $\mathbf{L}$ while keeping $\mathbf{S}$ constant. Minimizing over $\mathbf{S}$ is done for each column, $\mathbf{s}^{(t)}$, by solving the optimization problem,

$$\mathbf{s}^{(t)*} = \operatorname*{argmin}_{\mathbf{s}^{(t)}} \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)}, y_i^{(t)}) + \mu \|\mathbf{s}^{(t)}\|_1, \tag{3.2}$$

with a fixed L. Minimization over $\mathbf{L}$ with a fixed $\mathbf{S}$ is performed by solving the optimization problem,

$$\mathbf{L}^* = \operatorname*{argmin}_{\mathbf{L}} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)}, y_i^{(t)}) + \lambda \|\mathbf{L}\|_F^2. \tag{3.3}$$

These two optimization problems are iteratively solved until convergence.

Classification (two-class) with GO-MTL uses a logistic regression with labels $y \in \{0, 1\}$. This is performed by using the logistic function,

$$\sigma(\hat{y}) = \frac{1}{1 + e^{-\hat{y}}}, \tag{3.4}$$

with the corresponding loss,

$$\mathcal{L}(\hat{y}, y) = -y \ln(\sigma(\hat{y})) - (1 - y) \ln(1 - \sigma(\hat{y})). \tag{3.5}$$

Minimization over $\mathbf{S}$ with a fixed $\mathbf{L}$ is done with a Lasso [17] method such as the two-metric projection method [18].

The two-metric projection method uses the loss function,

$$\mathbf{s}^{(t)*} = \operatorname*{argmin}_{\mathbf{s}^{(t)}} \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)}, y_i^{(t)}) \qquad s.t. \quad \|\mathbf{s}^{(t)}\|_1 \le \nu. \tag{3.6}$$

which is equivalent to (3.2). The gradient and Hessian of this loss function are

$$f(\mathbf{s}^{(t)}) = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)}, y_i^{(t)}), \tag{3.7}$$

$$\mathbf{g}(\mathbf{s}^{(t)}) = \frac{\partial f(\mathbf{s}^{(t)})}{\partial \mathbf{s}^{(t)}} = \frac{-1}{N_t} \sum_{i=1}^{N_t} (y_i^{(t)} - \sigma(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)})) \mathbf{L}^\top \mathbf{x}_i^{(t)}, \tag{3.8}$$

$$\mathbf{H}(\mathbf{s}^{(t)}) = \frac{\partial^2 f(\mathbf{s}^{(t)})}{\partial \mathbf{s}^{(t)} \partial \mathbf{s}^{(t)\top}} = \frac{1}{N_t} \sum_{i=1}^{N_t} \sigma(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)})(1 - \sigma(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)})) \mathbf{L}^\top \mathbf{x}_i^{(t)} \mathbf{x}_i^{(t)\top} \mathbf{L}. \tag{3.9}$$

The two-metric projection method is an iterative optimization algorithm used to solve constrained optimization problems of the form

$$\underset{\mathbf{s}^{(t)} \in \mathcal{X}}{\text{Minimize}} \quad f(\mathbf{s}^{(t)}) \tag{3.10}$$

$\mathcal{X} \subset \mathbb{R}^p$ is a closed convex subset of $\mathbb{R}^p$. In this problem, $\mathcal{X} = \left\{\mathbf{s}^{(t)} \in \mathbb{R}^p; \|\mathbf{s}^{(t)}\|_1 \leq \nu\right\}$. GO-MTL must solve the optimization problem,

$$\underset{\|\mathbf{s}^{(t)}\|_1 \leq \nu}{\text{Minimize}} \quad f(\mathbf{s}^{(t)}). \tag{3.11}$$

The update equation used in [18] at each iteration n is

$$\mathbf{s}_{n+1}^{(t)} = P\left(\mathbf{s}_n^{(t)} - \alpha_n \mathbf{H}_n^{-1} \mathbf{g}_n\right), \tag{3.12}$$

where P is the projection onto $\mathcal{X}$, α_n is the step size,

$$\mathbf{H}_n = \left.\frac{\partial^2 f(\mathbf{s}^{(t)})}{\partial \mathbf{s}^{(t)} \partial \mathbf{s}^{(t)\top}}\right|_{\mathbf{s}^{(t)} = \mathbf{s}_n^{(t)}},$$

and

$$\mathbf{g}_n = \left.\frac{\partial f(\mathbf{s}^{(t)})}{\partial \mathbf{s}^{(t)}}\right|_{\mathbf{s}^{(t)} = \mathbf{s}_n^{(t)}}.$$

The projection $P(\mathbf{x})$ gives the vector $\mathbf{s}^{(t)}$ in $\mathcal{X}$ nearest to $\mathbf{x}$ and is defined as

$$P(\mathbf{x}) = \underset{\mathbf{s}^{(t)} \in \mathcal{X}}{\text{argmin}} \|\mathbf{s}^{(t)} - \mathbf{x}\|_2^2, \tag{3.13}$$

which can be solved exactly using the algorithm explained in [19]. This step ensures that $\mathbf{s}^{(t)}$ always satisfies the constraint. If $\mathbf{x}$ is already a feasible solution, then $P(\mathbf{x}) = \mathbf{x}$.

Minimization over $\mathbf{L}$ with a fixed $\mathbf{S}$ can be done with gradient descent or the Newton-Raphson method [20]. Gradient descent updates use the first partial derivative of (3.3) with respect to $\mathbf{L}$

$$\frac{\partial e_T(\mathbf{L}, \mathbf{S})}{\partial \mathbf{L}} = -\frac{1}{T} \sum_{t=1}^{T} \frac{1}{N_t} \sum_{i=1}^{N_t} \left[(y_i^{(t)} - \sigma(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)})) \mathbf{x}_i^{(t)} \mathbf{s}^{(t)\top}\right] + \lambda 2 \mathbf{L}. \tag{3.14}$$

Then, the update equation for $\mathbf{L}$ becomes

$$\mathbf{L}_{n+1} = \mathbf{L}_n - \alpha \frac{\partial e_T}{\partial \mathbf{L}}\bigg|_{\mathbf{L}=\mathbf{L}_n}, \tag{3.15}$$

where α is the chosen step size. When using the Newton-Raphson method instead of gradient descent, GO-MTL uses the first two partial derivatives of the vectorized $\mathbf{L}$ matrix, $\text{vec}(\mathbf{L})$, to find the direction $\mathbf{M}_n \in \mathbb{R}^{d\times p}$ in iteration n. In each iteration, $\mathbf{M}_n$ is calculated as the solution to the system of equations generated by the Newton-Raphson method,

$$\begin{aligned}&\left[\frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\delta_i^{(t)}\,\text{vec}(\mathbf{x}_i^{(t)}\mathbf{s}^{(t)\top})\,\text{vec}(\mathbf{x}_i^{(t)}\mathbf{s}^{(t)\top})^\top + 2\lambda\mathbf{I}\right]\text{vec}(\mathbf{M}_n) \\ &= \text{vec}\left(\frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\left[(y_i^{(t)} - \sigma(\mathbf{x}_i^{(t)\top}\mathbf{L}\mathbf{s}^{(t)}))\mathbf{x}_i^{(t)}\mathbf{s}^{(t)\top}\right] - 2\lambda\mathbf{L}\right),\end{aligned} \tag{3.16}$$

where $\delta_i^{(t)} = \sigma(\mathbf{x}_i^{(t)\top}\mathbf{L}\mathbf{s}^{(t)})(1-\sigma(\mathbf{x}_i^{(t)\top}\mathbf{L}\mathbf{s}^{(t)}))$ and $\text{vec}(\cdot)$ vectorizes matrices via column stacking. Instead of using $\mathbf{M}_n$ as the update to $\mathbf{L}_n$, GO-MTL uses Armijo rule [21] to find a step size α that allows the process to converge. Armijo rule tests step sizes $\alpha = 1, \beta, \beta^2, \ldots$ until a step size is found that satisfies

$$e_T(\mathbf{L}_n + \alpha\mathbf{M}_n, \mathbf{S}) - e_T(\mathbf{L}_n, \mathbf{S}) \leq c\alpha\,\text{vec}(\mathbf{M}_n)^\top \frac{\partial e_T(\mathbf{L},\mathbf{S})}{\partial\,\text{vec}(\mathbf{L})}\bigg|_{\mathbf{L}=\mathbf{L}_n}, \tag{3.17}$$

where $\beta, c \in (0,1)$ are chosen to be constants. Then, $\mathbf{L}$ is updated using

$$\mathbf{L}_{n+1} = \mathbf{L}_n + \alpha_n\mathbf{M}_n. \tag{3.18}$$

The organization of the algorithm is shown in Algorithm 1. It starts by solving each task individually with a single task learner which minimizes the loss function for a single task. Singular Value Decomposition (SVD) [6] is performed on the single task parameters to initialize the dictionary $\mathbf{L}$. Each iteration of the loop at line 6 solves the minimization problems (3.2) and (3.3).

Each iteration of gradient descent while solving (3.15) has a time complexity of $O(Ndp)$ while each iteration of the Newton-Raphson method has a time complexity of $O(Nd^2p^2 + d^3p^3)$.

Algorithm 1 GO-MTL

```
1:  for $t = 1$ to $T$ do
2:      $\boldsymbol{\theta}^{(t)} \leftarrow \text{singleTaskLearner}(\mathbf{X}^{(t)}, \mathbf{y}^{(t)})$
3:  end for
4:  Perform SVD: $[\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(k)}] = \mathbf{U\Sigma V}^\top$
5:  $\mathbf{L} \leftarrow$ first $p$ columns of $\mathbf{U}$
6:  repeat
7:      for $t = 1$ to $T$ do
8:          repeat
9:              $\mathbf{s}^{(t)} \leftarrow (3.12)$
10:         until convergence of $\mathbf{s}^{(t)}$
11:     end for
12:     repeat
13:         $\mathbf{M} \leftarrow (3.16)$
14:         $\mathbf{L} \leftarrow (3.18)$
15:     until convergence of $\mathbf{L}$
16: until convergence of $\mathbf{S}$ and $\mathbf{L}$
```

While GO-MTL can provide good results when training many related tasks, it can also be very computationally expensive for high-dimensional datasets. Even with gradient descent, many iterations are needed to solve for $\mathbf{L}$, and this is repeated until the process converges. Additionally, no tasks or even new data can be added without retraining the entire system.

