REGULARIZED LEARNING IN MULTIPLE TASKS WITH RELATIONSHIPS

by

Anveshi Charuvaka
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Huzefa Rangwala, Dissertation Director
_____	Dr. Daniel Barbara, Committee Member
_____	Dr. Carlotta Domeniconi, Committee Member
_____	Dr. Igor Griva, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: _____	Fall Semester 2015 George Mason University Fairfax, VA

Regularized Learning in Multiple Tasks with Relationships

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Anveshi Charuvaka
Master of Science
George Mason University, USA, 2012
Bachelor of Science
Acharya Nagarjuna University, India, 2006

Director: Dr. Huzefa Rangwala, Professor
Department of Computer Science

Fall Semester 2015
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my parents who have always supported me in all my efforts.

Acknowledgments

These past six years in graduate schools have been a roller coaster ride with many ups and downs. Along the way, several people have helped me stay motivated and keep my sanity through the arduous times. First of all I am indebted to my advisor Dr. Huzefa Rangwala for guiding me all these years and pushing me hard to achieve my best. I would also like to thank my dissertation committee members Prof. Daniel Brabara, Prof. Carlotta Domeniconi, and Prof. Igor Griva for being helpful, encouraging and accommodating throughout the process. Their invaluable feedback has helped me improve my thesis. Without friends, the journey to the finish line would not have been as much fun as it had been, so I would also like to thank my current and past collegues at the Data Mining Lab, new friends that I have made in grad school, and old friends who have stuck around, with whom I have had several intellectually stimulating conversations and some good times. But most importantly, I am eternally indebted to my family who have always been supportive of my every endeavor.

Table of Contents

List of Tables

List of Figures

Abstract

REGULARIZED LEARNING IN MULTIPLE TASKS WITH RELATIONSHIPS

Anveshi Charuvaka, PhD

George Mason University, 2015

Dissertation Director: Dr. Huzefa Rangwala

We often encounter classification problems in real world as groups of tasks with complex interactions. In order to fully take advantage of the underlying information and deal with the unique aspects of these problems, we need methods than can utilize the available information in the learning process. In this thesis I present several methods to deal with the learning problems in domains with multiple tasks, with primary focus on large scale hierarchical classification and multi-task learning.

Hierarchies form an integral part in information organization in several application areas ranging from protein function classification to document classification. Automated methods for classification into hierarchies can lessen the burden of manual curation or completely obviate the need for it. Research in hierarchical classification is quite mature and several methods have been developed for small scale classification tasks, however, scalability of these methods to large scale problems is one aspect that is being actively researched. In large scale classification, the number of classes, number of training examples as well and number of features can be so large that learning on commodity hardware can be prohibitively expensive. With growing popularity of cluster computing paradigms, it is becoming possible to scale-out computation using intelligent distributed methods. In many large scale classification problems, a significant number of classes are also plagued by problems such as insufficient

training data or huge skew in class distributions. Large scale classification techniques should also try to mitigate these problem by leveraging data from related classes, for example, by using techniques inspired by the research in multi-task and transfer learning.

In this regard, we have developed a distributed implementation of hierarchical regularization formulation which is scalable to very large scale hierarchical classification problems. Even though the hierarchical regularization is able to effectively tackle some of the challenges associated with large scale classification, it can be computationally expensive to train, compared to its binary alternatives in one-vs-rest settings. To address this shortcoming of recursive regularization, we also propose an alternative formulation using cost sensitive learning where the cost of misclassification are defined with respect to the hierarchy. This formulation allows us to tackle the class imbalance in training as well. Cost sensitive hierarchical learning scheme is able to improve the learning efficiency while at the same time improving the classification performance.

In several domains we encounter multiple classification schemes over the same or similar data elements. These classification schema may provide different views and it is logical to exploit these different view in order to improve classification performance. One such prominent domain is protein structure classification, where multiple hierarchical classification schemes exist for protein structure classification. In this thesis, we also present techniques to exploit multi-task learning for learning across disparate classification hierarchies and show that the combined scheme is able to outperform learning on a single hierarchy.

Several multi-task learning formulations make the assumption that all tasks under consideration are symmetrically related. However, this assumption may not hold true for a large number of cases. Even the information about task relatedness is often not known. When symmetric task relationship assumptions are naively used in learning, the classification performance can be negatively affected, a phenomenon known as negative transfer. Therefore, effective methods should only selectively transfer information between tasks that can improve performance. Several methods have been developed to infer task relationship structure to deal with this issue. In this thesis, we also present a method that extends task relationship inference techniques to max margin formulation of support vector machines.

Chapter 1: Introduction

1.1 Motivation

Traditionally machine learning research has primarily focused on binary classification, but in the real world, we often encounter classification problems that are more complex. The straighforward generalization of classification to more than two classes, known as multi-class classification, involves a finite number of class labels where each training instance can be assigned exclusively to one of the labels. When instances can be assigned multiple labels from the set of possible labels, the classification problems are known as multi-label problems. In addition to these scenarios, several variations of classification problems are encountered in practice where the different classification tasks may have implicit or explicit relationships defined between them. Hierarchical classification, for example, can be seen as type of multi-class or multi-label classification problem where the labels have an explicit structure in the form of a hierarchy. In multi-task learning several tasks are jointly learned to improve generalization performance, and in the related area of transfer learning the goal is to improve the learning on target task, which typically has insufficient data, with the aid of data from other source tasks with sufficient training examples.

One of the easiest way to deal with the more complex scenarios is to use a reductionist approach where the decisions in multi-class, multi-label, hierarchical and multi-task classification settings are reduced to a sequence of binary decisions [2–5]. Although in many cases the performance of such reductions can be competitive, recent years have witnessed a tremendous progress in complex learning domains — which is far too diverse and numerous to enumerate — in order to use the available data and class relationships in much more intelligent manners to improve learning performance.

As mentioned earlier, the vital aspect of hierarchical classification is the hierarchical

relationship between the class labels. Hierarchical classification algorithm that can exploit this class structure effectively are able to improve the learning performance [2]. In multi-task learning, several related tasks are presented to the learning algorithm and the goal is to leverage the inductive bias from related tasks to improve the generalization performance of learned classifiers through joint learning. The underlying assumption is that the inductive bias that is helpful to many tasks is unlikely to be accidental. Several empirical and theoretical works support this notion [6–10]. Although these two research domains are often seen as disparate areas with little overlap, their unifying aspect is the existence of multiple tasks. In fact, a detailed analysis of the learning principles used in hierarchical classification and multi-task learning brings out some underlying commonalities. For example, hierarchical methods that use regularization schemes or priors to model neighboring nodes [11, 12] and multi-task learning methods that use task networks [13] to impose the bias that related tasks should have similar model vectors use similar technical approaches to model regularization. One aspect of this thesis is to bring the insights from multi-task learning to model problems in multiple hierarchical classification settings.

The availability of computational resources at low cost has catapulted us into the era of *big data*. Today, clusters of commodity computers using distributed computing frameworks [14,15] are able to achieve what was inconceivable a few years ago even with supercomputers. This trend has led to a growth in interest in web-scale problems, which in turn has precipitated the need for learning algorithm that are scalable to thousands and millions of classes, and often, very large feature spaces. However, most of the early developments in machine learning deal with small scale problems which can not be trivially scaled to tackle web-scale problems. In addition to high computational demands, large scale classification must also deal with issues such insufficient training data, which can often be more challenging. One of the main contribution of this thesis is towards the development scalable methods for large scale hierarchical problems.

1.2 Problem Statement

Binary classification is a well researched problem in machine learning. As we delve into classification problems in more complex domains new insights are being gained. The current thesis is an effort towards furthering these insights. The primary focus of this thesis is classification involving multiple tasks having complex interrelationships. The two main focal points are problems in large scale hierarchical classification and multi-task learning. Within the domain of large scale hierarchical classification, the goal is to come up with learning schemes that are scalable and improve the generalization performance by understanding the unique aspects of the problem with respect to utilizing hierarchical relationships, addressing high computational demands, and dealing with categories with insufficient examples. When multiple hierarchical schemes exist on the similar data elements, conceivably, additional knowledge should provide a better understanding of the class structure, which can be exploited by classifiers. The research problem here is — what learning paradigms are best suited to exploit this common knowledge when an explicit mapping between the two hierarchical schemes is not available? Finally, when no relationships are known, we would like to understand how to uncover the task relationships in order to mitigate the adverse effects of joint learning.

1.3 Contributions

In this thesis, I have tried to address some important challenges encountered in large scale hierarchical and multi-task learning domains. As mentioned earlier, large scale classification in hierarchies poses some interesting and challenging problems. One of the main issues is that a majority of classes have insufficient training examples. Regularization strategies that impose similarity between neighboring nodes have proven very successful [11,16]. However, efficient training of these models with large number of tasks is a significant challenge. Chapter 3 presents our work related to distributed regularized training for hierarchical classification and its implementation using map-reduce distributed computation framework.

Even though distributed optimization methods are able to scale well to extremely large scale scenarios, the global optimization of all the model variables in an integrated fashion incurs communication overhead, which can sometimes be a considerable. Additionally, due its recursive nature of regularization the learning algorithm performs several basic iterations of the model training for terminal classes. To address this problem, we re-examined the regularization formulation and reformulated hierarchical classification problem as a cost sensitive classification problem, which is presented in Chapter 4. The cost sensitive classification method provides additional flexibility in modeling the costs and enables us to deal with the problem of imbalance in the training data as well.

Hierarchies are popular schemas for organizing information in many application areas, and often, several hierarchical schema may be available for similar information with some minor differences. Biological ontologies are a prominent example of such phenomena where multiple classification schemes for protein structure classification and protein function classification exist [17–19]. Multiple views provided by different classification schemes might provide diverse sources of information that should in principle help us build better classifiers. In the work presented in Chapter 5, we demonstrate the effectiveness of combining multiple hierarchies through the use of multi-task learning.

In certain cases, the relationships between tasks might be unknown, which poses a challenge in effective transfer of information between tasks. Uncovering relationships between tasks allows us to selectively learn only from the tasks which can assist in improving generalization performance. In Chapter 6, we present a max-margin method that is able to uncover task relationship while simultaneously learning the model weights.

To summarize, the contributions of this thesis are as follows:

- The development of a distributed hierarchical regularization scheme for large scale hierarchical classification.
- Reformulation of hierarchical regularization using cost sensitive loss and application to large scale hierarchical classification.

- Multi-task learning methods for classification involving multiple hierarchical classification systems and application to protein structure classification.
- Large margin method for learning task relationships and efficient optimization procedure using bundle methods.

Several peer reviewed publications were outcome of the work presented in this thesis [20–24].

The work presented in this thesis can be extended in several interesting directions. In Chapter 7 some problems for future work are presented. Chapter 2 summarizes the basic terminology and notations commonly used in this thesis and also provides an overview of foundational research related to our work.

Chapter 2: Background

This chapter introduces the background work related to the problems addressed by this thesis. A basic knowledge of machine learning and classification methods is assumed. Specifically, this chapter gives a broad overview of hierarchical classification and multi-task learning. A considerable research has been performed in these areas, and a comprehensive review of all the methods published in literature would be a massive undertaking, therefore, the discussion is restricted to most representative methods which illuminate upon the common techniques used.

2.1 Terminology and Notations

This section will explain some of the commonly used notations and terminology in this thesis. To cope up with the multitude of use cases, notations are commonly reused, but the context should leave little room for ambiguity. For convenience, Table 2.1 summarizes some of the most frequently used notations.

Typically, lower case letters are used to represent both vectors and scalar values, and upper case letters to denote matrices. j^{th} component of vector a is represented as $a^{(j)}$. $\mathbb{R}^m$ denotes the Eucledian space of dimensionality m. $[a : b]$ denotes the set of positive integers z such that $a \leq z \leq b$.

For general classification problems, the instances are represented using (x_i, y_i), where $x_i \in \mathcal{X}$ represents the set of input features, and $y_i \in \mathcal{Y}$ represents the label for the examples. Feature space $\mathcal{X}$ is generally a subset of $\mathbb{R}^d$, where d denote dimensionality of the feature space. For binary classification $\mathcal{Y} = \{+1, -1\}$. For multi-class and hierarchical classification the label space is represented as set of positive integers $\mathcal{Y} = [1 : T]$, where T is the number of possible labels. For simplify notation multi-class and hierarchical classification , we use

Table 2.1: Notations.

Notation	Meaning
$\mathbb{R}$	set of real numbers
$[a:b]$	set of integers i, such that $a \leq i \leq b$
n	Number of input instances
d	dimensionality of the feature space
T	Number of class labels/tasks
w_t	linear weight vector associated with classifier for class/task t
x_i	input feature vector associated with i^{th} examples
x_{it}	input feature vector associated with i^{th} example of task t
y_i	binary label of example i
y_{it}	binary label of example i with respect to class/tasks t
$\hat{y}_i\,(\hat{y}_{it})$	predicted label of example i (w.r.t. task t)
$\mathcal{L}$	generic loss function
$\mathcal{R}$	generic regularizer
$\gamma\,(a,b)$	length of the undirected path between nodes a and b
$\|a\|_p$	l_p norm of vector a
$\|M\|_{p,q}$	$L_{p,q}$ norm of matrix M

two different representations of class labels $l_i \in [1:T]$ represents the class index of example i, where $\mathcal{T}$ is the set of finite labels. This labels can also be represented using a vector of binary labels $\mathbf{y}_i = [y_{i1}, y_{i2}, \ldots, y_{iT}]$, where $y_{it} = +1$ iff $l_i = t$ and $y_{it} = -1$ otherwise. A collection of training instances is called a dataset, which is denoted by $\mathcal{D} \equiv \{(x_i, y_i)\}_{i=1}^{n}$, where n is the number of examples. In the case of general multi-task learning, where training examples are specific to each individual task, we denote the set of examples for task t using $\{(x_{it}, y_{it})\}_{i=1}^{n_t}$, where n_t denotes the number of examples of task t. In multi-task learning, the number of tasks is denoted by T. f_t is used to denote the learned classifier for class $t \in [1:T]$. For the most part, the classifiers learned in this thesis are linear functions of the form $f_t = sign\left(w_t^T x\right)$, where the weight vector w_t sufficiently describes the classifier for class t. We use W to represent the weight matrix whose v^{th} column is are denoted by w_v , therefore, $W \equiv [w_1, w_2, \ldots, w_T]$.