3.3 ELLA Method

ELLA is an MTL algorithm designed to extend GO-MTL to be used in lifelong learning by allowing tasks to be added and updated over time. The goal of lifelong learning is to create a system which can be updated after initial training by either adding new tasks or adding new training data to the existing tasks. Previous MTL algorithms [1,22] such as GO-MTL require all data and tasks to exist before training starts. If a task needs to be added to the model after training, the entire system must be retrained. This optimization method is performed over all previous and new data for all tasks, and every $\mathbf{s}^{(t)}$ is recalculated. The goal of ELLA is to allow lifelong learning by overcoming these two problems.

The formulation of ELLA starts with the same problem setup as GO-MTL with small modifications to the objective function. The objective function for ELLA is

$$\tilde{e}_T(\mathbf{L}, \mathbf{S}) = \frac{1}{T}\sum_{t=1}^{T}\left\{\frac{1}{N_t}\sum_{i=1}^{N_t}\mathcal{L}(g(\mathbf{x}_i^{(t)}; \mathbf{L}\mathbf{s}^{(t)}), y_i^{(t)}) + \mu\|\mathbf{s}^{(t)}\|_1\right\} + \lambda\|\mathbf{L}\|_F^2, \tag{3.19}$$

where $g(\cdot)$ is any activation function, e.g., $g(\mathbf{x}; \boldsymbol{\theta}) = \mathbf{x}^\top\boldsymbol{\theta}$ or $g(\mathbf{x}; \boldsymbol{\theta}) = 1/(1 + e^{-\mathbf{x}^\top\boldsymbol{\theta}})$. The problem of optimizing over all data is reduced by approximating the inner sum of the objective function with the second-order Taylor expansion around $\mathbf{L}\mathbf{s}^{(t)} = \boldsymbol{\theta}^{(t)}$ where

$$\boldsymbol{\theta}^{(t)} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\frac{1}{N_t}\sum_{i=1}^{N_t}\mathcal{L}\left(g(\mathbf{x}_i^{(t)}; \boldsymbol{\theta}), y_i^{(t)}\right) \tag{3.20}$$

is the the single-task parameter vector. The constant term of the Taylor expansion is ignored because it doesn't affect the minimizer, and there is no linear term in the expansion because $\boldsymbol{\theta}^{(t)}$ is defined as the minimizer for task t. This leaves the second order term of the approximation which is $\frac{1}{N_t}\|\boldsymbol{\theta}^{(t)} - \mathbf{L}\mathbf{s}^{(t)}\|_{\mathbf{D}^{(t)}}^2$, where

$$\mathbf{D}^{(t)} = \frac{\partial^2}{\partial\boldsymbol{\theta}^{(t)}\partial\boldsymbol{\theta}^{(t)\top}}\frac{1}{2N_t}\sum_{i=1}^{N_t}\mathcal{L}\left(g(\mathbf{x}_i^{(t)}; \boldsymbol{\theta}), y_i^{(t)}\right) \tag{3.21}$$

is the Hessian matrix evaluated at $\boldsymbol{\theta}^{(t)}$. This Taylor approximation is substituted into the objective function (3.19) to arrive at the new objective function

$$\varepsilon_T(\mathbf{L}, \mathbf{S}) = \frac{1}{T}\sum_{t=1}^{T}\left[\frac{1}{N_t}\|\boldsymbol{\theta}^{(t)} - \mathbf{L}\mathbf{s}^{(t)}\|_{\mathbf{D}^{(t)}}^2 + \mu\|\mathbf{s}^{(t)}\|_1\right] + \lambda\|\mathbf{L}\|_F^2. \tag{3.22}$$

By using this Taylor series approximation of (3.19), the N_t data points are only used to calculate the single-task parameters and the Hessian matrices. The rest of the algorithm relies on using these parameters and the calculated Hessians.

The second problem ELLA aims to solve is the need to retrain all tasks after all updates. This problem is solved by developing a method to update the MTL model by only updating $\mathbf{s}^{(t)}$ and $\mathbf{L}$ when task t is updated. This is done by solving the optimization problems,

$$\mathbf{s}_{m+1}^{(t)} = \underset{\mathbf{s}^{(t)}}{\operatorname{argmin}}\frac{1}{N_t}\|\boldsymbol{\theta} - \mathbf{L}_m\mathbf{s}^{(t)}\|_{\mathbf{D}^{(t)}}^2 + \mu\|\mathbf{s}_m^{(t)}\|_1 \tag{3.23}$$

$$\mathbf{L}_{m+1} = \underset{\mathbf{L}}{\operatorname{argmin}}\frac{1}{T}\sum_{t=1}^{T}\left[\frac{1}{N_t}\|\boldsymbol{\theta} - \mathbf{L}\mathbf{s}_m^{(t)}\|_{\mathbf{D}^{(t)}}^2\right] + \lambda\|\mathbf{L}\|_F^2, \tag{3.24}$$

Algorithm 2 ELLA

```
1:  $T \leftarrow 0$
2:  $\mathbf{L} \leftarrow \mathbf{0}_{d\times p}$
3:  $\mathbf{A} \leftarrow \mathbf{0}_{dp\times dp}$
4:  $\mathbf{b} \leftarrow \mathbf{0}_{dp\times 1}$
5:  for $(\mathbf{X}_{new}, \mathbf{y}_{new}, t)$ in NewUpdate do
6:      if NewTask then
7:          $T \leftarrow T + 1$
8:          $\mathbf{X}^{(t)} \leftarrow \mathbf{X}_{new}, \quad \mathbf{y}^{(t)} \leftarrow y_{new}$
9:      else
10:         $\mathbf{A} \leftarrow \mathbf{A} - \frac{1}{N_t}(\mathbf{s}^{(t)}\mathbf{s}^{(t)\top}) \otimes \mathbf{D}^{(t)}$
11:         $\mathbf{b} \leftarrow \mathbf{b} - \frac{1}{N_t}\operatorname{vec}\left(\mathbf{D}^{(t)}\boldsymbol{\theta}^{(t)}\mathbf{s}^{(t)\top}\right)$
12:         $\mathbf{X}^{(t)} \leftarrow [\mathbf{X}^{(t)}\ \mathbf{X}_{new}], \quad \mathbf{y}^{(t)} \leftarrow [\mathbf{y}^{(t)}; y_{new}]$
13:     end if
14:     $\boldsymbol{\theta}^{(t)} \leftarrow \text{singleTaskLearner}(\mathbf{X}^{(t)}.\mathbf{y}^{(t)})$
15:     $\mathbf{D}^{(t)} \leftarrow (3.21)$
16:     $\mathbf{L} \leftarrow \text{reinitialize}(\mathbf{L})$
17:     $\mathbf{s}^{(t)} \leftarrow (3.23)$
18:     $\mathbf{A} \leftarrow \mathbf{A} + \frac{1}{N_t}(\mathbf{s}^{(t)}\mathbf{s}^{(t)\top}) \otimes \mathbf{D}^{(t)}$
19:     $\mathbf{b} \leftarrow \mathbf{b} + \frac{1}{N_t}\operatorname{vec}\left(\mathbf{D}^{(t)}\boldsymbol{\theta}^{(t)}\mathbf{s}^{(t)\top}\right)$
20:     $\mathbf{L} \leftarrow \operatorname{mat}\left((\frac{1}{T}\mathbf{A} + \lambda\mathbf{I}_{dk\times dk})^{-1}\frac{1}{T}\mathbf{b}\right)$
21: end for
```

for the mth addition or update of a task. The process starts by calculating $\boldsymbol{\theta}^{(t)}$ and $\mathbf{D}^{(t)}$ for the updated task and solving for $\mathbf{s}_{m+1}^{(t)}$ using some numeric optimization method e.g., alternating direction method of multipliers (ADMM) [23] or regression shrinkage and selection (RSS) [17]. After solving for $\mathbf{s}_{m+1}^{(t)}$, (3.24) can now be solved by setting its partial derivative with respect to $\mathbf{L}$ equal to $\mathbf{0}$.

$$\begin{aligned}&\frac{\partial}{\partial \mathbf{L}}\left\{\frac{1}{T}\sum_{t=1}^{T}\left[\frac{1}{N_t}\|\boldsymbol{\theta} - \mathbf{L}\mathbf{s}_m^{(t)}\|_{\mathbf{D}^{(t)}}^2\right] + \lambda\|\mathbf{L}\|_F^2\right\} \\ =&\frac{1}{T}\sum_{t=1}^{T}\frac{2}{N_t}\left[\mathbf{D}^{(t)}\mathbf{L}\mathbf{s}^{(t)}\mathbf{s}^{(t)\top} - \mathbf{D}^{(t)}\boldsymbol{\theta}^{(t)}\mathbf{s}^{(t)\top}\right] + 2\lambda\mathbf{L}.\end{aligned} \tag{3.25}$$

By vectorizing this partial derivative and using the property,

$$\operatorname{vec}(\mathbf{ABC}) = (\mathbf{C}^\top \otimes \mathbf{A})\operatorname{vec}(\mathbf{B}), \tag{3.26}$$

where $\otimes$ is the Kronecker product, this process results in the system of equations

$$\begin{aligned}&\left[\frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}(\mathbf{s}^{(t)}\mathbf{s}^{(t)\top})\otimes\mathbf{D}^{(t)}+\lambda\mathbf{I}_{dk\times dk}\right]\mathrm{vec}(\mathbf{L}) \\ =&\frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\,\mathrm{vec}\left(\mathbf{D}^{(t)}\boldsymbol{\theta}^{(t)}\mathbf{s}^{(t)\top}\right),\end{aligned} \tag{3.27}$$

where $\mathbf{I}_{dk\times dk}$ is the identity matrix of size $dk \times dk$. Solving this system of equations for $\mathbf{L}$ gives the update equation as

$$\mathbf{L}_{m+1} = \mathrm{mat}(\mathbf{A}^{-1}\mathbf{b}), \tag{3.28}$$

where

$$\mathbf{A} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\left[(\mathbf{s}^{(t)}\mathbf{s}^{(t)\top})\otimes\mathbf{D}^{(t)}\right]+\lambda\mathbf{I}_{dk\times dk}, \tag{3.29}$$

$$\mathbf{b} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\,\mathrm{vec}\left(\mathbf{D}^{(t)}\boldsymbol{\theta}^{(t)}\mathbf{s}^{(t)\top}\right), \tag{3.30}$$

and $\mathrm{mat}(\cdot)$ is the function that reshapes the input vector to a matrix and is the inverse of the vectorization function $\mathrm{vec}(\cdot)$. $\mathbf{A}$ and $\mathbf{b}$ are updated incrementally to avoid summing over all tasks after each update. The entire process of ELLA is shown in Algorithm 2. The main loop is executed every time some task t is added or received new training data, $(\mathbf{X}_{new}, \mathbf{y}_{new})$. On line 16 of this algorithm, all zero columns of $\mathbf{L}$ are reinitialized either randomly or to the single task parameter vector $\boldsymbol{\theta}^{(t)}$ calculated on line 14 of Algorithm 2.

In each update of ELLA, the algorithm solves for $\boldsymbol{\theta}^{(t)}$ and $\mathbf{D}^{(t)}$ with a single task learner and updates $\mathbf{s}^{(t)}$ and $\mathbf{L}$. The single task learned has some time complexity $O(\xi(d, N_t))$ which depends on the loss function and single task learner used to solve (3.20). The update of $\mathbf{s}^{(t)}$ involves the eigen-decomposition of $\mathbf{D}^{(t)}$ which requires $O(d^3)$, multiplication of the square root of $\mathbf{D}^{(t)}$ with $\mathbf{L}$ which is of $O(d^2p)$ and solving a lasso problem which needs $O(dk^2)$. It has a total time complexity $O(d^3 + d^2p + dp^2)$. The update of $\mathbf{L}$ involves inverting the $dp \times dp$ matrix $\mathbf{A}$. Because of the low-rank updates of $\mathbf{A}$, its inverse can be calculated with a complexity of $O(d^pk^2)$ by recursively updating the eigen-decomposition [24]. The total time complexity of an update in ELLA as $O(d^3p^2 + \xi(d, N_t))$.

3.4 Conclusion

In this chapter, we reviewed two MTL methods, namely GO-MTL and ELLA. Both of these methods rely on the assumption that the parameters for related tasks lie on a low-dimensional subspace. The algorithms find the latent parameters that represent this subspace along with individual task parameters represented as sparse combinations of these latent parameters. GO-MTL introduced the use of sparse coefficients to determine which basis vectors are used by each task. This method can be slow to converge, especially with large numbers of tasks and high-dimensional datasets. ELLA enables lifelong learning capability by adding new tasks and updating existing ones while also increasing the speed of training by using the Taylor expansion of GO-MTL's objective function and only updating a task when it is added or receives new data. However, the use of very high-dimensional matrices can make this method unusable particularly for high-dimensional problems or learning on resource-constrained platforms.

CHAPTER 4

A GRADIENT DESCENT-BASED MTL ALGORITHM

4.1 Introduction

The two MTL algorithms reviewed in Chapter 3, GO-MTL [1] and ELLA [2], are effective algorithms though they are sensitive to the dimension of the data being used. That it, they cannot be used for problems with high-dimensional data. Another approach to solve the MTL problem defined by ELLA is to use gradient descent to minimize the objective function in (3.19). This method finds a local minimum without the need to calculate any Hessian matrices. In particular, when using high-dimensional data, computing Hessian matrices becomes very costly. Additionally, the Kronecker products used in ELLA in the calculation of the dictionary of latent tasks produce even larger matrices that must be inverted. This operation can be very time-consuming or even impossible to calculate when working with high-dimensional data. Gradient descent algorithm [20] avoids any of these time-consuming operations. Nevertheless, some drawbacks of this approach are the number of iterations required to converge and the choice of the step size. While each iteration of gradient descent is faster, it requires many more iterations to converge. In general, this approach is slower for low-dimensional problems and faster for high-dimensional problems.

Organization of the chapter is as follows. Section 4.2 explains the application of gradient descent to ELLA's general MTL problem. This is followed by the application of two loss functions for the purpose of classification. The loss functions used are log loss from logistic regression and hinge loss used in many SVMs. Section 4.4 gives concluding remarks on the method developed in this chapter.

4.2 Gradient Descent

Let us reconsider the cost function in (3.19).

$$e_T(\mathbf{L}, \mathbf{S}) = \frac{1}{T} \sum_{t=1}^{T} \left\{ \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(g(\mathbf{x}_i^{(t)}; \mathbf{L}\mathbf{s}^{(t)}), y_i^{(t)}) + \mu \|\mathbf{s}^{(t)}\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2 \tag{4.1}$$

To allow for the use of high-dimensional data, we minimize the cost function with the use of the gradient descent algorithm [20]. This algorithm uses the gradient to make small updates to the parameters to decrease the values of the objective function toward a local minimum. Taking the partial derivatives of this cost function with respect to **L** and **S** yields the following update equations.

$$\begin{aligned} \mathbf{S}_{n+1} &= \mathbf{S}_n - \alpha_n \left.\frac{\partial e_T}{\partial \mathbf{S}}\right|_{\mathbf{S}=\mathbf{S}_n} \\ \mathbf{L}_{n+1} &= \mathbf{L}_n - \beta_n \left.\frac{\partial e_T}{\partial \mathbf{L}}\right|_{\mathbf{L}=\mathbf{L}_n}, \end{aligned} \tag{4.2}$$

where α_n and β_n are the step sizes for the descent over **S** and **L**, respectively. Optimization is performed by applying the updates iteratively in an alternating pattern until convergence. A simple option to choose α_n and β_n is to set them to constants. This method requires small step sizes to guarantee convergence and hence may be slow. Instead, we choose the step sizes that minimize the objective function the most for each iteration using the following one-dimensional optimization problems,

$$\begin{aligned} \alpha_n &= \operatorname*{argmin}_{\alpha} e_T \left(\mathbf{L}_n, \mathbf{S}_n - \alpha \left.\frac{\partial e_T}{\partial \mathbf{S}}\right|_{\mathbf{S}=\mathbf{S}_n} \right) \\ \beta_n &= \operatorname*{argmin}_{\beta} e_T \left(\mathbf{L}_n - \beta \left.\frac{\partial e_T}{\partial \mathbf{L}}\right|_{\mathbf{L}=\mathbf{L}_n}, \mathbf{S}_n \right). \end{aligned} \tag{4.3}$$

These step sizes can be calculated using any search method such as the golden search method [20]. This option increases the computation time for each iteration but decreases the number of iterations needed to converge.

Alternatively, we can use batch learning to estimate the derivatives of (4.1) along with a momentum term [6]. Batch learning uses a small subset of training data in each iteration of

(4.2). By separating the training set into batches of size M with M_t samples for task t, the objective function (4.1) can be estimated in each iteration by taking the inner summation only over samples in the iteration's batch.

The momentum term is added to reduce training time and avoid getting stuck in local minima. Momentum adds a fraction of the previous update of the variables to each update. With momentum coefficients $0 \leq \eta, \nu < 1$ for $\mathbf{S}$ and $\mathbf{L}$, respectively, the update equations become

$$\begin{aligned} \mathbf{S}_{n+1} &= \mathbf{S}_n - \alpha_n \frac{\partial e_T}{\partial \mathbf{S}}\bigg|_{\mathbf{S}=\mathbf{S}_n} + \eta \Delta \mathbf{S}_n \\ \mathbf{L}_{n+1} &= \mathbf{L}_n - \beta_n \frac{\partial e_T}{\partial \mathbf{L}}\bigg|_{\mathbf{L}=\mathbf{L}_n} + \nu \Delta \mathbf{L}_n, \end{aligned} \tag{4.4}$$

where $\Delta \mathbf{S}_n = \mathbf{S}_n - \mathbf{S}_{n-1}$ and $\Delta \mathbf{L}_n = \mathbf{L}_n - \mathbf{L}_{n-1}$ are the previous updates.

Note that the objective function contains the non-differentiable ℓ_1-norm. Thus, to guarantee convergence, we use the differentiable approximation to the ℓ_1-norm i.e.

$$\|\mathbf{s}\|_1 = \sum_i |s_i| \approx \sum_i \sqrt{s_i^2 + \gamma}, \tag{4.5}$$

where $\gamma > 0$ and s_i is the i^{th} element of the vector $\mathbf{s}$. This approximation approaches the true ℓ_1-norm as γ approaches 0. This approximation is differentiable with the derivative

$$\frac{\partial \|\mathbf{s}\|_1}{\partial s_i} \approx \frac{2 s_i}{\sqrt{s_i^2 + \gamma}} \tag{4.6}$$

This derivative can be used in the updates of $\mathbf{s}^{(t)}$ in (4.2).

As mentioned in the previous chapter, for classification problems, a logistic regression model is typically used in both GO-MTL and ELLA. For class labels $y \in \{0, 1\}$, the unipolar logistic function,

$$\hat{y} = g(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1 + e^{-\mathbf{x}^\top \boldsymbol{\theta}}} \tag{4.7}$$

and the log-loss function [25]

$$\mathcal{L}(\hat{y}, y) = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y}) \tag{4.8}$$

are generally used. When using these functions, a single-task MTL problem is equivalent to a regularized logistic regression problem. Taking the partial derivatives of (4.1) using the logistic function along with the log-loss gives

$$\frac{\partial e_T}{\partial \mathbf{L}} = \frac{-1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\left[(y_i^{(t)} - \hat{y}_i^{(t)})\mathbf{x}_i^{(t)}\mathbf{s}^{(t)\top}\right] + \lambda 2\mathbf{L} \tag{4.9}$$

$$\frac{\partial e_T}{\partial \mathbf{s}^{(t)}} = \frac{-\mathbf{L}^\top}{T\ N_t}\sum_{i=1}^{N_t}\left[(y_i^{(t)} - \hat{y}_i^{(t)})\mathbf{x}_i^{(t)} + \mu\frac{\partial\|\mathbf{s}\|_1}{\partial \mathbf{s}}\bigg|_{\mathbf{s}=\mathbf{s}^{(t)}}\right] \tag{4.10}$$

where $\hat{y}_i^{(t)} = g(\mathbf{x}_i^{(t)}; \mathbf{L}\mathbf{s}^{(t)})$. The iterations of (4.2) can now be performed with (4.6).