The set difference of sets A and B is denoted by A/B. To denote the positive component of a scalar value, a we use $|a|_+ \equiv \mathbf{max}(0, a)$. The absolute value of a scalar value v is denoted by $|v|$; for a set or vector M, $|M|$ returns its length. For a vector v, for any $p \geq 0$, the l_p norm is defined as $\|v\|_p = \left(\sum_{i=1}^{d} \left|v^{(i)}\right|^p\right)^{1/p}$. The special case of $p = 2$ is commonly known as the *euclidean norm.* Other Special cases which are commonly used are $p = 1$ and $p = \infty$. Similarly for a matrix M, and for any $p, q > 0$, we define the $L_{p,q}$ norm,$\|M\|_{p,q}$ as the l_q norm of the l_p norms of the columns of M, therefore, $\|M\|_{p,q} = \left\{\sum_j \left(\sum_i |M_{ij}|^p\right)^{q/p}\right\}^{1/q}$. The special cases $L_{2,2}$, known as Frobenius-norm, and $L_{2,1}$ are frequently used in regularized learning literature. $M \succeq 0$ and $M \succ 0$ denote that M is a positive semi-definite matrix or positive definite matrix respectively. $Normal(\mu, \Sigma)$ denotes a multivariate normal distribution with mean μ and co-variance Σ. M^{-1} denote the inverse of matrix M. $tr\,(M)$ denotes the trace of M, which is defined as the sum of diagonal elements.

In the context of hierarchical classification problems, we use the class label, class, and

node interchangeably to denote the classes. Since the hierarchy is defined over the classes, there should be no ambiguity. The root node is represented with the label 0. We use $\mathcal{N}$ to denote the set of all nodes in the hierarchy and $\mathcal{T}$ to denote the set of terminal nodes, which can be assigned as labels to instances, there for hierarchical classification $\mathcal{Y} = \mathcal{T}$. The function $\pi : \mathcal{N}/\{0\} \rightarrow \mathcal{N}/\mathcal{T}$ maps a node to its parent. $\chi(v)$ denotes the set of children of node v. $\bar{\mathcal{A}}(v)$ denotes the set of ancestors of node v, not including the root node. $\mathcal{D}^{+}(v)$ denotes the union of the set of descendants and the node v itself. $\mathcal{S}(v)$ denotes the set of siblings of node v.

Additional notations not mentioned here will be described, as required, in specific chapters.

2.2 Multi-task Learning

In the standard settings of supervised machine learning, the objective is to learn a predictive function using example data. However, in real world settings, we often encounter situations with several related learning tasks. For example, in personalized email spam classification, the classification of spam for each user can be considered a separate task; in automated driving, steering and acceleration can be considered related tasks. Intuitively, it would seem that learning these related classification or regression tasks jointly should help us uncover common knowledge and improve generalization performance. In fact, this intuition is supported by empirical evidence provided by recent developments in transfer learning [25] and multi-task learning [9] [6] [7].

Multi-task learning (MTL) is a paradigm for learning several related tasks jointly. The generalization performance of the learned tasks is improved by utilizing inductive transfer across tasks. MTL achieves this by leveraging the training signal in related tasks [6, 9], and it has been empirically and theoretically [7] [26] shown to improve the generalization performance, especially when the training data is scarce. Some of the earliest models of multi-task learning were developed using multi-layer back-propagation neural networks [9].

Neural networks can be extended from single task to multiple tasks trivially with additional outputs for each task. Several tasks share the same input layer and one or more intermediate layers in this setting. By training multiple tasks simultaneously the back-propagation net prefers the inductive bias that helps multiple tasks [9].

Recently, there has been a significant progress in research in multi-task learning using Bayesian and regularized risk minimization framework (see [7–10, 13, 26–33] and the references therein). Various MTL models differ in the kinds of assumptions they make about the relatedness of the tasks and how these assumptions are incorporated into the learning algorithm. For example, in email spam classification it would be reasonable to assume that spam for multiple users contain words promoting certain products or sales gimmicks. However, there might also be user specific variations, for example, if a certain user has deliberately subscribed to receive information related to a specific product. Therefore, designing a common spam filter for all users might be effective, but tuning the spam filter specific to an individual might perform better. In this case we could make the assumption that all the spam classification tasks share a similar set of parameters with some task specific variations [10]. Different from this, some MTL formulations try to extract a good representation of the input features or a subset of features that are informative for all the tasks [8, 34]. Consider the task of automated driving which involves controlling steering, acceleration and braking, and the input is a continuous stream of images from a camera or some other sensory inputs. Here although the tasks are inherently different, but depend on the same inputs. Therefore, a good representation of the feature space should be able to aid all the tasks[9]. Finally, in some learning algorithms such as k-nearest neighbors which primarily use a measure of similarity or distance from the neighboring instances for classification, we might leverage the inductive bias from several tasks in order to learn a good distance metric for all the predictive tasks [7].

Although, the general multi-task learning concepts have been incorporated into various kinds of learning algorithms, this thesis will primarily focus on the formulations based on

regularized risk minimization. The MTL methods using regularized risk minimization predominantly incorporate regularizers that acts on several tasks simultaneously, as I discuss below.

2.2.1 Regularized Multi-task learning

Given a training set with n input-output example pairs, $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, the objective of standard machine learning models is to learn a mapping function $f : \mathcal{X} \to \mathcal{Y}$ between the input domain $\mathcal{X}$ and the output domain $\mathcal{Y}$ which minimizes the loss on data not encountered in training. The training examples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ are drawn from an unknown distribution. The regularized risk minimization framework achieves this goal by modeling an objective function as a trade-off between loss function, which minimizes the error, and regularization penalty, which controls the model complexity to discourage over-fitting.

$$\min_{w} \sum_{i=1}^{n} \underbrace{\mathcal{L}\left(w, x_i, y_i\right)}_{loss} + \lambda \quad \underbrace{\mathcal{R}\left(w\right)}_{regularization} \tag{2.1}$$

This can be generically represented as (2.1) where w is the set of model parameters to be learned. This principle can be extended to MTL, where we have T tasks with training data for each of the $t = 1 \ldots T$ tasks given by $\{(x_{it}, y_{it})\}_{i=1}^{n_t}$. The combined learning objective can be written as,

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \underbrace{\mathcal{L}\left(w_t, x_{it}, y_{it}\right)}_{loss} + \lambda \quad \underbrace{\mathcal{R}\left(W\right)}_{regularization} \tag{2.2}$$

Various multi-task learning methods take this general approach to build combined models for many related tasks, typically, by enforcing MTL assumptions through regularization term.

2.2.2 Group Regularization

One of the first MTL methods based on regularized risk minimization framework was proposed by Evegeniou and Pontil [10]. The key assumption of their model is that all the tasks are closely related and their model weights are similar. This assumption is incorporated into the method by a regularization term that penalizes the deviation of the model weights for each task from the mean of all tasks as shown in (2.3).

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \left| y_{it} w_t^T x_{it} - 1 \right|_+ + \rho_1 \sum_{t=1}^{T} \|w_t\|_2^2 + \rho_2 \sum_{t=1}^{T} \left\| w_t - \frac{1}{T} \sum_{s=1}^{T} w_s \right\|_2^2 \tag{2.3}$$

However, this formulation assumes that all the tasks are equally related to each other. In some case, we might have the knowledge of task relationships as a task network graph. Whenever such relationships between tasks are available, it is beneficial to take them into account and enforce similarity only between task pairs that are related. The MTL formulations proposed in [13, 35] incorporate externally provided task relationship into the regularization term and penalize the deviations of only related tasks. Specifically, the *Graph Regularization* formulation proposed by Evegeniou *et al.* [13] considers the regularizer of the form given in (2.4)

$$\mathcal{R}(W) = \frac{1}{2} \sum_{(l,q) \in [1:T] \times [1:T]} \|w_l - w_q\|^2 A_{lq} \tag{2.4}$$

where A_{lq} denotes the weight of the edge joining task indexed by l and q.

2.2.3 Feature Learning and Feature Selection

Sparse learning schemes, such as lasso and elastic net [36] [37], which are based on l_1 regularization of the model weights have been common components of feature learning methods in machine learning literature. It has been noted in literature that l_1 norm penalty promotes

sparsity by forcing several components to zeros [38]. Several MTL methods inspired by such schemes have tried to incorporate this characteristic into matrix norms such as $L_{2,1}$ norms.

Argyriou *et al.* [27] proposed a sparse feature learning scheme represented in (2.5).

$$\min_{U,A} \sum_{t=1}^{T} \sum_{i=1}^{n_t} L\left(y_{it}, a_t^T U^T x_{it}\right) + \rho \left\|A\right\|_{2,1}^2 \tag{2.5}$$

In this formulation, U acts as a linear projection matrix which projects the input features into the space defined by U, and A is the matrix of learned weights in the projected dimensions, whose columns are represented by a_t for $t \in [1:T]$. The $L_{2,1}$ norm regularizer promotes sparsity by selecting the same features across all the tasks. A similar approach using $L_{2,1}$ regularization was also used by Obozinski *et al.* [39], without the projection onto a linear sub-space. An efficient optimization scheme to solve the $L_{2,1}$ regularization formulation using accelerated gradient descent method was proposed by Liu *et al.* [28].

In the previous methods, one underlying assumption is that all tasks share a common subset of informative features. This may be a limitation in certain settings. This was addressed by the tree guided group lasso method proposed by Kim *et al.* [40] where external task relationships guide the feature selection by enforcing a group-wise sparsity constraints.

In contrast to the previous regularization formulations, the method proposed by Jebara [34] uses a maximum entropy discrimination based formulation by defining binary feature switches as part of the trained model.

2.2.4 Relevant Tasks and Subgroups Learning

Several MTL methods make the assumptions that the tasks are symmetrically related to each other. However, the assumption of symmetric relationships between all tasks made by this formulation is not suitable for many real world problems where the degree of relatedness between different tasks can vary. When the task relationships are externally provided, the graph regularization method [13] is able to leverage them, but in certain cases even the task

relationships might be unknown and might have to be inferred from the data. A major challenge in MTL is to avoid the sharing of information between tasks which are unrelated. If unrelated tasks are allowed to influence each other then using MTL for learning can deteriorate performance instead of improving it, a phenomenon which is termed as *negative transfer.*

One way to avoid negative transfer is by clustering related tasks. Clustered MTL formulations [41, 42] make the assumption that tasks are grouped into clusters such that tasks within each cluster share greater similarity with other tasks in the same cluster. The clustering of the tasks is not known *a priori* and needs to be inferred by the learning algorithm. Therefore, these formulations simultaneously learn the task clustering and infer the model parameter. One of the drawbacks of this approach is that in case of hard clustering the tasks are limited to a particular group and might not be able to contribute towards the learning of tasks from the other group. Task clustering tries to avoid negative transfer by avoiding learning dissimilar tasks together, but it might also sometimes limit the positive inductive transfer by imposing hard grouping. Although clustered multi-task learning can extract related groups of tasks to some extent, one of their shortcomings is that the number of task clusters is not known beforehand and hence, needs to be determined through parameter tuning.

In contrast to task clustering formulations, a low-dimensional sub-space assumption can be imposed on the task weight matrix such that the regularizer tries to minimize the rank of the weight matrix W. Although directly minimizing the rank of a matrix is a difficult problem to solve, the trace norm minimization has been shown to achieve low rank solution [43, 44]. An interesting approach taken by Kumar *et al.* assumes that the task weight parameters are sparse combinations of some underlying weight vectors as shown in the objective function (2.6).

$$\sum_{t=1}^{T} \sum_{i=1}^{n_t} L\left(y_{it}, s_t^T L^T x_{it}\right) + \rho_1 \left\|S\right\|_1 + \rho_2 \left\|L\right\|_2^2 \tag{2.6}$$

here L is matrix of size $d \times k$ where $k < T$ which contains the weights of the latent basis tasks. The weight vector $w_t = Ls_t$ (s_t is vector of size k) is a sparse combination of the latent task weights, L and a sparse vector s_t ($S = [s_1, \ldots, s_T]$). In this formulation, tasks from different groups are allowed to share the basis task parameters. However, as with the task clustering formulations where the number of clusters must be tuned, here we must tune the number of latent tasks.

Another set of approaches, mostly based on Gaussian Process models, learn the task co-variance structure [30, 31] and are able to take advantage of both positive and negative correlations between the tasks.

2.3 Hierarchical Classification

Categorizing entities according to a hierarchy of general to specific classes is a common practice in many disciplines. It can be seen as an important aspect of various fields such as bioinformatics, music genre classification, image classification and more importantly document classification [2]. A large number of databases organize information in a hierarchical format. For e.g., popular protein function database, Gene Ontology (GO) [18], catalogs proteins in a functional hierarchy. Web document classification databases, such as Wikipedia [1], DMOZ Open Directory Project [2], International Patent Classification system [3], and Yahoo! web directory [4], define hierarchical structure over the topics which are used to label the web pages. In most cases, curation in these databases remains a manual process. However, with accelerated growth in the size of these databases, automated methods for organizing and labeling new instances in hierarchical classification systems have become essential.