Another popular loss function in the hinge loss which is used in support vector machines (SVM) to find the hyperplane with the greatest separation margin between two classes. Using classification labels $y \in \{-1, 1\}$, the hinge loss is

$$\mathcal{L}(\hat{y}, y) = \max(0, 1 - \hat{y}y) \tag{4.11}$$

This loss function is not differentiable at $\hat{y}y = 1$. We can instead use the quadratically smoothed hinge loss described in [26],

$$\mathcal{L}(\hat{y}, y) = \begin{cases} \frac{1}{2\gamma}\max(0, 1-\hat{y}y)^2 & \hat{y}y \geq 1-\gamma \\ 1 - \frac{\gamma}{2} - \hat{y}y & \hat{y}y < 1-\gamma \end{cases} \tag{4.12}$$

where $\gamma > 0$. This approximation approaches the hinge loss as $\gamma \to 0$. The derivative of this loss function is

$$\frac{\partial \mathcal{L}(\hat{y}, y)}{\partial \hat{y}} = \begin{cases} 0 & \hat{y}y \geq 1 \\ \frac{y}{\gamma}(\hat{y} - 1) & 1 > \hat{y}y \geq 1-\gamma \\ -y & 1-\gamma > \hat{y}y \end{cases} \tag{4.13}$$

If we use the derivative of this smoothed hinge loss along with $\hat{y} = g(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{x}^\top\boldsymbol{\theta}$ instead of the logistic function in (4.9) and (4.10), we obtain

$$\frac{\partial e_T}{\partial \mathbf{L}} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\left[\frac{\partial \mathcal{L}(\hat{y}, y_i^{(t)})}{\partial \hat{y}}\bigg|_{\hat{y}=\hat{y}_i^{(t)}}\mathbf{x}_i^{(t)}\mathbf{s}^{(t)\top}\right] + \lambda 2\mathbf{L} \tag{4.14}$$

$$\frac{\partial e_T}{\partial \mathbf{s}^{(t)}} = \frac{\mathbf{L}^\top}{T\ N_t}\sum_{i=1}^{N_t}\left[\frac{\partial \mathcal{L}(\hat{y}, y_i^{(t)})}{\partial \hat{y}}\bigg|_{\hat{y}=\hat{y}_i^{(t)}}\mathbf{x}_i^{(t)} + \mu\frac{\partial\|\mathbf{s}\|_1}{\partial \mathbf{s}}\bigg|_{\mathbf{s}=\mathbf{s}^{(t)}}\right] \tag{4.15}$$

where $\hat{y}_i^{(t)} = g(\mathbf{x}_i^{(t)}; \mathbf{L}\mathbf{s}^{(t)}) = \mathbf{x}_i^{(t)\top}\mathbf{L}\mathbf{s}^{(t)}$. These equations are solved iteratively until convergence.

4.3 Computational Complexity

Each iteration of our algorithm with either log loss or hinge loss calculates the gradient of (4.1) with respect to $\mathbf{L}$ and $\mathbf{S}$. Calculation of $\frac{\partial e_T}{\partial \mathbf{L}}$ has time complexity $O(Ndp)$. The calculations of each $\frac{\partial e_T}{\partial s^{(t)}}$ has time complexity $O(N_t dp)$ for a total of $O(Ndp)$ for all gradients. Evaluation of (4.1) using the golden search method has time complexity $O(Ndp)$. The update equations for $\mathbf{S}$ and $\mathbf{L}$ in (4.2) have time complexity $O(pT)$ and $O(dp)$, respectively. This makes each iteration of the gradient descent algorithm $O(Ndp)$. When using batch learning, each iteration uses a single batch with $M < N$ training samples leading to an overall complexity of $O(Mdp)$.

Both GO-MTL and this gradient descent-based algorithm are iterative algorithms, so comparing their computational complexities to each other and to ELLA is difficult. Each iteration of GO-MTL contains two other iterative algorithms. The iterations of these algorithms have complexities of $O(Nd^2p^2 + d^3p^3)$ and $O(Ndp + Np^2)$. Each iteration of the

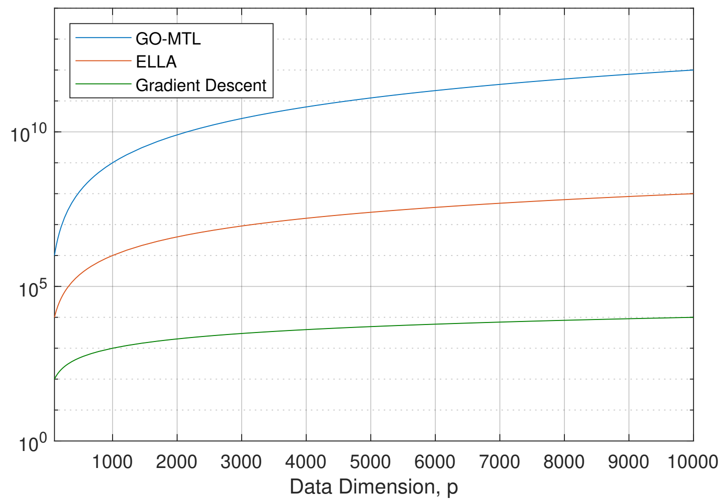


Figure 4.1: Computational complexity of different MTL algorithms as a function of data dimension

gradient descent-based algorithm has a complexity of $O(Ndp)$. Each update of ELLA has a complexity of $O(d^3p^2)$. A single iteration of GO-MTL has a higher computational complexity than an update in ELLA and an iteration of our MTL algorithm. When using high-dimensional data, a single iteration of GO-MTL can take a very long time making the algorithm unusable. High-dimensional data with a large number of dictionary atoms makes ELLA unusable with its high complexity. Our gradient descent-based algorithm has iterations with low complexity making them easy to calculate especially with high-dimensional data. To compare the computational complexity of our gradient descent-based algorithm with those covered in Chapter 3, Figure 4.1 is generated which shows how the complexities of each algorithm grows as the dimension of the data increases. While this figure does not include the number of iterations each algorithm needs to converge, it still demonstrates how a single iteration or update of GO-MTL or ELLA may become too impractical to solve.

4.4 Conclusion

ELLA and GO-MTL cannot be used with high-dimensional data due to extremely high computational costs. This calls for a new method specifically for high dimensional data. This chapter introduced a new gradient descent-based algorithm for training MTL models that use ELLA's objective function. This algorithm can be applied to high-dimensional problems without calculating any Hessian matrices or performing any matrix inversion operations. The chapter also explained how to set up an MTL model for the purpose of classification by using the logistic loss from logistic regression or the hinge loss used in many SVMs. The computational complexity of our algorithm is $O(Ndp)$ per iteration compared to $O(Nd^2p^2 + d^3p^3)$ and $O(Ndp + Np^2)$ for iterations of GO-MTL and $O(d^3p^2)$ for updates in ELLA.

CHAPTER 5

KERNEL MULTI-TASK LEARNING

5.1 Introduction

The classification methods explained in Chapters 3 and 4 are all examples of linear classifiers. Linear classifiers work best when classifying data that is linearly separable. Non-linearly separable data is guaranteed to have some classification error. In many cases, data cannot be classified with a linear classifier while achieving acceptable error rates. This problem may be solved by mapping the data to a high-dimensional feature space where the mapped data becomes linearly separable.

Kernel methods map data to a higher dimensional space using a non-linear kernel-producing map $\mathbf{\Phi} : \mathbb{R}^d \longrightarrow \mathbb{R}^m$ where $m \gg d$. The m-dimensional mapped data may be too large to use directly, so we instead use kernel tricks which rely only on the inner products between mapped data points. The inner products of mapped data points in the kernel method can be calculated from the original points with a kernel function.

In this chapter, we introduce kernel multi-task learning (KMTL) as a kernelized version of MTL [1, 2] using the gradient descent-based algorithm found in Chapter 4. Organization of this chapter is as follows. Section 5.2 explains how the objective function in Chapter 4 for the linear gradient descent-based method can be rewritten in terms of the inner products of data vectors. This is followed by the addition of a kernel function and the use of both logistic and hinge loss functions. In Section 5.3, we compare the computational complexity of this algorithm to those of previous methods. Section 5.4 gives a conclusion of the changes made to the algorithm to kernelize it.

5.2 Kernel Multi-Task Learning

The kernel method maps the data to a very high, or possibly infinite, dimensional space with a kernel-producing [6] mapping $\mathbf{\Phi} : \mathcal{X} \longrightarrow \mathcal{F}$ where $\mathcal{X}$ is the input space, and $\mathcal{F}$ is the high-dimensional feature space. Because the mapped data vectors $\mathbf{\Phi}(\mathbf{x}_i^{(t)})$ may have too high of a dimension, we rewrite the objective function (5.1) in terms of the inner products of the data vectors which does not increase the dimensionality. We begin by writing the objective function of the linear MTL (4.1) with the linear transfer function, $g(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{x}^\top \boldsymbol{\theta}$. This allows for the objective function to be rewritten in terms of the inner products of the data vectors. Recall that the objective function for GO-MTL in (3.1) was

$$e_T(\mathbf{L}, \mathbf{S}) = \frac{1}{T} \sum_{t=1}^{T} \left\{ \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\mathbf{x}_i^{(t)\top} \mathbf{L} \mathbf{s}^{(t)}, y_i^{(t)}) + \mu \|\mathbf{s}^{(t)}\|_1 \right\} + \lambda \|\mathbf{L}\|_F^2. \tag{5.1}$$

The columns in $\mathbf{L}$ lie in the span of $\mathcal{X}$ and hence we can write $\mathbf{L} = \mathbf{X}\mathbf{A}$ where $\mathbf{A} \in \mathbb{R}^{N \times p}$. Substituting this in (5.1) yields

$$\tilde{e}_T(\mathbf{A}, \mathbf{S}) = \frac{1}{T} \sum_{t=1}^{T} \left\{ \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}(\mathbf{k}_i^{(t)\top} \mathbf{A} \mathbf{s}^{(t)}, y_i^{(t)}) + \mu \|\mathbf{s}^{(t)}\|_1 \right\} + \lambda \|\mathbf{A}\|_{\mathbf{K}}^2, \tag{5.2}$$

where $\mathbf{k}_i^{(t)} = \mathbf{X}^\top \mathbf{x}_i^{(t)}$ and $\|\mathbf{A}\|_{\mathbf{K}}^2 = \mathrm{tr}(\mathbf{A}^\top \mathbf{K} \mathbf{A})$ with $\mathbf{K} = \mathbf{X}^\top \mathbf{X}$ being the Gram matrix [6]. This is very similar in form to the objective function in (5.1) with the linear transfer function. The only differences are the uses of $\mathbf{k}_i^{(t)}$ instead of $\mathbf{x}_i^{(t)}$, $\mathbf{A}$ instead of $\mathbf{L}$, and $\|\cdot\|_{\mathbf{K}}^2$ instead of the Frobenius norm.