Even though some progress has been made in the recent years on the task of hierarchical classification [2], a large amount of research deals with either flat classification, where the

[1] http://www.wikipedia.org
[2] http://www.dmoz.org
[3] http://www.wipo.int/classifications/ipc/en/
[4] https://business.yahoo.com

hierarchical structure is disregarded entirely, or performs top-down classification using existing binary or multi-class classifiers. Performance of these straightforward methods tends to be unsatisfactory [1] compared to methods that utilize the hierarchy in a more principled manner. In recent years, many methods have been proposed to tackle the global hierarchical classification problem as a whole. We provide a discussion of the different strategies in hierarchical classification and a review of some global classification methods in section 2.3.2.

2.3.1 Definition and Characteristics

In this section, we provide a brief background and a formal description of the machine learning problem addressed by hierarchical classification.

In standard supervised machine learning problems, we are given a dataset $\mathcal{D}$ of n input instances $\mathcal{D} \equiv \{(x_i, y_i)\}_{i=1}^{n}$, where $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ and the goal is to learn a predictive function $f : \mathcal{X} \to \mathcal{Y}$ that accurately maps the inputs x to the outputs y and generalizes well to data not seen during training.

Different classes of supervised machine learning problems can be distinguished by different choices output spaces. In *regression* problems the continuous output space $\mathcal{Y} \subseteq \mathbb{R}$. Whereas in *classification* problems the output labels are defined by a finite set of distinct labels ; in *binary* classification problems $\mathcal{Y} = \{0, 1\}$; *multi-class* classification extends the binary classification from two to m, a fine number, of discrete labels, which are conveniently represented by set of integers $[1 : m]$; in *multi-label* classification problems, although the labels come from a finite set of classes represented by $[1 : m]$, each instance can be associated with multiple classes, therefore the instance labels can be considered as elements of the power set of classes, $2^{[1:m]}$. *Hierarchical* classification problems, can be seen as a special case of multi-label classification where we additionally define a hierarchical structure $\mathcal{H}$ over the set of output labels, and the label assignment must be consistent with the hierarchical structure.

Since hierarchy is central to the hierarchical classification problem, we define the concept of a hierarchy as well as its generalization, graph, as follows.

Graph $\mathcal{G}$ A graph is defined as a tuple $\mathcal{G} \equiv (V, E)$ where V is a set of objects (referred to as vertices or nodes of the graph) and $E \subseteq V \times V$ is a set of ordered or unordered pairs of objects referred to as edges or links. If the ordering is unimportant for $e \in E$, then the graph is an undirected graph otherwise, it is referred to as a directed graph. A graph defines the concept of adjacency such that, two vertices u, v which are joined by an edge are called adjacent, i.e. $(u, v) \in E \vee (v, u) \in E$.

Hierarchical Structure or Hierarchy $\mathcal{H}$ In its most general form, a hierarchy $\mathcal{H}$ consists of a set of objects V and a partial order $\preceq$ over the pairs of elements of V. The partial order arises from the parent-child relationship between the elements of V. A hierarchy can also be considered a special case of a directed acyclic graph where the directed edges define the parent-child relationships between vertices/nodes. A hierarchy where each node has a single parent is known as a Tree.

Several variants of the basic hierarchical classification problem exists, which pose different challenges from machine learning perspective. Different aspects of hierarchical classification are described below and summarized in Table 2.2.

Hierarchy Structure The hierarchy over the class labels can be either a Tree or a directed acyclic graph (DAG). It is significantly more challenging to deal with the problems where the hierarchies have a DAG structure. In some rare cases, the structure defined over the labels can be also a general graph, although, the label structure can no longer be described as a hierarchy in such cases.

Mandatory Leaf Node Prediction Generally, in most problems the most specific labels assigned to instances belong to the set of leaf nodes. In these cases, the internal nodes define a *virtual hierarchy* with no concrete instance assignment other than those assigned to their descendants. In some problems, the internal nodes can also be the terminal labels of instances. Therefore, in the latter case the algorithm has to decide whether an internal node is the terminal label or to proceed to its descendant, making it more difficult than the former case.

Table 2.2: Categorization of hierarchical classification problems.

Criterion	Description
Hierarchy Structure	Is the hierarchical structure over the labels a Tree, Directed Acyclic Graph (DAG) or a general Graph?
Mandatory Leaf Node	Is the most specific labels of an example necessarily a leaf node?
Multi-label	Can instances have multiple labels with no ancestor-descendant relationships between the labels?

Multi-label Hierarchical classification problems are inherently multi-label prediction problems because an example assigned to a particular category inherits the labels of all its parents. But we call a hierarchical classification problem multi-label if an example can be assigned two labels u, v such that u is neither an ancestor nor a descendant of v.

Various methods for Hierarchical classification have been proposed in the literature . They can be broadly categorized according to aspects of the problems they can address and how they utilize the hierarchical structure. Typically, most methods only address hierarchies involving trees with mandatory leaf node prediction, hierarchical single label prediction problems.

Since, hierarchical structure over labels is the principal aspect of these methods, the most important categorization deals with utilization of the hierarchical structure by the methods. The simplest approach, known as *flat classification*, disregards the hierarchical relationships between categories and trains independent binary or multi-class classifiers to distinguish the leaf categories from other leaf categories in the hierarchy. The *local classifier* approaches deals with the hierarchy by splitting the overall problem into smaller problems, whereas the *global classifiers* incorporate the hierarchy from a global perspective. We provide a detailed overview of these categories below. There has been extensive research in the past few decades in hierarchical classification and numerous methods have been proposed to address this problem. The research in this area is based on several fundamental machine learning techniques such as rule mining [45], decision trees [46], neural networks [47], probabilistic methods [48,49], regularized risk minimization [1,50] and so on. A comprehensive discussion

of all the research in this area would be a massive undertaking. Therefore, we limit our discussion primarily to the methods set in regularized risk minimization framework and to a limited extent we also discuss methods using probabilistic framework. In our discussion, depending on the context we use the terms class, label and node interchangeably and refer to both the nodes in the hierarchy as well as the class labels. Since the nodes in the hierarchy are indeed the class labels it is obvious that we are referring to the same entities.

2.3.2 Flat Classification

Flat classification disregards the hierarchical structure of the problem and treats it as a multi-class or multi-label problem. Therefore, classifiers are trained to distinguish each of the terminal nodes in the hierarchy with the others nodes. Traditional binary classifiers, such as SVMs [51] can be used to in one-vs-rest (also known as one-vs-all) setting, where a classifier f_t is associated with each terminal labels $t \in \mathcal{T}$, which returns a score indicating the confidence that an instance belongs to class t. The class labels $\hat{l} \in \mathcal{T}$ for test instance x is predicted according to (2.7).

$$\hat{l} = \mathbf{argmax}_{t \in \mathcal{T}} f_t(x) \tag{2.7}$$

Methods which perform direct multi-class classification instead of utilizing binary methods in one-vs-rest settings such as Crammer-Singer [52] formulation of multi-class SVM (CS-SVM) can also be utilized. The CS-SVM, formulation learns a max-margin linear function by minimizing the convex objective function given in (2.8). The margin maximization requirement is enforced by the first constraint which tries to enforce a margin between the

hyperplane for the correct class and the incorrect classes.

$$\begin{aligned} &\underset{\{w_t\}_{t\in\mathcal{T}},\{\xi_i\}_{i=1}^n}{\textbf{minimize}}: \ \frac{1}{2}\sum_{t\in\mathcal{T}}\|w_t\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ &\textbf{subject to}: w_{l_i}^T x_i - w_t^T x_i \geq 1-\xi_i \qquad \forall t \in \mathcal{T} - \{l_i\}, \forall i \in [1:n] \\ &\qquad\qquad \xi_i \geq 0, \quad \forall i \in [1:n] \end{aligned} \tag{2.8}$$

As we will see later, this formulation is the basis for other methods, such as Structured output SVMs [53] which are applicable to complex output spaces but have also been used for the special case of hierarchical classification.

2.3.3 Top-down Local Classification

Flat classifiers train one-vs-rest classifiers to distinguish each class from the rest. Consequently, each classifier is trained on the entire training dataset. In contrast, the local classification methods decompose the global hierarchical problem into several local binary or multi-class problems based on the hierarchy [2, 54]. These can be further subdivided into three types of approaches.

Local classifier per Node (LCN) approaches train binary classifiers f_v for every non- root node $v \in \mathcal{N}/\{0\}$ to predict the membership of a test example to that node. For training f_v, we choose the examples belonging to all the descendants of the node's parent parent $\pi(v)$. The examples belonging to all of v's descendants $\mathcal{D}^+(v)$ are labeled as positive and the remaining examples as negative. Hence, essentially the LCN classifiers learn to distinguish a node from all its siblings. In the prediction phase, the collection of classifiers $\{f_v\}_{v\in\mathcal{N}/\{0\}}$ is used in a top-down classification approach given by (2.9).

$$\hat{l} = \left\{ \begin{array}{l} \textbf{initialize} \quad p := 0 \\ \\ \textbf{while } \chi(p) \text{ is not empty} \\ \\ \quad p := \textbf{argmax}_{q \in \chi(p)} \ f_q(x) \\ \\ \textbf{return } p \end{array} \right\} \tag{2.9}$$

The Local Classifier per Parent Node (LCPN) approach trains a multi-class classifier for each non-leaf node $v \in \mathcal{N}$ for predicting the appropriate child node. Similar to the LCN approach, the purpose of the classifiers is also to differentiate between the children of a node. However, instead of a binary classifier at the non-root nodes, a multi-class classifier is trained at each non-leaf nodes. If a binary classifier is utilized in one-vs-rest multi-class classifier settings in LCPN, then it would be equivalent to LCN. The training examples for multi-class classifier f_v are those belonging to $\mathcal{D}^+(v)$, and each of its children $c \in \chi(v)$ are the different classes. The prediction algorithm is similar to that illustrated in (2.9).

Finally, the Local classifier per level (LCL), which is the least popular among the local classification approaches, trains a single multi-class classifier for each level of the hierarchy, to distinguish between all the nodes at that level.

Since the local classifiers decompose the problem, existing methods for binary and multi-class classification can be used. Also, due to the local nature of decomposition, each classifier can be independently trained. Therefore, these methods are trivially parallelizable. Since each classifier is trained on a smaller dataset, the training is more efficient than flat classifiers. However, there are several drawbacks of this top-down approach. In the prediction phase, the classification is performed top-down, therefore any prediction errors performed in the top levels can not be corrected. Hence, for the local methods to perform well, all the classifiers should be very accurate, especially at the higher levels, otherwise the prediction errors get compounded. In LCL approach, the classifiers at different levels can output inconsistent

predictions *i.e.* the labels predicted by classifiers at different might be unrelated according to the hierarchy. As we traverse down the hierarchy, the most specific classes at the bottom have fewer examples. Since the LCP and LCPN strategies partition data into progressively smaller sets down the hierarchy, some nodes might not have sufficient data to train accurate classifiers.

2.3.4 Problem Transformation

As we saw in the previous sections, local methods transform the original problem by decomposing it into smaller problems. The methods in Problem Transformation category, instead of partitioning the instance space according to the hierarchy, transform the output space into independent problems or perform consistency correction on the independently predicted outputs. Therefore, these methods retain the modularity of flat and local classification approaches. They can also take advantage any existing binary classification or regression methods. We describe two methods below which illustrate these approaches.

The method proposed by Bi and Kwok [55], which inspired by the label space transformation methods in multi-label literature [56], [57], applicable to both Tree and DAG hierarchies. The central idea of this method consists of transforming the labels $\mathbf{y}_i \in \{0,1\}^{|\mathcal{N}|}$ from the original output space to a lower dimensional space $z_i \subset \mathbb{R}^p$, creating new set of instances $\{(x_i, z_i)\}_{i=1}^n$, where $p < m$. In the modified output space, the assumption is that the components $z_i^{(j)}$ are de-correlated. Therefore, p independent regression model can be trained for each of the p datasets $\{(z_i^{(j)}, x_i)\}_{i=1}^n$, where $j \in [1:p]$. Any regression method can be used in the learning step. In prediction step, for a test example x, each model predicts $\left\{\hat{z}^{(j)}\right\}_{j=1}^p$ which are projected back onto the original output space to get the label $\hat{\mathbf{y}}$. This is the basic idea as used in KDE for multi-label classification. However, the labels predicted by this procedure can be inconsistent with the label hierarchy. Therefore, Bi and Kwok [55] propose a consistency correction procedure to enforce hierarchical consistency of the labels according to both Tree and DAG hierarchies.

Barutcuoglu *et al.* [58] proposed a method to address the problem of protein function classification on Gene Ontology (GO) [18]. GO organizes the functional classes into a DAG structured hierarchical ontology. The model consists of two steps. In the first step, a binary classifier is trained for each functional class. The outputs of these models can be either binary prediction or confidence scores. In the second step these inconsistent predictions are processed by a second level Bayesian network model to produce the final output labels.

2.3.5 Global Classification

The methods categorized as *Global Classification* methods treat the global hierarchical classification problem as a whole. Typically, a specialized solutions for the hierarchical problem is proposed which lacks the modularity of the approaches discussed previously. These methods train a single global model taking into account the interrelated classes in the hierarchy. However, the prediction algorithm in many cases can be similar to *flat* or *local* methods. In the following section we look at some models proposed for the purpose of hierarchical classification and discuss the various ways in which the hierarchical assumptions are imposed by the classifiers.

Typically in most classification problems, any incorrect prediction is treated as a misclassification, irrespective of severity of the error. In real world hierarchical classification, we might be interested in how far-off the predicted label is from the true label according to the hierarchy, because an error in the most specific category might be considered less severe.

The model proposed by Dekel *et al.* [59] is similar to the max-margin formulation for multi-class problem. This method tries to maximize the margin between the correct class and the remaining classes. However, the margin is scaled by a factor of $\sqrt{\gamma\left(l_i, r\right)}$, as shown in (2.10). Here l_i is the true class and $r \in \mathcal{N}/\{l_i\}$. Therefore, the classes which are closer to the true class will be required to have a smaller unscaled margin.

$$\left\{ \frac{w_{l_i}^T x_i - w_r^T x_i}{\sqrt{\gamma\left(l_i, r\right)}} \geq 1 \right\}, \forall \left(x_i, l_i\right) \in \mathcal{D},\ \forall r \neq l_i \tag{2.10}$$

Orthogonal Transfer model proposed by Zhou *et al.* [50] is similar in sprit to the top-down local classifier, but encourages orthogonality between the model vectors learned at a node to its ancestors at higher levels. It is motivated by the fact that classification at different levels of the hierarchy may rely on different features or feature combinations. Therefore, the classification hyperplanes learned at each level should be as different as possible from their ancestors. In other words, the learned hyperplanes should be orthogonal or nearly orthogonal to those of the ancestors. The model tries to minimize the objective function in (2.11) to incorporate this assumption.

$$\begin{aligned}
&\underset{\{w_t\}_{t\in\mathcal{N}}}{\textbf{minimize}} : \frac{1}{2}\sum_{(p,q)\in\mathcal{N}\times\mathcal{N}} K_{pq}\left|w_p^T w_q\right| + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\
&\textbf{subject to} : \\
&\qquad w_p^T x_i - w_q^T x_i \geq 1-\xi_i, \qquad \forall p\in\mathcal{A}^+(l_i), \forall q\in\mathcal{S}(p), \forall i\in[1:n] \\
&\qquad \xi_i \geq 0, \quad \forall i\in[1:n]
\end{aligned} \tag{2.11}$$

Contingent up certain conditions on the matrix K, the problem can be shown to be convex [50]. $K_{pq}=0$ wherever p and q are not related according to the hierarchy, and $K_{pq}>0$, when they p,q are related. Hence, $\left|w_p^T w_q\right|$ term disappears when p,q are unrelated; when $p=q$, it reduces to $\|w_p\|_2^2$, which regularizes the weights towards $\mathbf{0}$; when p,q have an ancestor-descendant relationship, the term $\left|w_p^T w_q\right|$ enforces orthogonality between the vectors w_p and w_q.

Hierarchical classification can also be viewed a special case of structured output prediction [53, 60] because the collection of interrelated hierarchical labels is complex output. Cai and Hoffman [61] applied a structured output model similar to Structured Output SVMs

[53], for hierarchical classification. This model is presented in (2.12).

$$\begin{aligned}
&\underset{\{u_t\}_{t\in\mathcal{N}}}{\textbf{minimize}} : \frac{1}{2} \underbrace{\sum_{t\in\mathcal{N}} \|u_t\|_2^2}_{regularization} + \frac{C}{n} \underbrace{\sum_{i=1}^{n} \xi_i}_{loss} \\
&\textbf{subject to} : \\
&\underbrace{\sum_{t\in\mathcal{A}^+(l_i)} u_t^T x_i - \sum_{q\in\mathcal{A}^+(r)} u_q^T x_i}_{margin} \geq 1 - \frac{\xi_i}{\Delta(l_i, r)}, \qquad \forall r \in \mathcal{N} - \{l_i\} \\
&\xi_i \geq 0, \quad \forall i \in [1:n]
\end{aligned} \tag{2.12}$$

Comparing this model with that of the standard SVM [51] formulation, the regularization term is the sum of l_2 norm of the weight vectors of all the classes and the loss is the sum of the slack variables. The hierarchical knowledge is incorporated through the first constraint equation where a) the margin of x_i , with respect to the true class label l_i and some other label $r \neq l_i$, considers the ancestors of these classes, and b) the slack is scaled by $\Delta(l_i, r)$ where $\Delta : \mathcal{N} \times \mathcal{N} \rightarrow \mathbb{R}$ encodes the loss of predicting r when true label is l_i. Therefore, this model can incorporate complex losses Δ. However, the model size becomes exponentially large with respect to the number of categories and is infeasible for large scale problems.

The motivation for the top-down (LCPN) based hierarchical feature selection scheme proposed by Koller and Shahami [48] is that the discriminant function at higher levels of the hierarchy are considerably different from the discriminant functions at lower levels and the relevant features at each classifier node might vary considerably. This assumption is incorporated by performing feature selection at each local node using an information theoretic criterion. The features which are most correlated with the class under consideration are selected.

For each feature X_i, the algorithm determines $E[\delta_i]$, the expected value of δ_i, where

$$\delta_i \equiv P(X_i) D_{KL}(P(Y \mid X) \| P(Y \mid X_{-i})) \tag{2.13}$$

where $D_{KL}(p\|q) \equiv \int_z p(z) \log p(z)/q(z)\, dz$ is generally known as the Kullback Leibler divergence, but in this paper it is called cross-entropy; X is the set of all features, X_i is the i^{th} feature and $X_{-i} = X - \{X_i\}$. $E[\delta_i]$ specifies the importance of the feature i w.r.t. the class. Therefore, eliminating the feature that minimizes this value disrupts the conditional distribution $P(Y \mid X)$ least. To compute $P(Y \mid X)$ the algorithm uses Naive Bayes model. This procedure is iteratively applied to reduce the feature space until the required number of features are selected.

McCallum *et al.* [16] proposed a hierarchical shrinkage model which is based on the observation that most specific classes (terminal nodes), often suffer from insufficient number of training examples while the more general classes (top levels of the hierarchy) contain more examples and hence the classifiers learned for them tend to be more robust. Therefore, this method exploits the hierarchy by shrinking the parameter estimates in data sparse children towards the data rich ancestors. Their model was proposed for the problem of hierarchical document classification using Naive Bayes as base classifiers. Initially, considering a flat classification model, the problem is formulated a probabilistic mixture model where the classes correspond to mixture components.

The Bayesian model proposed by Gopal *et al.* [12] is also based on the assumption that the classes related according to the hierarchy have similar model parameters. The generative form of the of the probabilistic model proposed in this papers is given in (2.14).

$$\begin{aligned} w_t \mid m_t, \Sigma_t &\sim Normal(m_t, \Sigma_t), \qquad \forall t \\ p_l(x) &= \exp\left(w_l^T x\right) / \sum_{q \in \mathcal{T}} \exp\left(w_q^T x\right) \\ l \mid x &\sim Categorical\left(p_1(x), \ldots, p_{|T|}(x)\right), \qquad \forall (x, l) \in \mathcal{D} \end{aligned} \tag{2.14}$$

The weight parameters w_t for class t are generated by Gaussian distribution whose mean m_t is set to that of the parent node, in other words $m_t := w_{\pi(t)}$. Therefore, the model weights of the children are effectively shrunk towards the parents. Since the root node has no parent, the mean is user defined, usually set to $\mathbf{0}$.

However, there is now considerable freedom in defining additional assumptions on the models through the co-variance matrix Σ_t. In this work, three different ways of modeling the co-variances of related models were evaluated. In two models, siblings share the same co-variance matrix through parameters generated from common parent. In the third model, the co-variance matrix is allowed to be independent for the siblings. The major contribution here, however, is developing a scalable inference procedure.

2.4 Classifier Performance Evaluation

Accuracy, which is defined as the fraction of correct predictions, is the most commonly used evaluation measures in classification. If the number of positive and negative examples are highly unbalanced, then a classifier can achieve high accuracy by predicting all the examples into the majority class. Alternate performance measures, such as *Area Under Receiver Operating Characteristics (AUC-ROC)* curve [62], sometimes abbreviated as Area Under Curve (AUC), that take class imbalance into consideration are better suited for such cases. The curve is created by plotting the true positive rate against the false positive rate at various thresholds. Since end users can choose a threshold value to balance the tradeoff between true positive and false positive, AUC metric provides the aggregate tradeoff at various thresholds.

Other classification metrics such as Precision (P), Recall (R), and their harmonic mean, F1-score, have their origins in information retrieval. Their definitions are provided below.

$$P = \frac{\#\text{ true positive}}{\#\text{ predicted positive}} \tag{2.15}$$

$$R = \frac{\#\text{ true positive}}{\#\text{ actual positives}} \tag{2.16}$$

$$F_1 = \frac{2PR}{P+R} \tag{2.17}$$

Set based measures such as *Micro-*F_1 and *Macro-*F_1 are commonly used to evaluate multi-class and multi-label classifiers. Many authors have also employed these measures for evaluating performances on hierarchical classification problems.

$$\text{Micro-F}_1 = \frac{2PR}{P+R} \tag{2.18a}$$

$$\text{Macro-F}_1 = \frac{1}{m}\sum_{t=1}^{m}\frac{2P_tR_t}{P_t+R_t} \tag{2.18b}$$

where m is the number of classes, P_t and R_t are the precision and recall values for class $t \in [1:m]$. P and R are the overall precision and recall values for the all the classes taken together. Micro-F_1 gives equal weight to all the examples therefore it favors the classes with more number of examples. In the case of single label classification, Micro-F_1 is equivalent to accuracy. Macro-F_1 gives equal weight to all the classes. Hence, the performance on the smaller categories is give equal importance.

In the context of hierarchical classification, set based measures do not consider the distance of misclassification with respect to the true label of the example, but in general, it is reasonable to assume in most cases that predictions that are closer to the actual class are less severe than predictions which are far apart in the hierarchy. Hierarchical measures, defined in (2.19) , take the distances between the actual and predicted class into consideration.

Hierarchy based measures include Hierarchical Precision (hP), Hierarchical Recall (hR), and their harmonic mean, Hierarchical F_1 (hF_1) and Tree-induced Error (TE) [59, 62].

$$TE = \frac{1}{n}\sum_{i=1}^{n}\gamma\left(\hat{l}_i, l_i\right) \tag{2.19a}$$

$$hP = \frac{\sum_{i=1}^{n}\left|\bar{\mathcal{A}}\left(\hat{l}_i\right)\cap\bar{\mathcal{A}}\left(l_i\right)\right|}{\sum_{i=1}^{n}\left|\bar{\mathcal{A}}\left(\hat{l}_i\right)\right|} \tag{2.19b}$$

$$hR = \frac{\sum_{i=1}^{n}\left|\bar{\mathcal{A}}\left(\hat{l}_i\right)\cap\bar{\mathcal{A}}\left(l_i\right)\right|}{\sum_{i=1}^{n}\left|\bar{\mathcal{A}}\left(l_i\right)\right|} \tag{2.19c}$$

$$hF_1 = \frac{2 * hP * hR}{(hP + hR)} \tag{2.19d}$$

where, $\hat{l}_i$ is the predicted label and l_i is the true label of example i. $\gamma(a, b)$ gives the length of the undirected graphical path between categories a and b. $\bar{\mathcal{A}}(v)$ is the sets of ancestors of label v which includes the label itself, but does not include the root node.

For multi-label classification, where l_i as well as $\hat{l}_i$ are sets of micro-labels, we redefine graph distance and ancestors as: $\gamma_{ml}(l_i, \hat{l}_i) = \left|\hat{l}_i\right|^{-1}\sum_{a\in\hat{l}_i}\min_{b\in l_i}\gamma(a, b)$ and $\mathcal{A}_{ml}(l) = \cup_{a\in l}\mathcal{A}(l)$.

Chapter 3: Distributed Regularized Learning for Large Scale Hierarchical Classification

3.1 Introduction

Much progress has been made in recent years on the task of hierarchical classification [2, 46, 58, 59, 61, 63–65]. Yet, majority of the work has dealt with only small scale datasets. In recent years, with the upsurge in interest in *big data*, large scale classification problem have attracted considerable attention. Although some hierarchical classification methods which train only local classifiers or trivially use binary or multi-class classification can scale to large classification problems, the performance of these straightforward methods tends to be unsatisfactory [1], because they are unable to utilize the hierarchical information. Furthermore, large scale settings present additional challenges such as huge skew in the number of examples of positive and negative classes and the presence of extremely rare categories. Rare categories have very few training examples, and therefore, the trained models tend to not perform well on these classes. Under these circumstances the overall performance of the classifier tends to favor larger classes.

Owing to only a recent interest in the topic of large scale classification, few methods have been proposed in literature that are scalable. In this work, we build upon a hierarchical classification method [11] which biases the classification decision boundary of a category towards the aggregate decision boundary of neighboring categories. This method extends the flat classification scheme, where each of the terminal classes are modeled with a linear binary classifier, since the top-down local classification method tends to perform poorly due to error propagation from mis-classifications at higher level nodes. The hope in biasing the linear classifier towards that on the neighbors is to mitigate the effect of high variance due to the small number of examples in rare categories. In this work, we propose an approximate

block coordinate descent scheme, which improves the efficiency while achieving the same level of classification effectiveness as the exact scheme. Further, we provide a distributed implementation of the method using map-reduce distributed computation framework on Hadoop and compare the approximate and exact schemes for training efficiency. We further perform a detailed analysis on large hierarchical text datasets to understand different aspects of the method which contribute towards its success.