To show that the columns of $\mathbf{L}$ lie in the span of $\mathcal{X}$, we start by letting $\mathbf{P}_{\mathbf{X}}$ be the orthogonal projection onto subspace $\mathcal{X}$ and $\mathbf{P}_{\mathbf{X}}^{\perp} = \mathbf{I} - \mathbf{P}_{\mathbf{X}}$ be its orthogonal complement subspace $\mathcal{X}^{\perp}$. Now, if we plug in $\mathbf{L} = \mathbf{P}_{\mathbf{X}}\mathbf{L} + \mathbf{P}_{\mathbf{X}}^{\perp}\mathbf{L}$ into our linear objective function (5.1), it should not change anything. Doing so yields

$$
\begin{aligned}
e_T(\mathbf{L},\mathbf{S}) =& \frac{1}{T}\sum_{t=1}^{T}\left\{\frac{1}{N_t}\sum_{i=1}^{N_t}\mathcal{L}(\mathbf{x}_i^{(t)\top}(\mathbf{P_X L}+\mathbf{P_X^\perp L})\mathbf{s}^{(t)}, y_i^{(t)}) + \mu\|\mathbf{s}^{(t)}\|_1\right\} \\
&+ \lambda\|\mathbf{P_X L}+\mathbf{P_X^\perp L}\|_F^2 \qquad (5.3)\\
=& \frac{1}{T}\sum_{t=1}^{T}\left\{\frac{1}{N_t}\sum_{i=1}^{N_t}\mathcal{L}(\mathbf{x}_i^{(t)\top}\mathbf{P_X L}\mathbf{s}^{(t)}, y_i^{(t)}) + \mu\|\mathbf{s}^{(t)}\|_1\right\} \\
&+ \lambda\|\mathbf{P_X L}\|_F^2 + \lambda\|\mathbf{P_X^\perp L}\|_F^2,
\end{aligned}
$$

since $P_{\mathbf{X}}^{\perp}\mathbf{x}_i^{(t)} = \mathbf{0}$. Also, it is clear that $\mathbf{L} = \mathbf{P_X L}$, in which case $\lambda\|\mathbf{P_X^\perp L}\|_F^2$ is reduced to 0 and the rest of the objective function is left unchanged. This implies that the optimal $\mathbf{L}$ is in the span of $\mathcal{X}$. Note instead of using all N training vectors, we can use a subset of M vectors as the basis for $\mathbf{L}$ in the matrix $\tilde{\mathbf{X}} \in \mathbb{R}^{d\times M}$.

Now that the objective function in (5.1) depends only on the inner product of data points, we can apply this method to the nonlinearly mapped data $\{\mathbf{\Phi}(\mathbf{x}_i^{(t)})\}$ where the objective function in (5.2) will be represented with kernel vector $\mathbf{k}_i^{(t)} = \mathbf{\Phi}(\mathbf{X})^\top\mathbf{\Phi}(\mathbf{x}_i^{(t)})$ and kernel Gram matrix $\mathbf{K} = \mathbf{\Phi}(\mathbf{X})^\top\mathbf{\Phi}(\mathbf{X})$. Each element is calculated with a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$ e.g., the Gaussian kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}}$ with parameter σ or the polynomial kernel $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top\mathbf{x}_j + c)^d$ with parameters c, d. As mentioned before, we can lower the dimension of these matrices and reduce the computational complexity by using the subset $\tilde{\mathbf{X}}$ of the training data instead of the full dataset $\mathbf{X}$ to generate the kernel matrix.

Minimization of this objective function is performed similar to that in Chapter 4. However, instead of minimizing over $\mathbf{L}$, we minimize over $\mathbf{A}$ with the update equations

$$
\begin{aligned}
\mathbf{S}_{n+1} &= \mathbf{S}_n - \alpha_n \left.\frac{\partial \tilde{e}_T}{\partial \mathbf{S}}\right|_{\mathbf{S}=\mathbf{S}_n} \qquad (5.4)\\
\mathbf{A}_{n+1} &= \mathbf{A}_n - \beta_n \left.\frac{\partial \tilde{e}_T}{\partial \mathbf{A}}\right|_{\mathbf{A}=\mathbf{A}_n}
\end{aligned}
$$

where α_n and β_n are chosen to be the step sizes. As in Chapter 4, these values can be chosen to be constants, or we can calculate the best step sizes at each iteration by solving the one-dimensional optimization problems

$$\alpha_n = \underset{\alpha}{\operatorname{argmin}}\ \tilde{e}_T\left(\mathbf{A}_n, \mathbf{S}_n - \alpha \frac{\partial \tilde{e}_T}{\partial \mathbf{S}}\bigg|_{\mathbf{S}=\mathbf{S}_n}\right) \tag{5.5}$$
$$\beta_n = \underset{\beta}{\operatorname{argmin}}\ \tilde{e}_T\left(\mathbf{A}_n - \beta \frac{\partial \tilde{e}_T}{\partial \mathbf{A}}\bigg|_{\mathbf{A}=\mathbf{A}_n}, \mathbf{S}_n\right)$$

using a search algorithm [20]. The ℓ_1 norm is unaffected by the process of kernelizing MTL, so we use the same approximation (4.5) and its derivative (4.6). Next, we show how to use KMTL for logistic regression and SVMs. To apply logistic regression to KMTL, we use the loss function

$$\mathcal{L}(\hat{y}, y) = y\ln(1+e^{-\hat{y}}) + (1-y)\ln(1+e^{\hat{y}}) \tag{5.6}$$

where $\hat{y}$ is the predicted output, and y is the true label. We use this logistic loss because KMTL uses $g(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{x}^\top\boldsymbol{\theta}$. The partial derivatives of (5.2) using this loss function are

$$\frac{\partial \tilde{e}_T}{\partial \mathbf{A}} = \frac{-1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\left[(y_i^{(t)} - \hat{y}_i^{(t)})\mathbf{k}_i^{(t)}\mathbf{s}^{(t)\top}\right] + \lambda 2\mathbf{K}\mathbf{A} \tag{5.7}$$

$$\frac{\partial \tilde{e}_T}{\partial \mathbf{s}^{(t)}} = \frac{-\mathbf{A}^\top}{T\ N_t}\sum_{i=1}^{N_t}\left[(y_i^{(t)} - \hat{y}_i^{(t)})\mathbf{k}_i^{(t)} + \mu\frac{\partial\|\mathbf{s}\|_1}{\partial \mathbf{s}}\bigg|_{\mathbf{s}=\mathbf{s}^{(t)}}\right] \tag{5.8}$$

where $\hat{y}_i^{(t)} = 1/(1+e^{-\mathbf{k}_i^{(t)\top}\mathbf{A}\mathbf{s}^{(t)}})$. These derivatives are used in the update equations (5.4).

Using the hinge loss approximation in (4.12) and its derivative in (4.13), the KMTL update equations use the following partial derivatives

$$\frac{\partial \tilde{e}_T}{\partial \mathbf{A}} = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{N_t}\sum_{i=1}^{N_t}\left[\frac{\partial \mathcal{L}(\hat{y}, y_i^{(t)})}{\partial \hat{y}}\bigg|_{\hat{y}=\hat{y}_i^{(t)}}\mathbf{k}_i^{(t)}\mathbf{s}^{(t)\top}\right] + \lambda 2\mathbf{K}\mathbf{A} \tag{5.9}$$

$$\frac{\partial \tilde{e}_T}{\partial \mathbf{s}^{(t)}} = \frac{\mathbf{A}^\top}{T\ N_t}\sum_{i=1}^{N_t}\left[\frac{\partial \mathcal{L}(\hat{y}, y_i^{(t)})}{\partial \hat{y}}\bigg|_{\hat{y}=\hat{y}_i^{(t)}}\mathbf{k}_i^{(t)} + \mu\frac{\partial\|\mathbf{s}\|_1}{\partial \mathbf{s}}\bigg|_{\mathbf{s}=\mathbf{s}^{(t)}}\right], \tag{5.10}$$

where $\hat{y}_i^{(t)} = \mathbf{k}_i^{(t)\top}\mathbf{A}\mathbf{s}^{(t)}$

5.3 Computation Complexity

The kernel matrix $\mathbf{K}$ and set of kernel vectors $\{\mathbf{k}_i^{(t)}\}$ are calculated once at the beginning with time complexities $O(M^2d)$ and $O(NMd)$, respectively. Each iteration of KMTL calculates the inner products $\frac{\partial \tilde{e}_T}{\partial \mathbf{A}}$ and $\frac{\partial \tilde{e}_T}{\partial \mathbf{s}^{(t)}}$. Calculation of $\frac{\partial \tilde{e}_T}{\partial \mathbf{A}}$ has time complexity $O(NMp)$

while calculation of $\frac{\partial \tilde{e}_T}{\partial \mathbf{s}^{(t)}}$ for each task has time complexity $O(N_t M p)$ with a total of $O(NMp)$ for all tasks. The total time complexity of this algorithm is $O(NMd)$ at the beginning and $O(NMp)$ for each iteration compared to the linear version with time complexity $O(Ndp)$ for each iteration.

Unlike the previous MTL algorithms, each iteration of KMTL does not depend on the dimension of the data. They instead depend on the number of training samples used to generate the kernel matrix $\mathbf{K}$. As with the linear gradient descent-based algorithm, KTML has a lower complexity than ELLA with complexity $O(d^3p^2)$ and GO-MTL with complexities $O(Nd^2p^2 + d^3p^3)$ and $O(Ndp + Np^2)$ allowing it to be more effective with high-dimension datasets.

5.4 Conclusion

The linear MTL models explained in the previous chapters do not perform well when used with datasets that are not linearly separable. A kernelized version of the gradient descent-based MTL algorithm can be used with many different datasets other than linearly separable ones while still having a low enough computational complexity to be practical. This chapter showed how the MTL algorithm in Chapter 4 can be kernelized to produce non-linear classifiers that only use the inner products between mapped data vectors. The computational complexity of this kernel algorithm is $O(NMd)$ along with $O(NMp)$ for each iteration compared to the linear algorithm with complexity $O(Ndp)$ for each iteration.

CHAPTER 6

TEST RESULTS AND PERFORMANCE COMPARISON

6.1 Introduction

To show the effectiveness of the proposed gradient descent-based MTL algorithm in Chapter 4, we compare this algorithm to ELLA [2] and single-task learners (STL) by training the classifiers in two experiments. The first experiment deals with classifying UXO vs. non-UXO targets from the FRM and TREX13 sonar datasets explained in chapter 2. This MTL model is constructed with two tasks from the FRM dataset and one task from a portion of the TREX13 dataset. The results of this experiment are presented as receiver operating characteristic (ROC) curves generated from testing data pulled from the rest of the TREX13 dataset. Performance of the classifiers in this experiment are shown by comparing the ROC curves and knee-point performance. The second experiment is classifying hand-written digits from the EMNIST [3] dataset. The MTL model in this experiment contains one task for each digit. Each task is a binary classification problem whose goal is to discriminate between its digit and the other nine digits. The results of this experiment are given as confusion matrices generated from a set of testing data.

We begin this chapter with Section 6.2 by explaining the setup of the UXO experiment on the TIER and TREX13 datasets and continue by presenting and analyzing the results. This is followed by Section 6.3 which contains the setup and results of the experiment on classification of EMNIST digits. Section 6.4 gives concluding remarks on the results presented in this chapter.

Table 6.1: AUC (left) and knee-point P_{CC} (right) for the three UXO classifiers using two loss functions.

	Log Loss	Hinge Loss
STL	0.817	0.813
MTL	0.840	0.804
ELLA	0.839	

P_{CC}	Log Loss	Hinge Loss
STL	0.7685	0.7596
MTL	0.7796	0.7370
ELLA	0.7718	

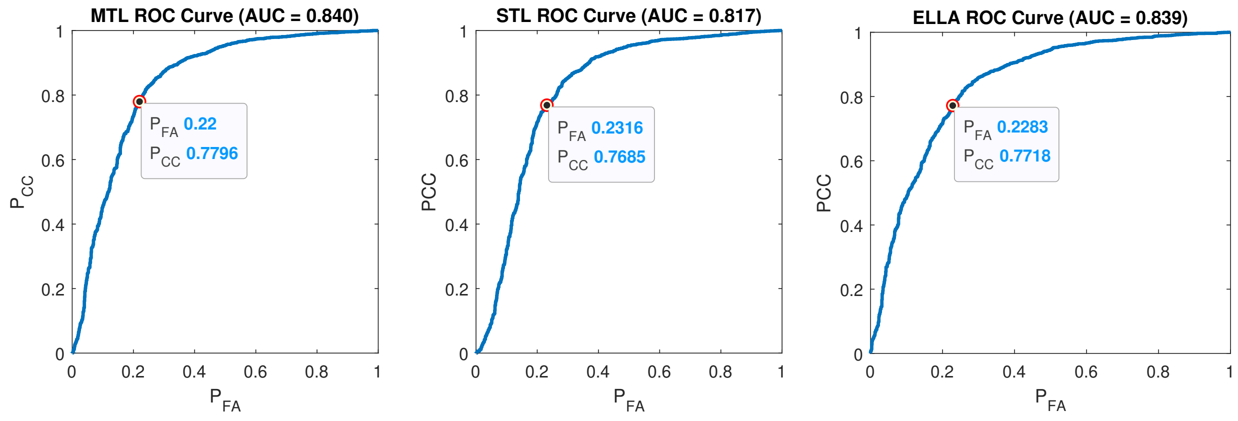


Figure 6.1: ROC curves from the gradient descent-based MTL (left), STL (middle), and ELLA (right) with log loss.

6.2 UXO vs. Non-UXO Classification

The purpose of this experiment is to show how MTL can be used to train a classification model when real data is limited while synthetic data is readily available. In this experiment, we used $T = 3$ tasks represented with $u = 2$ atoms in the dictionary matrix $\mathbf{L}$. To demonstrate this scenario with limited real data we used training set consisting of two tasks containing 2000 synthetic samples each from the FRM dataset and one task containing 400 real samples from the rest of the TREX13 dataset. Testing was performed on 4000 samples from the TREX13 dataset. Each data sample is a 272-dimensional vector containing acoustic color features at a single aspect angle. The MTL classifiers were trained using the gradient descent described in Chapter 4. The STL classifier with the log loss was trained using a logistic regression solver while the STL classifier with the hinge loss was trained using an SVM solver.

The results of the tests were generated by evaluating the output prediction $\hat{y} = \mathbf{x}^{\top}\boldsymbol{\theta}^{(t)}$

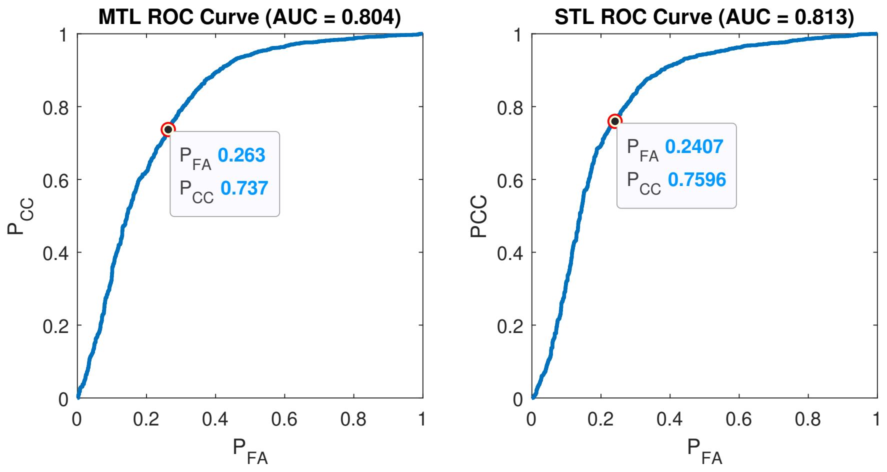


Figure 6.2: ROC curves from gradient descent-based MTL (left) and STL (right) with hinge loss.

at each task t, where $\mathbf{x}$ is the testing sample and $\boldsymbol{\theta}^{(t)}$ is the parameter vector for task t. For ELLA and the gradient descent MTL, the parameter vectors were $\boldsymbol{\theta}^{(t)} = \mathbf{Ls}^{(t)}$. The prediction and true label were used to generate the ROC curves shown in Figure 6.1 and Figure 6.2 when using the log loss and hinge loss, respectively. These figures also show the area under the curve (AUC) and knee-point (where $P_{CC}+P_{FA} = 1$) of each ROC curve. Table 6.1 shows the AUC and knee-point of each of the classifier's ROC curves. ELLA cannot be used with the hinge loss because the Hessian matrix $\mathbf{D}^{(t)}$ is not generally invertible. In this experiment, multi-task learning does not give a large improvement in classification performance when compared to the STL classification results. MTL provided a small improvement over STL when using log loss and performed worse than STL when using the hinge loss. The reason for this is unclear. However, the STL classifier's lower performance implies that the cause is not simply a problem with our MTL algorithm's ability to work with the hinge loss. With only a small number of tasks, MTL does not provide a significant benefit in classification performance.

Figure 6.3 shows the ROC curve from the orthogonal matching pursuit matched subspace classifier (OMP-MSC) used in [10]. This method performed better than the other classifiers with a correct classification rate of $P_{CC} = 0.787$ and an AUC of 0.866. For this two-class

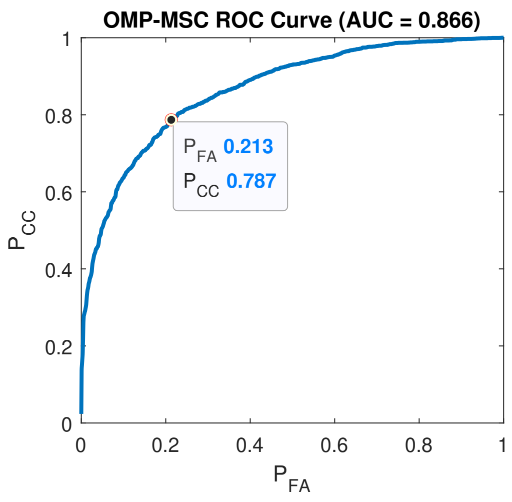


Figure 6.3: ROC curve from OMP-MSC with in-situ.

problem, OMP-MSC gives better results than either of the two MTL methods tested here.