Notations $\mathcal{N}$ is used to denote the set of all hierarchical nodes. $\mathcal{T}$ denotes the set of terminal nodes which can be assigned as class labels to instances. The class membership of instances is represented using binary labels $y_{it} \in \{-1, +1\}$ where $y_{it} = +1$ iff the instance belongs to class t and $y_{it} = -1$ otherwise. $\chi(t)$ denotes the set of children of node t, and $\pi(t)$ denotes the parent node of t. Where the class label is implicitly understood to be t, we drop the subscript denoting the class and denote the class labels as y_i (instead of y_{it}) and the model weight vector as w (instead of w_t).

3.2 Large Scale Hierarchical Classification

With the emergence of large scale hierarchical databases, the problem of hierarchical classification especially in the text domain has exploded in size. There has been a growing interest in large scale hierarchical classification, which is also emphasized by data mining challenges such as Large Scale Hierarchical Text Classification (LSHTC) challenge [66] [1].

In addition to thousands of categories, typically for hierarchical text classification, number of features range in the order of several hundred thousands to millions. This poses a major challenge for the storage and retrieval of the learned models. For e.g. Large Wikipedia dataset, the largest dataset in LSHTC competition, has 478,021 categories with a feature space of size 1,617,899. A global vector space model for a problem of this size, using all the features, would require a memory of roughly 3TB. This huge space requirement makes it infeasible to solve a problem of this magnitude on a typical computer of moderate capacity.

[1] `http://lshtc.iit.demokritos.gr/`

Furthermore, one of the primary problems of large scale hierarchical classification is data sparsity because majority of the classes do not contain sufficient number of examples to train classifiers with good generalization performance.

Early methods in Large Scale Hierarchical Classification focused on top down local classification methods [1, 48, 63, 67]. Since the classifiers are trained to distinguish only between the children of a node in the hierarchy, the larger classification problem is decomposed into smaller local problems which are considerably easier to handle. In addition, due to the problem transformation nature of this method, any base classifier can be used in learning. The major disadvantage of this approach is the propagation of errors committed in higher levels to the lower levels. To mitigate this problem to some extent, Bennett and Nguyen [68] used a strategy called refined experts. In their method, different from the basic top down approach, the data used for training a node is selected to be similar to what would be expected during the testing phase by using the union of the data predicted at the parent node as well as the actual positive examples. In addition, a bottom up strategy is used to augment the feature set used at a parent node which includes the predictions of the previous lower levels. However, due to the nature of the method, it restricts the parallelism of the algorithm to level by level training of the nodes.

Typically, one tractability issue with large scale classification problems is that the number of categories is huge and classifiers have a hard time trying to deal with the large number of categories. To avoid this problem, Xue *et al.* [69] proposed a two stage classification approach consisting of a category search stage and a classification stage. In the search stage, a relevant set of documents is retrieved from the training corpus and in the classification stage a classifier is trained on the relevant documents to predict the category of the test example. The major bottleneck of this approach is having to train a classifier for every test example. Recently, Gopal *et al.* have proposed two distributed methods [11, 12] to deal with large scale hierarchies. Like many other hierarchical classification methods, the primary assumption in this model as well, is that the weight parameters learned for the parent are similar to those of the children.

3.3 Methods

In the following section we describe a recursive hierarchical regularization method which overcomes the limitations of these methods and is scalable to large scale hierarchical classification problems.

3.3.1 Hierarchical SVM Classification

The objective function of Hierarchical SVM classification proposed by Gopal *et al.* [11], is presented in (3.1).

$$\min_{\{w_t\}_{t\in\mathcal{N}}} \underbrace{\sum_{t\in\mathcal{N}} \frac{1}{2}\|w_t - w_{\pi(t)}\|_2^2}_{regularization} + C \underbrace{\sum_{t\in\mathcal{T}} \sum_{i=1}^{n} \left|1 - y_{it} w_t^T x_i\right|_+}_{loss} \tag{3.1}$$

Here, C denotes the regularization trade-off parameter. The objective function tries to learn a decision boundary similar to one-vs-rest SVM classification for each class with the additional constraint that the model vector learned for a category is constrained to be similar to the model vector for the parent. The similarity is defined as the euclidean norm of the difference of two vectors. Gopal *et al.* [11] proposed a block coordinate descent method for optimizing (3.1) in the context of large scale classification problems in a distributed environment.

3.3.2 Optimization

In serial block coordinate descent method [70] the optimization variables are partitioned into groups or blocks of variables and the optimization proceeds by optimizing the objective function with respect to a single block of variables keeping the variables corresponding to other blocks fixed.

In order to apply block coordinate descent to (3.1), we update each block of variables, w_t for each node $t \in \mathcal{N}$ one at a time, in a sequential manner. For non leaf nodes $t \in \mathcal{N}\backslash\mathcal{T}$,

when the optimization problem (3.1) is restricted to w_t, reduces to (3.2)

$$\min_{w_t} \frac{1}{2} \left(\sum_{c \in \chi(t)} \|w_c - w_t\|_2^2 + \left\|w_t - w_{\pi(t)}\right\|_2^2 \right) \tag{3.2}$$

where $\chi(t)$ represents the set of children of category t in the category tree. The minimizer of (3.2) can be determined analytically which is given by (3.3)

$$w_t = \frac{1}{|\chi(t)| + 1} \left(w_{\pi(t)} + \sum_{c \in \chi(t)} w_c \right) \tag{3.3}$$

When $t \in \mathcal{T}$ is a leaf node, (3.1) w.r.t. the block w_t can be written as

$$\min_{w_t} \frac{1}{2} \|w_t - w_{\pi(t)}\|_2^2 + C \sum_{i=1}^{n} \left|1 - y_{it} w_t^T x_i\right|_+ \tag{3.4}$$

which reduces to an optimization problem similar to standard SVM with the difference that w_t is regularized towards $w_{\pi(t)}$ instead of $\mathbf{0}$ as in the case of standard SVM. To solve (3.4), again co-ordinate descent strategy proposed by Hsieh et at. [71] can be utilized; described below.

3.3.3 Dual Coordinate Descent SVM

In this section, we discuss an efficient optimization procedure for solving the SVM problem for each leaf node given by (3.4). First, we rewrite the primal problem with the introduction of additional slack variables ξ_i as (3.5). We have dropped the subscripts denoting the node in the following discussion to simplify notation; w_t and $w_{\pi(t)}$ are represented by w and w_π respectively.

$$\begin{aligned} \min_{w} \quad & \frac{1}{2} \|w - w_\pi\|^2 + C \sum_{i=1}^{n} \xi_i \\ s.t.\ & \forall i \in 1 \dots n \\ & \xi_i \geq 0 \\ & \xi_i - 1 - y_i w^T x_i \geq 0 \end{aligned} \tag{3.5}$$

Using the standard Lagrangian transformations [72] similar to the case of standard SVM problem we arrive at the dual problem given in (3.6)

$$\begin{aligned} \min_{\{\alpha_i\}_{i=1}^n} \quad & \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{n} \alpha_i \left(1 - y_i {w_\pi}^T x_i\right) \\ s.t. \quad & 0 \leq \alpha_i \leq C \qquad \forall i \in 1 \dots n \end{aligned} \tag{3.6}$$

The relationship between primal and dual variables is given by (3.7)

$$w = w_\pi + \sum_{j=1}^{n} \alpha_j y_j x_j \tag{3.7}$$

We apply co-ordinate descent strategy to solve (3.6). In each step of the coordinate descent algorithm, we minimize (3.6) w.r.t a single dual variable to update α_i to $\alpha_i^{new} := \alpha_i + \delta$ while keeping all the other dual variables $\{\alpha_j \mid j \in [n] \setminus \{i\}\}$ fixed. The minimization of (3.6), w.r.t. δ reduces to (3.8),

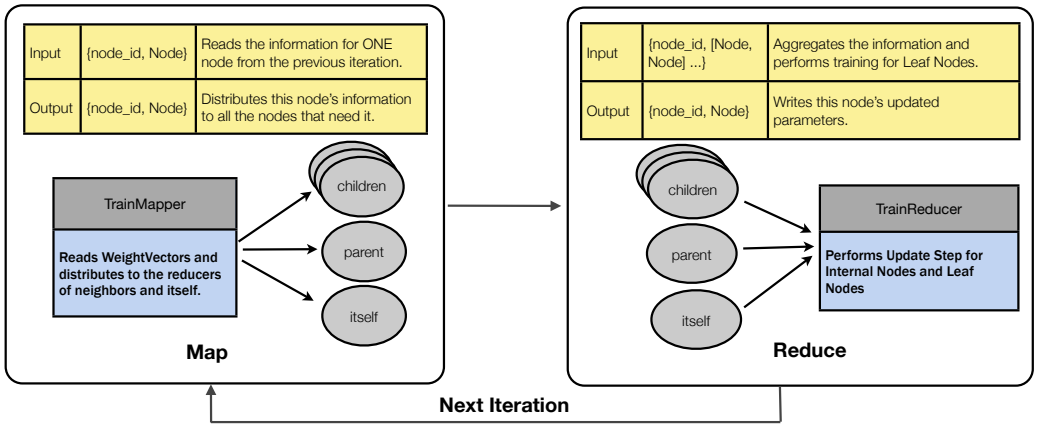


Figure 3.1: Iterative MapReduce processing.

$$\begin{aligned} \min_{\delta} \quad & \frac{1}{2}\delta^2 Q + dG \\ s.t. \quad & 0 \leq \alpha_i + \delta \leq C \end{aligned} \tag{3.8}$$

where $Q = x_i^T x_i$ and $G = \left[y_i x_i^T \left(\sum_{j=1}^{n} \alpha_j y_j x_j + w_\pi \right) - 1 \right]$. The analytical solution for the scalar problem (3.8) is given by (3.9).

$$\alpha_i^{new} = \alpha_i + \delta = \min\left(\max\left(\alpha_i - G/Q, 0\right), C\right) \tag{3.9}$$

Therefore, each macro step of the sequential coordinate descent method iterates over all the dual variables α_i and performs the update given by (3.9). The optimization procedure is summarized in Algorithm 2. A few practical tricks can be used for efficient implementation. Firstly, we can cache $a = \sum_{j=1}^{n} \alpha_j y_j x_j + w_\pi$ and update it to $a_{new} \leftarrow a_{old} + \left(\alpha_i^{new} - \alpha_i^{old}\right) y_i x_i$, whenever an update to the dual variables is performed. Secondly, it has been observed in practice that selecting the coordinates/blocks to be updated at random instead of sequentially iterating through the coordinates/blocks improves convergence [73].

Algorithm 1 HRS2

Initialize $w_t \leftarrow \mathbf{0} \quad \forall t \in \mathcal{N}$
while *Not Converged* **do**
  Update in Parallel
  if $t \in$ *Leaf Nodes* **then**
    $w_t \leftarrow \texttt{BinarySVM}\left(w_\pi, C\right)$
  else
    $w_t \leftarrow \frac{1}{|\chi(t)|+1}\left(w_{\pi(t)} + \sum_{c\in\chi(t)} w_c\right)$
  end
end

Algorithm 2 Binary SVM with parent bias using Dual Co-ordinate Descent

Initialize $w \leftarrow w_\pi, \quad \alpha \leftarrow 0$
while α *not optimal* **do**
  for $i = 1, \ldots n$ **do**
    $G \leftarrow w^T y_i x_i - 1$
    $\alpha_i^* \leftarrow \min\left(\max\left(G/x_i^T x_i, 0\right), C\right)$
    $w \leftarrow w + \left(\alpha_i^* - \alpha_i\right) y_i x_i$
    $\alpha_i \leftarrow \alpha_i^*$
  end
end

3.3.4 Parallelization

In large scale hierarchical classification problems, such as those typically encountered in text classification, solving the optimization problem on a single computer becomes impractical. For large scale problems the storage requirements for the model itself far exceed the memory capacity of a typical moderate sized computer. Secondly, due to the size of the problem, the computational processing required is immense and it would take an inordinate amount of time to solve a large scale optimization problem. Consequently, parallelization of model training becomes necessary. In HRSVM block coordinate descent, the objective function is partially separable w.r.t. the blocks of optimization variables w_t. Specifically, if t represents a leaf node then, w_t is independent of all other nodes except its parent, $w_{\pi(t)}$. Similarly, if t represents an internal node then, t is independent of all other nodes except its children

Algorithm 3 Map

Key: t `// node identifier`
Value: V^t `// node value object`
for $m \in V^t.neighbors$ **do**
    `/* send node object to reducers */`
    `/* of all the neighbors */`
    `output` (m,V^t)
end
`/* send node object to own reducer */`
`output` (t,V^t)

$c \in \chi(t)$ and its parent $\pi(t)$. Therefore, nodes of a tree can be partitioned into two groups by even and odd levels such that the nodes in each group can be independently updated. Instead of sequentially updating the blocks of variables one at a time, in each step, groups of blocks can be updated simultaneously. This update scheme can be extended to hierarchies that are not trees, by partitioning the nodes of the hierarchies such that the nodes within each partition are not connected through an edge. In graph theory terms, this partitioning problem is equivalent to graph coloring problem. However, it is well known that the graph coloring problem is NP-complete. We call this optimization strategy as **HRS1**. The training strategy of partitioning the nodes limits the number of nodes that can be parallely trained. In this work, we propose the training of all the nodes simultaneously without partitioning the nodes into sets of independent blocks. We call this update strategy **HRS2**. Unlike HRS1, HRS2 strategy does not yield an exact block coordinate descent algorithm because it does not restrict the parallel updates to independent sets of variables. Hence, it parallely updates even the dependent blocks, which in some cases might result in inexact updates. This pseudo-code for HRS2 is presented in Algorithm 1. The `BinarySVM(·)` method is implemented using coordinate descent strategy discussed in section 3.3.3.