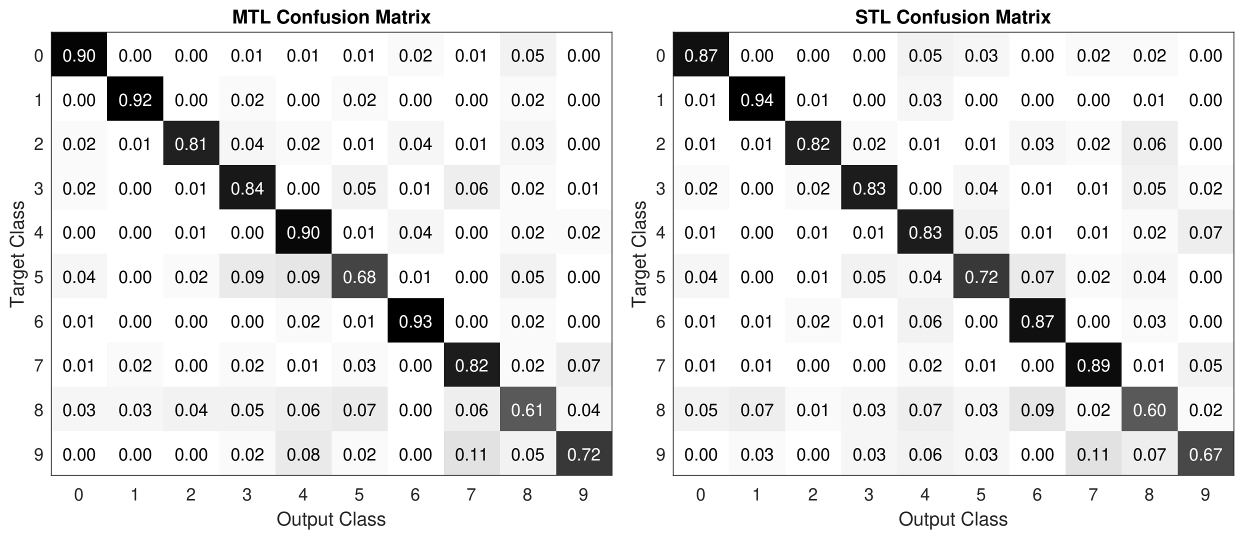
The main reason why MTL did not provide a significant improvement in performance over STL is likely the lack of overlap between TREX13 and the FRM tasks. Of the three tasks, the two from the FRM dataset were very similar, so the training of the MTL model grouped them together by making them use the same atom in matrix **L** while providing minimal overlap with the TREX13 task. In the next section, we show a classification problem with more tasks.

6.3 EMNIST Digits

In this section, we used MTL on the EMNIST [3] digits dataset described in Chapter 2. The MTL model in this experiment had $u = 6$ atoms in the dictionary matrix $\mathbf{L}$ to represent the parameters for $T = 10$ tasks. Each task had the goal of determining if any given image contains that digit. The training set consisted of 500 different images for each task. The testing set contained 4000 randomly selected images. We began by using principal component analysis [6] to reduce the dimension of the samples from 784 to 64. To make a decision on an image, the classifier evaluates the response of each task using the same method as in Section 6.2 and chooses the task that gives the highest output. This is performed on each image in the testing set in order to create the confusion matrices. Figure 6.5 shows the

Table 6.2: P_{CC} for the three EMNIST classifiers using two loss functions.

	Log Loss	Hinge Loss
STL	0.8033	0.8017
MTL	0.8442	0.8125
ELLA	0.7508	



MTL Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	0.90	0.00	0.00	0.01	0.01	0.01	0.02	0.01	0.05	0.00
1	0.00	0.92	0.00	0.02	0.00	0.02	0.00	0.00	0.02	0.00
2	0.02	0.01	0.81	0.04	0.02	0.01	0.04	0.01	0.03	0.00
3	0.02	0.00	0.01	0.84	0.00	0.05	0.01	0.06	0.02	0.01
4	0.00	0.00	0.01	0.00	0.90	0.01	0.04	0.00	0.02	0.02
5	0.04	0.00	0.02	0.09	0.09	0.68	0.01	0.00	0.05	0.00
6	0.01	0.00	0.00	0.00	0.02	0.01	0.93	0.00	0.02	0.00
7	0.01	0.02	0.00	0.02	0.01	0.03	0.00	0.82	0.02	0.07
8	0.03	0.03	0.04	0.05	0.06	0.07	0.00	0.06	0.61	0.04
9	0.00	0.00	0.00	0.02	0.08	0.02	0.00	0.11	0.05	0.72

STL Confusion Matrix

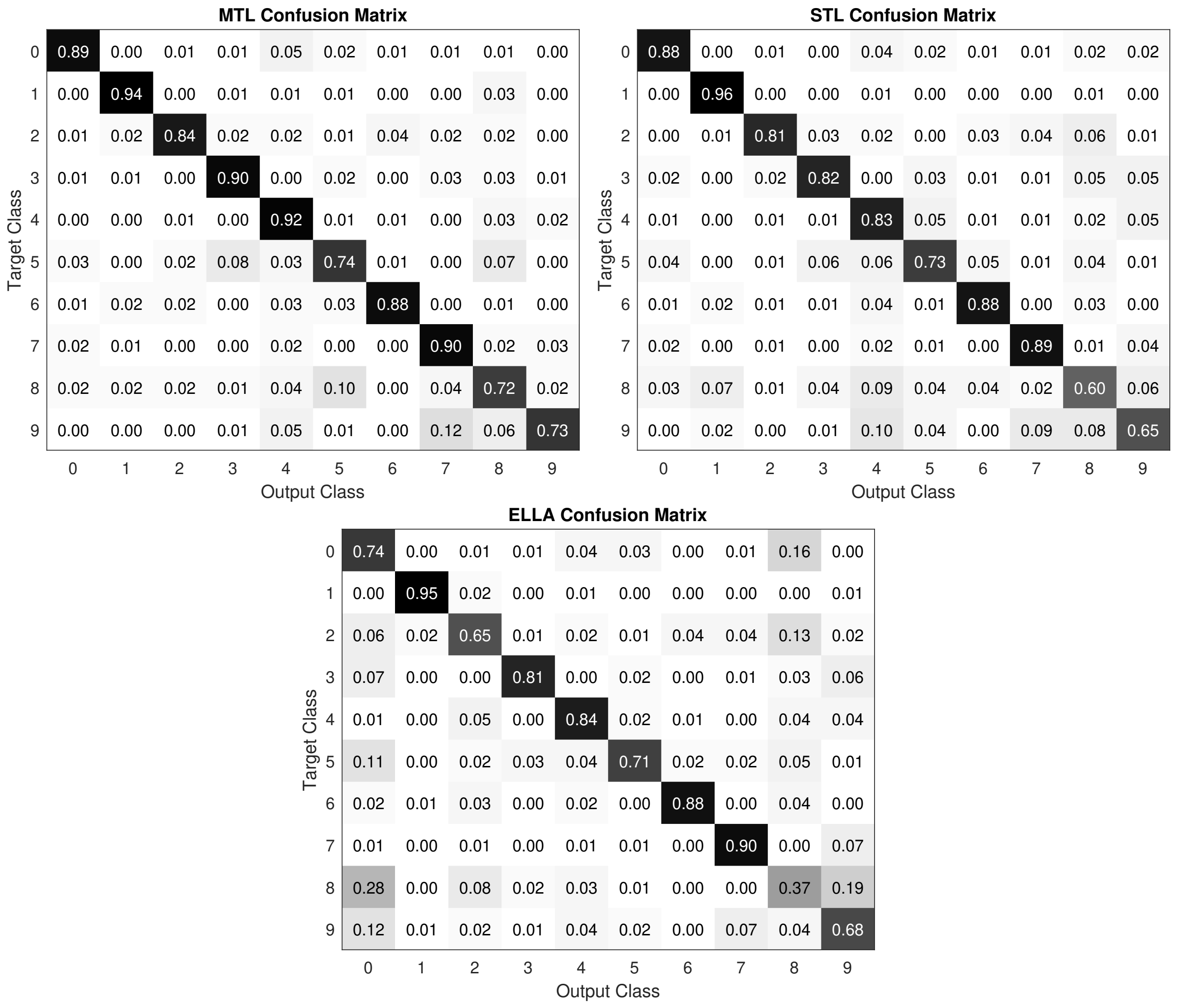
	0	1	2	3	4	5	6	7	8	9
0	0.87	0.00	0.00	0.00	0.05	0.03	0.00	0.02	0.02	0.00
1	0.01	0.94	0.01	0.00	0.03	0.00	0.00	0.00	0.01	0.00
2	0.01	0.01	0.82	0.02	0.01	0.01	0.03	0.02	0.06	0.00
3	0.02	0.00	0.02	0.83	0.00	0.04	0.01	0.01	0.05	0.02
4	0.01	0.00	0.01	0.01	0.83	0.05	0.01	0.01	0.02	0.07
5	0.04	0.00	0.01	0.05	0.04	0.72	0.07	0.02	0.04	0.00
6	0.01	0.01	0.02	0.01	0.06	0.00	0.87	0.00	0.03	0.00
7	0.01	0.01	0.00	0.00	0.02	0.01	0.00	0.89	0.01	0.05
8	0.05	0.07	0.01	0.03	0.07	0.03	0.09	0.02	0.60	0.02
9	0.00	0.03	0.00	0.03	0.06	0.03	0.00	0.11	0.07	0.67

Figure 6.4: Confusion matrices for our gradient descent-based MTL (left) and STL (right) with hinge loss

confusion matrices using the gradient descent-based MTL, STL, and ELLA when trained with the log loss. Figure 6.4 shows the confusion matrices from MTL and STL when using the hinge loss. The overall probability of correct classification for the classifiers are shown in Table 6.2. When using the hinge loss, the gradient descent-based MTL increased the accuracy on some digits and provided a slight improvement in the total classification accuracy. A similar behavior is observable when using the log loss function as shown in confusion matrix in 6.5. ELLA performed the worst on this dataset.

6.4 Conclusion

In this chapter, we compared ELLA and the gradient descent-based MTL algorithms described in Chapters 3 and 4 with single-task learners by using them to train classifiers for the datasets described in Chapter 2. The tests run on the sonar datasets showed little difference in performance between the three algorithms for both log loss and hinge loss.