Algorithm 4 Reduce

Key: t `// node identifier`

Value: $\mathrm{W} := \{V^m \mid m \in \texttt{neighbors}(t)\} \cup \{V^t\}$ `// List of node objects`

Data: trainingData `// training data on DistributedCache`

`/* Compute average model vector of neighbors */`

$\bar{w} \leftarrow \mathbf{0}$

for $m \in V^t.neighbors$ **do**

| $\bar{w} = \bar{w} + V^m.w$

end

$$\bar{w} \leftarrow \frac{1}{|V^t.neighbors|}\bar{w}$$

`/* For leaf node, update model vector through Binary SVM training using` $\bar{w}$ `as bias. For non-leaf nodes set the new weight to the` $\bar{w}$ `*/`

if $V^t.isleaf$ **then**

| $V^t.w \leftarrow \texttt{BinarySVM}(\mathrm{trainingData}, \bar{w}, C)$

else

| $V^t.w \leftarrow \bar{w}$

end

`output` (t, V^t)

3.3.5 MapReduce Implementation

Map-Reduce [14] is a distributed computing paradigm designed to facilitate implementation of distributed code by taking care of common communication overheads involved in distributed computing. Map-Reduce programs are composed of user defined `map` and `reduce` constructs which encode the behavior of the parallel program. Map-Reduce framework communicates the information between mappers, reducers, and persistent store as key-value pairs.

In our implementation the mappers and reducers use the node identifiers (t) of the hierarchy nodes as keys and the corresponding value is an object (V^t) containing the model vector for the node ($V^t.w$), a list of its neighbors' node identifiers ($V^t.neighbors$), a boolean flag indicating whether the node is a leaf node or not ($V^t.isLeaf$), and block partition of the node ($V^t.set$), which is used in HRS1 exact block coordinate descent for training groups by odd/even levels. Each iteration of Algorithm 1 is performed as a separate map-reduce job. The iterative process is depicted in Fig. 3.1. In the map phase, summarized in Algorithm 3, each mapper reads the node information of a single node and scatters it to all the reducers of its neighbors and its own reducer using key-value pairs. The reducer, summarized in Algorithm 4, gets a list of nodes' information of its neighbors, itself and its children. It then decides the update rule depending on whether the node is a leaf node or not and either computes an average of the model vectors for internal nodes or trains a binary SVM for leaf nodes using the parent's model vectors as the prior bias. Finally, it writes the updated model to a persistent file system, which is then read by the next iteration's map job. The training data which is utilized by all the reducers for the leaf nodes made available to them using the DistributedCache mechanism of Hadoop, which can be used to distribute read-only copies of data to the slave nodes. The source code for our implementation has been made available at `http://www.cs.gmu.edu/~mlbio/supplements/lshc`.

Table 3.1: Dataset statistics.

Dataset	Nodes	Leaves	Edges	Height	Training	Testing	Features
CLEF	97	63	96	4	10,000	1,006	80
IPC	553	451	552	4	46,324	28,926	1,123,497
DMOZ-SMALL	2,388	1,139	2,387	6	4,463	1,858	51,033
DMOZ-2010	17,222	12,294	17,221	6	93,805	34,880	347,256
DMOZ-2012	13,963	11,947	13,962	6	383,408	103,435	575,555

3.4 Experimental Setup

In the following sections we discuss the empirical evaluations and compare different update strategies for HRSVM and flat SVM. On one of the datasets, dmoz-2010, we have performed more detailed analysis of the results.

3.4.1 Datasets

We used the following datasets for evaluating the hierarchical classifiers. **CLEF** [74] is a dataset comprising of medical images annotated with Image Retrieval in Medical Applications (IRMA) codes. Images are described with 80 features extracted using a technique called local distribution of edges. IRMA codes are hierarchically organized. **IPC** is a collection of patent documents classified according to the International Patent Classification (IPC) [2]. **DMOZ-small, DMOZ-2010, DMOZ-2012** are hierarchical text classification datasets released as part of PASCAL Large Scale Hierarchical Text Classification Challenge (LSHTC) [3]. Since the competition involves blind evaluation, the labels for the test set are not publicly available. However, the competition website provides access to an on-line evaluation system which can compute certain performance metrics for the submitted predictions. Statistics for the datasets are summarized in Table 3.1. For all the datasets used here, the hierarchy has a tree structure and the internal nodes define a virtual category tree, i.e. the examples are assigned to only leaf nodes directly. For text datasets, we applied tf-idf transformation with $l2 - norm$ normalization to the word frequency features.

[2] `http://www.wipo.int/classifications/ipc/en/`

[3] `http://lshtc.iit.demokritos.gr/`

3.4.2 Validation Protocols

For the coordinate descent SVM optimization procedure described in section 3.3.3, we have set the optimization tolerance to 10^{-2} and maximum number of iterations to 10^3. We fixed the number of iteration over outer block coordinate descent to 10, for HRS1 and HRS2, which showed reasonable convergence. Regularization parameter C is tuned using a validation set. The model is trained for a range of values for parameter $C \in \{10^{-3}, \ldots, 10^3\}$ and the best model is selected using the validation set. We retrained the models using the best parameters on the entire training set and measured the performance on a held out test set.

We implemented the HRSVM classifier using both HRS1 and HRS2 training strategies with Apache Hadoop map-reduce framework. The experiments were performed on a MapReduce cluster using Hadoop version 1.2.1 on 15 Dell C8220 nodes with dual Intel Xeon E5-2670 8 core CPUs. Each node has a physical memory of 64 GB RAM. Due to resource sharing with other cluster processes only 2 map and 2 reduces slots per machine were available for Hadoop. We allocated a maximum of 4GB memory for each of the worker processes.

Table 3.2: Performance comparison on set based evaluation measures.

	Our results			Results from [11]		
Dataset	SVM	HRS1	HRS2	HRLR	TD	HRSVM
CLEF						
Macro-F_1	50.75	51.56	51.25	55.83	32.32	53.92
Micro-F_1	78.33	79.03	78.93	80.12	70.11	80.02
IPC						
Macro-F_1	45.19	44.56	44.87	49.60	42.62	47.89
Micro-F_1	51.12	50.53	50.81	55.37	50.34	54.26
DMOZ-SMALL						
Macro-F_1	35.09	35.15	35.37	28.48	20.01	28.94
Micro-F_1	48.01	48.33	48.55	45.11	38.48	45.31
DMOZ-2010						
Macro-F_1	32.42	33.24	33.21	32.42	22.30	33.12
Micro-F_1	43.69	44.04	44.02	45.84	38.64	46.02
DMOZ-2012						
Macro-F_1	35.41	36.05	36.10	20.04	30.01	33.05
Micro-F_1	53.41	53.79	53.76	53.18	55.14	57.17

3.5 Results

3.5.1 Effectiveness

Table 3.2 compares the micro and macro F_1 scores for the different methods discussed in this chapter. From the evaluations performed by Gopal *et al.* [11], we extracted the results of two baselines which scaled to the datasets. These baselines are Top Down (TD), top down Pachinko style support vector machine classifier and Hierarchical Logistic regression (HRLR), which is modeled in a similar fashion to HRSVM with the only difference that it uses Logistic loss instead of SVM's hinge loss in the objective. We also include the HRSVM results reported there, for comparison with the results we obtained.

As can be seen from the results, the performances of both implementations of HRSVM are equivalent and better that of the flat SVM, on most datasets. However HRSVM did not consistently outperform SVM as was also concluded in [11]. We found that the difference between the hierarchical and non-hierarchical methods is a little less pronounced in our evaluations. Compared to HRSVM results reported earlier, we found improvement in macro-F_1 scores in our experiments at a compromise of micro-F_1 scores.

Table 3.3 reports the performance using the hierarchical evaluation measures for the different methods, for the datasets where these scores could be obtained. For DMOZ-2010 and DMOZ-2012 the labels of held out sets are not available, and the scores are obtained from an on-line evaluation system. Therefore, some of the scores could not be reported. The results for the hierarchical evaluation also shows the superiority of HRSVM, which beats SVM in all cases except the IPC dataset on all evaluation measures. One point to note is that for hierarchical datasets where all the leaf nodes are at the same depth, as is the case for CLEF and IPC datasets, the hP, hR and consequently hF_1 will have exactly the same values.

Table 3.3: Performance comparison using hierarchical evaluation measures.

Dataset	hP	hR	hF_1	TE
CLEF				
SVM	81.15	81.15	81.15	1.131
HRSVM(HRS2)	82.70	82.70	82.70	1.038
IPC				
SVM	63.10	63.10	63.10	2.196
HRSVM(HRS2)	62.62	62.62	62.62	2.225
DMOZ-SMALL				
SVM	61.83	62.21	62.02	3.635
HRSVM(HRS2)	62.36	62.72	62.54	3.584
DMOZ-2010				
SVM	-	-	-	3.655
HRSVM(HRS2)	-	-	-	3.617
DMOZ-2012				
SVM	72.81	72.93	72.87	-
HRSVM(HRS2)	73.14	73.27	73.19	-

Table 3.4: Training times (in min.).

Dataset	SVM	HRS1	HRS2
CLEF	0.2	1.2	1.0
IPC	26.1	56.4	44.8
DMOZ-SMALL	6.7	14.9	12.7
DMOZ-2010	27.1	327.4	227.4
DMOZ-2012	69.6	641.3	488.0

3.5.2 Efficiency

In Table 3.4, we report the measured wall-clock time for the training of various models. These results show the main advantage of using our proposed update strategy of HRS2 over HRS1. The training times for large datasets were reduced by approximately 20-30% using HRS2, whereas for the smaller datasets the training times were reduced by approximately 15%. However, we note that the training times of both HRSVM methods were significantly higher than those for flat SVM, especially for larger datasets.

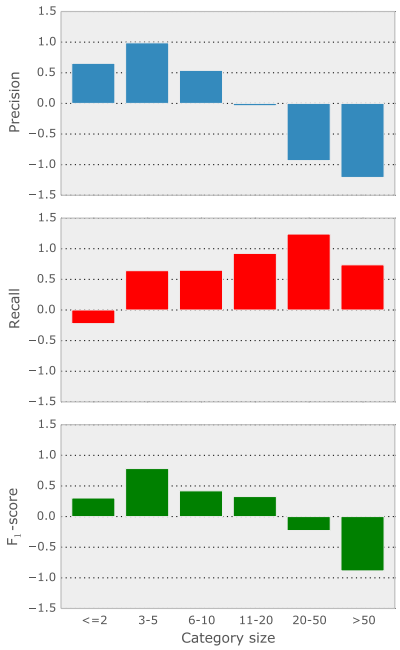


Figure 3.2: Average improvement in precision, recall, and F_1-score for HRSVM over SVM for categories of different sizes for DM-25 dataset.

3.5.3 Detailed analysis on DMOZ-2010

In the following sections, we present a detailed comparison of HRSVM and SVM methods for different aspects of the training data and number of examples in a category. For these experiments, the complete dmoz-2010 dataset could not be used due to two reasons. First, the labels for the test set are not available. Only the evaluation scores from predictions can be computed by submitting the results to an on-line evaluation system [4]. Secondly, for measuring the performance of variable training size, we do not have sufficient training examples in the all the categories. In order to address these issues, we removed the categories with few positive examples and created partitions of the original completed training dataset for testing.

[4] `http://lshtc.iit.demokritos.gr/node/81`

Effect of training set size

For these experiments, we created 5 equal sized partitions. To ensure that each fold has at least one example from each class, we filtered out the classes with less than 5 examples. For the training set, only 7087/12294 (57%) of the classes have more than 5 examples. The total number of examples in classes with ≥ 5 examples is 105894/115839 (88%). With the removal of the leaf categories with fewer than 5 examples, some of the corresponding internal nodes were also removed. The final number of nodes retained (internal as well as leaf nodes) is 10931/17222 (64%), out of which 3844/4928 (78%) of the internal nodes were retained. We fixed one split of the data as test set and used the remaining four sets for creating training sets containing 25%, 50%, 75% and 100% of the training data. Hereafter, we refer to these datasets as DM-25, DM-50, DM-75 and DM-100 respectively.

The results of training with variable training data proportions is reported in Table 3.5. As is expected, both the micro-F_1 and macro-F_1 scores for SVM as well as HRSVM increase with more training data, however it seems that the difference between SVM and HRSVM is more pronounced for smaller training sizes. For the larger dataset the improvement of HRSVM over SVM is minimal. Since, for each training set proportions (25%, 50%, 75%, and 100%) we could replicate the datasets for 5 different splits, we performed paired t-test for the scores and found the differences in performance between SVM and HRSVM to be statistically significant.

Performance by class size

One of the major challenges in large scale hierarchical classification is the lack of sufficient labeled examples for smaller categories. In fact majority of the categories have very few positive examples to learn from. In order to assess performance of HRSVM in comparison to SVM with respect to the number of positive examples in a category, we analyzed the performance per category of the two methods on DM-25 dataset. We chose this dataset because we created it such that at least some categories have less than two examples to highlight the data sparsity issue.

Table 3.5: Performance on varying training set size for DMOZ-2010 (mean ± std). The cells marked with † are statistically significant at p-value of 5%, rest are significant at p-value of 1%, with paired t-test on results for 5 independent splits.

Size	Method	Micro-F_1	Macro-F_1	hP	hR	hF_1	TE
25%	HRSVM (HRS2)	38.55 ± 0.29	21.14 ± 0.19	55.51 ± 0.25	55.75 ± 0.24	55.63 ± 0.24	4.17 ± 0.02
	SVM	38.08 ± 0.27	20.73 ± 0.14	54.91 ± 0.21	55.34 ± 0.22	55.13 ± 0.22	4.22 ± 0.02
50%	HRSVM (HRS2)	43.53 ± 0.28	27.41 ± 0.25	59.88 ± 0.24	60.08 ± 0.18	59.98 ± 0.19	3.76 ± 0.02
	SVM	43.19 ± 0.32	26.82 ± 0.18	59.39 ± 0.23	59.79 ± 0.26	59.59 ± 0.24	3.80 ± 0.02
75%	HRSVM (HRS2)	45.95 ± 0.33†	30.64 ± 0.31	61.84 ± 0.19	62.13 ± 0.23†	61.98 ± 0.21	3.57 ± 0.02
	SVM	45.76 ± 0.30	30.09 ± 0.26	61.54 ± 0.20	61.98 ± 0.25	61.76 ± 0.22	3.60 ± 0.02
100%	HRSVM (HRS2)	47.74 ± 0.40	33.14 ± 0.34	63.23 ± 0.19	63.52 ± 0.23†	63.37 ± 0.21	3.44 ± 0.02
	SVM	47.50 ± 0.39	32.49 ± 0.37	62.95 ± 0.18	63.38 ± 0.24	63.17 ± 0.21	3.46 ± 0.02

Fig. 3.2 shows the difference in average scores of HRSVM and SVM for different size classes. It can be seen that in general the recall of HRSMV tends to be higher. The possible explanation for this observation is that by biasing the classification hyperplane of a category towards the parent category, we make its decision boundary more inclusive. However, the interesting observation with respect to precision of the classifier is that for smaller class sizes, HRSVM tends to perform better up to a point where the class size is in the range of 10-20 and then the performance starts to deteriorate. Consequently the overall F_1-score also takes a hit for larger categories. Hence, for classes with sufficient number of examples biasing the hyperplane towards the parent negatively affects the performance.

3.6 Summary

In this work we study the problem of large scale hierarchical text classification. We have proposed an inexact distributed block coordinate descent update method for training the Hierarchical SVM (HRSVM) formulation proposed by Gopal *et al.* [11], which offers 15-30%

of saving in training time in our MapReduce implementation without any decrease in the effectiveness of the trained classifier. Our detailed analysis of the performance of the two HRSVM training schemes demonstrates their effectiveness for solving large scale hierarchical classification problems compared to binary classification and top-down classification. In order to understand the various properties of hierarchical datasets on performance, we performed detailed analyses on large scale datasets by varying training size; studying the effect of training size per category on the precision and recall of the classifier; and finally analyzing the similarity of the learned models for related categories.

Chapter 4: Large Scale Hierarchical Classification using Cost Sensitive Learning

4.1 Introduction

We discussed a recursive regularization based method for large scale classification in the previous chapter. Although regularization methods which constrain the learned models to be close to its neighboring classes according to the hierarchy have been effective, they induce large scale optimization problems which require specialized solutions [21]. Even though distributed optimization methods are able to scale well to extremely large scale scenarios, the global optimization of all the model variables in an integrated fashion incurs communication overhead, which can sometimes be a considerable. Additionally, due to the recursive nature of regularization the learning algorithm performs several basic iterations of the model training for the terminal classes.

The work presented in this chapter, addresses some of these shortcomings by a different approach to hierarchical regularization. Instead of encoding problem information through regularizaton, we shift the burden on to the loss function, by noting that the regularization that biases towards neighbors, in effect, achieves the same objective through indirect means. This observation, obviates the need for hierarchical regularizaton and presents a simple cost sensitive learning alternative to hierarchical classification. By decoupling the training in such manner, we are essentially able to leverage the benefits of hierarchical classification at practically the same training cost as flat classification. These models can be trained in parallel without incuring any communication cost. Additionally, by up-weighting the importance of smaller categories, we are also able to achieve a better performance on these categories. Therefore, the method proposed here address two main issues of large scale hierarchical classification, class imbalance and training efficiency, by extending the flat

classification approach using cost sensitive training examples. We study various methods to incorporate cost-sensitive information into hierarchical classification and empirically evaluate their performance on several datasets. Finally, one additional advantage of this methods is that we can use any cost sensitive base classifier, and therefore the HC problem is able to benefit from advancement in this area.

Notations $\mathcal{N}$ is used to denote the set of all hierarchical nodes. $\mathcal{T}$ denotes the set of terminal nodes which can be assigned as class labels to instances. The class labels of instances have dual representation for convenience. We use $l_i \in \mathcal{N}$ to denote the class identifier and $y_{it} \in \{-1, +1\}$ as binary labels denoting class membership. For hierarchical single-label problems $y_{it} = +1$ if $l_i = t$ and $y_{it} = -1$ otherwise, and for hierarchical multi-label problems l_i denotes a set of labels and equality can be replaced by set membership, *i.e.* $y_{it} = +1$ iff $t \in l_i$ and $y_{it} = -1$ iff $t \notin l_i$. We use c_{it} to denote the cost of example i in training of the model for class t. Where the class is implicitly understood to be t, we drop the sub-script explicitly indicating the class to simplify notation, and use y_i , c_i and w in place of y_{it}, c_{it} and w_t. In the current work, logistic loss function is used, which is defined as $\mathcal{L}(y, f(x)) = \log(1 + \exp(-yf(x)))$.

4.2 Motivation and Related Work

In this section, we discuss the motivation for the approach taken in the current work and examine various related methods proposed in the literature for addressing the hierarchical classification problem.

Several large margin methods have been proposed as cost sensitive extensions to the multi-class classification problem. Dekel et al. [59] proposed a large margin method where the margin is defined with respect to the tree distance. Although their method shows improvement on tree-error, the performance degrades with respect to misclassification error. The methods proposed by Cai et al. [61] and more recently by Chen et al. [75], also make an argument in favor of modifying the misclassification error by making it dependent on the

hierarchy. Both these methods can be seen as special cases of a more general large margin structured output prediction method proposed by Tsochantaridis et al. [76]. Although all these methods try to incorporate cost sensitive losses based on the hierarchy, they formulate a global optimization problem where the models for all the classes are learned jointly and are not scalable to large scale classification problems.

Several methods try to incorporate the bias that categories which are semantically related according to the hierarchy should also be similar with respect to the learned models. McCallum et al. [16] show that for Naive Bayes classifier, smoothing the parameter estimates of the data-sparse children nodes with the parameter estimates of parent nodes, using a technique known as shrinkage, produces more robust models. Other models in this class of methods typically incorporate this assumption using parent child regularization or hierarchy based priors [11,16,77]. In one of the prototypical models in this class of works, which extends Support Vector Machines (SVM) and Logistic Regression (LR) [11], the objective function takes the form given in (4.1),

$$\min_{w_1,\ldots,w_{|\mathcal{N}|}} \sum_{t\in\mathcal{N}} \frac{1}{2} \left\| w_t - w_{\pi(t)} \right\|_2^2 + C \sum_{t\in\mathcal{T}} \sum_{i=1}^{n} \mathcal{L}\left(y_{it}, w_t^T x_i\right) \tag{4.1}$$

where, $\pi(t)$ represents the parent of the class t according to the provided hierarchy, and C denotes the loss/regularization trade-off parameter. The loss function $\mathcal{L}$ has been modeled as logistic loss or hinge loss. Note that the loss is defined only on the terminal nodes $\mathcal{T}$, and the non-terminal node $\mathcal{N} - \mathcal{T}$, are introduced only as a means to facilitate regularization. Since the weights associated with different classes are coupled in the optimization problem, Gopal et al. [11] used a distributed implementation of block coordinate descent where each block of variables corresponds to w_t for a particular class t. The model weights are learned similarly to standard LR or SVM for the leaf nodes $t \in \mathcal{T}$, with the exception that the weights are shrunk towards parents instead of towards the zero vector by the regularizer. For the internal non-leaf nodes, the weights updates are averages of the other nodes which are connected to it according to the hierarchy, i.e., the parents and children in the hierarchy.

The kind of regularization used here can be compared to the formulations proposed in transfer and multi-task learning literature [13], where externally provided task relationships can be utilized to constrain the jointly learned model weights to be similar to each other. In the case of HC, the task relationship are explicitly provided as hierarchical relationships. However, one significant difference between the application of this regularization between HC and MTL is that the sets of examples in MTL for different tasks are, in general, disjoint. Whereas, in the case of HC, the examples which are classified as positive for one class are negative for all other classes except those which belong to the ancestors of that class. Therefore, even though these models impose similarity between siblings indirectly through the parent, when their respective models are trained, the negative and positive examples are flipped. Hence, the opposing forces for examples and regularization are acting simultaneously during the learning of these models. However, due to the regularization strength being imposed by the hierarchy, the net effect is that the importance of misclassifying the examples coming for nearby classes is down-weighted. This insight can be directly incorporated into the learning by defining the loss of nearby negative examples for a class, where "near" is defined with respect to the hierarchy, to be less severe than the examples which are farther. This leads to a simple cost sensitive classification where the misclassification cost is directly proportional to the distance between the nodes of the classes, which is the key contribution of our work. With respect to prediction there are only two classes for each trained model, but the misclassification costs of negative examples are defined according to which node they originate from, with respect to the class for which the binary classifier is being learned.

In this framework for HC, we essentially, decouple the learning of multiple models of the hierarchy and train each one independently. Thus, rendering scalability to this method. Instead of jointly formulating the learning of the model parameters for all the classes, we turn the argument around from that of regularizing the model weights towards those of the neighboring models, to the rescaling the loss of example depending on the class relationships. A similar argument was made in the case of multi-task transfer learning by Saha et al. [78], where, in place of joint regularization of model weights, as is typically done in multi-task

learning [10], they augment the target tasks with examples from source tasks. However, the losses for the two sets of examples are scaled differently.

4.3 Methods

As shown in some previous works [11, 79], the performance of flat classification has been found to be very competitive, especially for large scale problems. Although, the top-down classification method is efficient in training, it fares poorly with respect to classification performance due to error propagation. Hence, in this work, we extend the flat classification methodology to deal with HC. We use the one-vs-all approach for training, where for each class t, to which examples are assigned, we learn a classification model with weight w_t. Note, that it is unnecessary to train the models for non-terminal classes, as they only serve as virtual labels in the hierarchy. Once the models for each terminal class are trained, we perform prediction for input example x as per (4.2)

$$\hat{y} = \mathbf{argmax}_t \; w_t^T x \tag{4.2}$$

The essential idea is formulate the learning algorithm, such that the mis-predictions on negative examples coming from nearby classes are treated as being less severe. We encode this assumption through cost sensitive classification. Standard regularized binary classification models, such as SVMs and Logistic Regression, minimize an objective function consisting of loss and regularization terms as shown in (4.3).

$$\min_w \underbrace{\sum_{i=1}^{n} \mathcal{L}\left(y_i, f\left(x_i \mid w\right)\right)}_{loss} + \rho \underbrace{\mathcal{R}\left(w\right)}_{regularizer} \tag{4.3}$$

where ρ controls the trade-off between regularization and loss. Here, the loss function $\mathcal{L}$, which is generally a convex approximation of zero-one loss, measures how well the model fits the training examples. Here, each example is treated as equally important. As per

the arguments made previously, we modify the importance of correctly classifying examples according to the hierarchy using example based costs. For models such as logistic regression, incorporating example based costs into the learning algorithm is simply a matter of scaling the loss by a constant positive value. Assuming that the classifier is being learned for class n, we can write the cost sensitive objective function as shown in (4.4).

$$\min_{w} \sum_{i=1}^{n} c_i \mathcal{L}\left(y_i, w^T x_i\right) + \rho \mathcal{R}\left(w\right) \tag{4.4}$$

Here, c_i denotes the cost associated with misclassification of the i^{th} example. Although, this scaling works for the smooth loss function of Logistic Regression, it is not as straightforward in the case of non-smooth loss functions such as hinge loss [80]. Therefore, using the formulation given in (4.4), for each of the models, we can formulate the objective function for class t as a cost sensitive logistic regression problem where the cost of the example x_i for the binary classifier of class t depends on how far the actual label $l_i \in \mathcal{T}$ is from t, according to the hierarchy. Additionally, to deal with the issue of rare categories, we can also increase the cost of the positive examples for data-sparse classes thus mitigating the effects of highly skewed datasets. Since our primary motivation is to argue in favor of using hierarchical cost sensitive learning instead of more expensive regularization models, we only concentrate on logistic loss, which is easier to handle, from optimization perspective, than non-smooth losses such as SVM's hinge loss. The central issue, now, is that of defining the appropriate costs for the positive and negative examples based on the distance of the examples from the true class according to the hierarchy and the number of examples available for training the classifiers. In the following section we discuss the selection of costs for negative and positive examples.

4.3.1 Cost Calculations

Hierarchical Cost.

Hierarchical costs impose the requirement that the misclassification of negative examples that are farther away from the training class according to the hierarchy should be penalized more severely. Encoding this assumption, we define the following instantiations of hierarchical costs. We assume the class under consideration is denoted by t.