MTL Confusion Matrix
STL Confusion Matrix
ELLA Confusion Matrix
Target Class
Output Class

MTL Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	0.89	0.00	0.01	0.01	0.05	0.02	0.01	0.01	0.01	0.00
1	0.00	0.94	0.00	0.01	0.01	0.01	0.00	0.00	0.03	0.00
2	0.01	0.02	0.84	0.02	0.02	0.01	0.04	0.02	0.02	0.00
3	0.01	0.01	0.00	0.90	0.00	0.02	0.00	0.03	0.03	0.01
4	0.00	0.00	0.01	0.00	0.92	0.01	0.01	0.00	0.03	0.02
5	0.03	0.00	0.02	0.08	0.03	0.74	0.01	0.00	0.07	0.00
6	0.01	0.02	0.02	0.00	0.03	0.03	0.88	0.00	0.01	0.00
7	0.02	0.01	0.00	0.00	0.02	0.00	0.00	0.90	0.02	0.03
8	0.02	0.02	0.02	0.01	0.04	0.10	0.00	0.04	0.72	0.02
9	0.00	0.00	0.00	0.01	0.05	0.01	0.00	0.12	0.06	0.73

STL Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	0.88	0.00	0.01	0.00	0.04	0.02	0.01	0.01	0.02	0.02
1	0.00	0.96	0.00	0.00	0.01	0.00	0.00	0.00	0.01	0.00
2	0.00	0.01	0.81	0.03	0.02	0.00	0.03	0.04	0.06	0.01
3	0.02	0.00	0.02	0.82	0.00	0.03	0.01	0.01	0.05	0.05
4	0.01	0.00	0.01	0.01	0.83	0.05	0.01	0.01	0.02	0.05
5	0.04	0.00	0.01	0.06	0.06	0.73	0.05	0.01	0.04	0.01
6	0.01	0.02	0.01	0.01	0.04	0.01	0.88	0.00	0.03	0.00
7	0.02	0.00	0.01	0.00	0.02	0.01	0.00	0.89	0.01	0.04
8	0.03	0.07	0.01	0.04	0.09	0.04	0.04	0.02	0.60	0.06
9	0.00	0.02	0.00	0.01	0.10	0.04	0.00	0.09	0.08	0.65

ELLA Confusion Matrix

	0	1	2	3	4	5	6	7	8	9
0	0.74	0.00	0.01	0.01	0.04	0.03	0.00	0.01	0.16	0.00
1	0.00	0.95	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.01
2	0.06	0.02	0.65	0.01	0.02	0.01	0.04	0.04	0.13	0.02
3	0.07	0.00	0.00	0.81	0.00	0.02	0.00	0.01	0.03	0.06
4	0.01	0.00	0.05	0.00	0.84	0.02	0.01	0.00	0.04	0.04
5	0.11	0.00	0.02	0.03	0.04	0.71	0.02	0.02	0.05	0.01
6	0.02	0.01	0.03	0.00	0.02	0.00	0.88	0.00	0.04	0.00
7	0.01	0.00	0.01	0.00	0.01	0.01	0.00	0.90	0.00	0.07
8	0.28	0.00	0.08	0.02	0.03	0.01	0.00	0.00	0.37	0.19
9	0.12	0.01	0.02	0.01	0.04	0.02	0.00	0.07	0.04	0.68

Figure 6.5: Confusion matrices for gradient descent-based MTL (left), STL (right), and ELLA (bottom) with log loss.

While the gradient descent-based MTL allows tasks to learn from each other to increase performance of all tasks, it does not provide a large benefit if the number of tasks is too small. Classification with $T = 10$ tasks on the EMNIST digits dataset showed a slight improvement over STL when using MTL and decreased performance with ELLA when using log loss.

CHAPTER 7

CONCLUSION AND FUTURE WORK

The main topic addressed in this thesis is the multi-task learning for the purpose of pattern classification. Two existing MTL algorithms, namely GO-MTL and ELLA, were reviewed, and their strengths and drawbacks were compared. The problem with these algorithms is that they are impractical while working with high-dimensional datasets. To solve this problem, a gradient descent-based algorithm was introduced as an alternative to GO-MTL and ELLA which can be used on high-dimensional datasets. This algorithm has a lower computational complexity per iteration than the previous algorithms. However, it may require many more iterations to converge owing to the nature of the gradient-based methods. A KMTL algorithm was introduced to train nonlinear classification models for datasets that are not linearly separable. By mapping the data vectors to very high-dimensional spaces, they may become linearly separable. The gradient descent-based algorithm was tested on the underwater UXO and EMNIST datasets and compared to ELLA and an STL. When used to classify UXO vs. non-UXO based on their acoustic color features, our gradient descent-based algorithm achieved a correct classification rate of approximately $P_{CC} = 0.78$ with the log loss while ELLA and an STL achieved rates of approximately $P_{CC} = 0.77$ and $P_{CC} = 0.77$, respectively. When used to classify digits in the EMNIST dataset, our algorithm achieved a rate of approximately $P_{CC} = 0.84$ with the log loss while ELLA and an STL achieved rates of approximately $P_{CC} = 0.75$ and $P_{CC} = 0.80$, respectively.

7.1 Future Work

The work done in this thesis has revealed new topics that need to be researched including improvements to the proposed gradient descent-based algorithm and methods to test the algorithm. A few possible ways to continue the research are listed below.

1 – Lifelong Learning: While our gradient descent-based algorithm has been shown to be effective on the datasets used during testing, it is not suitable for lifelong learning situations. As with GO-MTL, this algorithm requires all tasks to be present along with all training data. Any changes in the tasks requires the MTL model to be retrained from the beginning or with the previous parameters as the initial condition for training. A modified version of our gradient descent-based algorithm is needed to allow for updated tasks while maintaining a low computational complexity. ELLA [2] and CoLLA [4] allow for lifelong learning by only updating the dictionary matrix and the updated task's coefficient vector. A method similar to this may allow our algorithm to allow tasks to be updated without retraining all tasks.

2 – Improvement and Testing of KMTL: The KMTL algorithm introduced in this work was created to allow for nonlinear classifiers which can provide improved performance on many datasets. However, without proper testing, this method's effectiveness cannot be shown. Preliminary tests performed during the development showed the current algorithm's results to be inconsistent. That is, different tests with the same training and testing data had significantly different performance. It is possible that the algorithm is converging to local minima with high error. Ideas to consider researching include better ways to choose the initial conditions and regularization parameters. Improved initial values may allow the algorithm to converge to a more optimal solution, and varying the values of the regularization parameters may improve the performance achieved in these local minima.

3 – Testing UXO Dataset with Additional Tasks: The UXO dataset described in Chapter 2 contains just three tasks with only one of them containing real data. Data from new environments could be added to the MTL model as new tasks. Increasing the number of tasks allows for more transfer learning to occur and may more effectively demonstrate the benefit of using MTL for UXO vs. non-UXO classification. The UXO dataset can be expanded using real data collected during physical experiments at more locations and synthetic data generated using the FRM with different parameters.

REFERENCES

[1] A. Kumar and I. Daumé, Hal, "Learning Task Grouping and Overlap in Multi-task Learning," *arXiv e-prints*, p. arXiv:1206.6417, Jun. 2012.

[2] P. Ruvolo and E. Eaton, "Ella: An efficient lifelong learning algorithm," in *Proc. International Conference on Machine Learning*, 2013, pp. 507–515.

[3] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *arXiv e-prints*, p. arXiv:1702.05373, Feb. 2017.

[4] M. Rostami, S. Kolouri, K. Kim, and E. Eaton, "Multi-Agent Distributed Lifelong Learning for Collective Knowledge Acquisition," *arXiv e-prints*, p. arXiv:1709.05412, Sep. 2017.

[5] A. Barzilai and K. Crammer, "Convex Multi-Task Learning by Clustering," in *Proc. Eighteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 38, May 2015, pp. 65–73.

[6] S. S. Haykin, *Neural networks and learning machines*, 3rd ed. Upper Saddle River, NJ: Pearson Education, 2009.

[7] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[8] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021.

[9] D. S. Kargl, "Acoustic response of underwater munitions near a sediment interface: Measurement-model comparisons and classification schemes," *Final Report, SERDP Project MR-2231*, pp. 1–104, April 2015.

[10] J. J. Hall, M. R. Azimi-Sadjadi, S. G. Kargl, Y. Zhao, and K. L. Williams, "Underwater unexploded ordnance (uxo) classification using a matched subspace classifier with adaptive dictionaries," *IEEE Journal of Oceanic Engineering*, vol. 44, no. 3, pp. 739–752, 2019.

[11] S. G. Kargl, A. L. Espana, K. L. Williams, J. L. Kennedy, and J. L. Lopes, "Scattering from objects at a water–sediment interface: Experiment, high-speed and high-fidelity models, and physical insight," *IEEE Journal of Oceanic Engineering*, vol. 40, no. 3, pp. 632–642, 2014.

[12] M. Zampolli, A. Tesei, F. B. Jensen, N. Malm, and J. B. Blottman, "A computationally efficient finite element model with perfectly matched layers applied to scattering from axially symmetric objects," *The Journal of the Acoustical Society of America*, vol. 122, no. 3, pp. 1472–1485, 2007.

[13] K. L. Williams, S. G. Kargl, E. I. Thorsos, D. S. Burnett, J. L. Lopes, M. Zampolli, and P. L. Marston, "Acoustic scattering from a solid aluminum cylinder in contact with a sand sediment: Measurements, modeling, and interpretation," *The Journal of the Acoustical Society of America*, vol. 127, no. 6, pp. 3356–3371, 2010.

[14] P. T. Gough and D. W. Hawkins, "Unified framework for modern synthetic aperture imaging algorithms," *International journal of imaging systems and technology*, vol. 8, no. 4, pp. 343–358, 1997.

[15] P. Grother and K. Hanaoka, "Nist special database 19 handprinted forms and characters database 2nd edition," *National Institute of Standards and Technology, Tech. Rep*, 2016.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[18] E. M. Gafni and D. P. Bertsekas, "Two-metric projection methods for constrained optimization," *SIAM Journal on Control and Optimization*, vol. 22, no. 6, pp. 936–964, 1984.

[19] E. van den Berg and M. P. Friedlander, "Probing the pareto frontier for basis pursuit solutions," *SIAM Journal on Scientific Computing*, vol. 31, no. 2, pp. 890–912, 2008. [Online]. Available: http://link.aip.org/link/?SCE/31/890

[20] E. K. Chong and S. H. Zak, *An introduction to optimization.* Wiley, 2013.

[21] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Proc. Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[22] A. Argyriou, T. Evgeniou, and M. Pontil, "Convex multi-task feature learning," *Machine learning*, vol. 73, no. 3, pp. 243–272, 2008.

[23] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers.* Now Publishers Inc, 2011.

[24] K.-B. Yu, "Recursive updating the eigenvalue decomposition of a covariance matrix," *IEEE Transactions on Signal Processing*, vol. 39, no. 5, pp. 1136–1145, 1991.

[25] J. A. Nelder and R. W. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, no. 3, pp. 370–384, 1972.

[26] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Proc. Twenty-First International Conference on Machine Learning*, 2004, pp. 919–926.