Tree Distance (TrD): In (4.5), we define the cost of negative examples as the undirected graphical distance, $\gamma(t, l_i)$, between the class t and l_i, the class label of example x_i. We call this cost *Tree Distance (TrD)*. We define $\gamma_i \equiv \gamma(t, l_i)$ and $\gamma_{max} = \max_{j \in \mathcal{T}} \gamma_j$. Since dissimilarity increases with increasing γ_i, the cost is a monotonically increasing function of γ_i.

$$c_i = \begin{cases} \gamma_{max} & l_i = t \\ \gamma_i & l_i \neq t \end{cases} \tag{4.5}$$

Number of Common Ancestors (NCA): In some applications, where the depth (distance of a node from the root node) of all terminal labels is not uniform, a better definition of similarity might be the number of common ancestor between two nodes. This is encoded in *NCA* costs, represented in (4.6). In the definition, α_i is used to denote the number of common ancestors between the pair of nodes l_i and t. Unlike γ_i which is a monotonically increasing function of dissimilarity, α_i is a monotonically increasing function of similarity. $\alpha_{max} = \max_{j \in \mathcal{T}} \alpha_j$.

$$c_i = \begin{cases} \alpha_{max} + 1 & l_i = t \\ \alpha_{max} - \alpha_j + 1 & l_i \neq t \end{cases} \tag{4.6}$$

Exponentiated Tree Distance (ExTrD): Finally, in some cases especially for deep hierarchies, the tree distances can be large, and therefore, in order to shrink the values of cost into a smaller range, we define *ExTrD* in (4.7), where $k > 1$, can be tuned according to the hierarchy. Through tuning we found that on our dataset, the range of values $1.1 \leq k \leq 1.25$ of works well.

$$c_i = \begin{cases} k^{\gamma_{max}} & l_i \neq t \\ k^{\gamma_i} & l_i \neq t \end{cases} \tag{4.7}$$

In all these cases, we set the cost of the positive class to the maximum cost of any example.

Imbalance Cost.

In certain cases, especially for large scale hierarchical text classification, some classes are extremely small with respect to the number of examples available for training. In these cases, the learned decision boundary might favor the larger classes. Therefore, to deal with this imbalance in the class distributions, we increase the cost of misclassifying rare classes. This has the effect of mitigating the influence of skew in the data distributions of abundant and rare classes. We call the cost function incorporating this as *Imbalance Cost (IMB)*, which is given in (4.8). We noticed that using cost such as inverse of class size diminishes the performance. Therefore, we use a squashing function inspired by logistic function $f(u) = L/\left[1 + \exp -k\left(u - u_0\right)\right]$, which would not severely disadvantage very large classes.

$$c_i = 1 + L/\left[1 + \exp\left(\left|n_i - n_0\right|_+\right)\right] \tag{4.8}$$

where $|a|_+ = \max(a, 0)$ and n_i is the number of examples belonging to class denoted by l_i. The value of c_i lies in the range $(1, L/2 + 1)$. We can use a tunable parameter n_0, which can be intuitively interpreted as the number of examples required to build a "good" model, above which increasing the cost does not have a significant effect or might adversely affect

the classification performance. In our experiments, we used $n_0 = 10$ and $L = 20$.

In order to combine the Hierarchical Cost as well as the Imbalance Costs, we simply multiply the contributions of both the costs. We also experimented with several other hierarchical cost variants, which are not discussed here due to space constraints.

4.3.2 Optimization

Since we are dealing with large scale classification problems, we need an efficient optimization method which relies only on the first order information to solve the learning problem given in (4.9).

$$\min_{w} \left[f(w) = \sum_{i=1}^{n} c_i \log\left(1 + \exp\left(-y_i w^T x_i\right)\right) + \rho \|w\|_2^2 \right] \tag{4.9}$$

Since the cost values c_i are predefined positive scalars, we can adapt any method used to solve the standard regularized Logistic Regression (LR). However, we make use of accelerated gradient descent due to its efficiency and simplicity. The ordinary gradient descent method has a convergence rate of $O(1/k)$, where k is the number of iterations. The gradient method can be accelerated by using the gradient information from the previous iteration [81] to improve the rate of convergence to $O\left(1/k^2\right)$. The complete algorithm to solve the cost-sensitive binary logistic regression is provided in Algorithm 5. We describe the notations and expressions used to describe the algorithm below.

n is the number of examples; $X \in \mathbb{R}^{n \times d}$ denotes the data matrix; $\mathbf{y} \in \{\pm 1\}^n$ is the binary label vector for all examples; $\rho \in \mathbb{R}_+$ is the regularization parameter; $\mathbf{c} = (c_1, c_2, \ldots, c_n) \in \mathbb{R}_+^n$ denotes the cost vector, where c_i is the cost for example i ; $w \in \mathbb{R}^d$ denotes the weight vector learned by the classifier; $f(w)$ denotes the objective function value, given in (4.10)

$$f(w) = \mathbf{c}^T \left(\log\left[1 + \exp\left(Xw\right)\right]\right) + \rho \|w\|_2^2 \tag{4.10}$$

$\nabla \mathbf{f}$ is the gradient of f w.r.t. w, which is defined in (4.11), where $(\mathbf{y} \circ \mathbf{c})$ denotes the vector

Algorithm 5 Accelerated Gradient Method for Cost Sensitive LR

Data: $X, \mathbf{y}, \mathbf{c}, \rho, \beta \in (0,1), max\_iter$
Result: w
Let $\lambda_0 := 1; w_{-1} = w_0 = \mathbf{0}$ **for** $k = 1 \ldots max\_iter$ **do**
  $\theta^k = \frac{k-1}{k+2}$
  $\lambda = \lambda_{k-1}$
  while *TRUE* **do**
    $w = u_k - \lambda \nabla f(u_{k-1})$
    if $f(w) \leq \hat{f}_\lambda(u, w)$ **then**
      $\lambda_k = \lambda$
      $w_k = w$
      break
    else
      $\lambda = \beta\lambda$
    end
  end
  if *converged* **then**
    break
  end
end
return w_k

obtained from the element-wise product of $\mathbf{y}$ and $\mathbf{c}$.

$$\nabla \mathbf{f}(w) = 2w + X^T \left(\frac{-\mathbf{y} \circ \mathbf{c}}{1 + \exp\{(Xw) \circ \mathbf{y}\}} \right) \tag{4.11}$$

$\hat{f}_\lambda(u, w)$, described in (4.12), is the quadratic approximation of $f(u)$ at w using approximation constant or step size λ. The appropriate step size in each iteration is found using line search.

$$\hat{f}_\lambda(u, w) = f(w) + (u - w)^T \nabla \mathbf{f}(w) + 1/2\lambda \|u - w\|_2^2 \tag{4.12}$$

4.3.3 Dealing with Hierarchical Multi-label Classification

HC problems are trivially multi-label problems because every example belonging to terminal class also inherits the labels of the ancestor classes. But in the current context, we call a problem as hierarchical multi-label problem if an example can be assigned multiple labels

such that neither is an ancestor or descendant of the other.

In the case of single label classification, we perform prediction as per (4.2), which selects only a single label per example. A trivial extension to multi-label classification can be done by choosing a threshold of 0 such that we assign label t to example x if $w_t^T x > 0$ as in the case of binary classification. However, a better strategy is to optimize the threshold s_t for each class using a validation set, such that the label t is assigned to the test example if $w_t^T x > s_t$. This strategy is called SCut method [82]. Other strategies such as learning a thresholding function $s\left(w_1^T x, w_2^T x, \ldots, w_{|\mathcal{T}|}^T x\right)$ using the margin scores [11] might improve the results, but they are somewhat more expensive to tune for large scale problems. The SCut method can tune the threshold independently of all other classes. In cases where we do not have sufficient examples to tune the threshold, i.e. the class has a single training example, we set the threshold to $s_t = 0$.

The second issue that we must deal with is the definition of cost based on hierarchical distances and class sizes. With respect to the training of a class t, an example x_i might be associated with multiple labels $l_1, l_2 \ldots, l_K$. In this case the tree distance γ_i is not uniquely defined. Hence, we must aggregate the values of $\gamma\left(t, l_1\right), \ldots, \gamma\left(t, l_K\right)$. One strategy is to use an average of the values, but we found that the taking the minimum works a little better. Similarly we can use a minimum of of the number of common ancestors to all target labels for *NCA* costs.

Finally, since an example is associated with multiple class labels, the class size n_i of the examples is also not uniquely defined, in this case as well, we use the the effective size as the minimum size out of all the labels associated with x_i for our *IMB* cost. It also makes intuitive sense, because we are trying to up-weight the rare classes, and the rarest class should take a precedence in terms of the cost definition.

Table 4.1: Dataset statistics.

Dataset	Nodes	Labels	Depth	#Train	#Test	#Feat	LCard
CLEF	97	63	4	10000	1006	80	1.00
DMOZ-SMALL	2388	1139	6	4463	1858	51033	1.00
IPC	553	451	4	46324	28926	345479	1.00
RCV1	117	101	6	23149	781265	48728	3.18
DMOZ-2010	17222	12294	6	128710	34880	381580	1.00
DMOZ-2012	13963	11947	6	383408	103435	348548	1.00

4.4 Experimental Setup

4.4.1 Datasets

The details of the datasets used for our experimental evaluations are provided in Table 4.1, which provides summary information such as number of training (#Train) and test examples (#Test), number of features (#Feat), and label cardinality (LCard), which is equal to the average number of labels per example. Nodes refers to the total number of nodes in the hierarchy and Labels are the number of nodes to which examples can be assigned. Depth is the maximum number of edges in the path from any leaf node to the root node. **CLEF** [74] is a dataset comprising of medical images annotated with hierarchically organized Image Retrieval in Medical Applications (IRMA) codes. The task is to predict the IRMA codes from image features. Images are described with 80 features extracted using a technique called local distribution of edges. **IPC** is a collection of patent documents classified according to the International Patent Classification (IPC) [1]. **DMOZ-small, DMOZ-2010** and **DMOZ-2012** are hierarchical text classification datasets released as part of PASCAL Large Scale Hierarchical Text Classification Challenge (LSHTC) [2]. For LSHTC datasets, labels of the test datasets are not available, but certain classification metrics can be obtained through their online evaluation system. **RCV1-v2** [83] is a multi-label text classification dataset

[1] http://www.wipo.int/classifications/ipc/en/
[2] http://lshtc.iit.demokritos.gr/

extracted from Reuters corpus of manually categorized newswire stories. **RCV1** is the only multi-label dataset, rest of the datasets are single label hierarchical datasets. For all the text datasets, raw term frequencies were converted to term weights using Cornell ltc term weighting [83].

4.4.2 Validation Protocols

In our experimental evaluations, we compare our cost sensitive hierarchical classification methods with the following hierarchical and flat classification methods proposed in the literature.

Logistic Regression (LR) One-vs-rest binary logistic regression is used in the conventional flat classification setting. For single label classification, we assign test examples to the class which achieves best classification score.

Hierarchical Regularization for LR (HRLR) This method proposed by Gopal et al. [11], extends flat classification using recursive regularization based on hierarchical relationships. This was the only scalable method in literatures with state of art performance on the datasets used. Since we used exactly the same setup as the authors, in terms of training and test datasets, we are reporting their classification scores directly from [11].

Top-Down Logistic Regression (TD) This denotes Top-down logistic regression model, where we train a one-vs-rest multi-class classifier at each internal node. At testing time, the predictions are made starting from the root node. At each internal node, the highest scoring child node is selected, until we reach a leaf node.

For all the experiments, the regularization parameter is tuned using a validation set. The model is trained for a range of values 10^k with appropriate values for k selected depending on the dataset. Using the best parameter selected on validation set, we retrained the models on the entire training set and measured the performance on a held out test set. The source

code implementing the methods discussed in the current work is available on our website [3]. The experiments were performed on computers with Dell C8220 processors with dual Intel Xeon E5-2670 8 core CPUs and 64 GB memory.

4.5 Results

In this section, we present experimental comparisons of various cost sensitive learning strategies with other baseline methods. We provide separate comparisons of different cost based improvements on smaller datasets, and finally compare our best method with the competing methods. In the tables, statistically significant results for Micro-F_1 and Macro-F_1 [84] are marked with either † or ‡ which correspond to p-values < 0.05 and < 0.001 respectively.

In Table 4.2, we compare LR with various *hierarchical costs* defined in section 4.3.1. The results show a uniform improvement in all the metrics reported. There was a statistically significant improvement in Micro-F_1, especially for DMOZ Small, IPC and RCV1 datasets. Macro-F_1 scores were also improved, but due to the presence of only a small number of categories in CLEF and RCV1 datasets, statistical significance could not be established, except for ExTrD.

In Table 4.3 we compare the effect of introducing *imbalance costs*, discussed in section 4.3.1, on standard LR and hierarchical costs. In IMB+LR only the imbalance cost is used, in others, we use the product of costs derived from IMB strategy and the corresponding hierarchical costs. We also measured the significance of the improvement over the corresponding results from Table 4.2. Only for DMOZ Small, which has a large number of classes with few examples, imbalance costs further improve the results significantly for all the methods. On CLEF, IPC and RCV1, where majority of the classes have sufficient number of examples for training, the results did not improve significantly in most cases. Overall, the IMB+ExTrD method provides more robust improvements.

The final comparison of our best method (IMB+ExTrD, which we call *HierCost* in the following) against various baseline methods is presented in Table 4.4. The evaluations on

[3] `http://cs.gmu.edu/~mlbio/HierCost/`

Table 4.2: Performance comparison with hierarchical costs.

		Micro-F_1 (↑)	Macro-F_1 (↑)	hF_1 (↑)	TE (↓)
CLEF	LR	79.82	53.45	85.24	0.994
	TrD	80.02	55.51	**85.39**	0.984
	NCA	80.02	57.48	85.34	0.986
	ExTrD	**80.22**	**57.55**†	85.34	**0.982**
DMOZ SMALL	LR	46.39	30.20	67.00	3.569
	TrD	**47.52**‡	**31.37**‡	**68.26**	**3.449**
	NCA	47.36‡	31.20‡	68.12	3.460
	ExTrD	47.36‡	31.19‡	68.20	3.456
IPC	LR	55.04	48.99	72.82	1.974
	TrD	55.24‡	50.20‡	73.21	1.954
	NCA	**55.33**‡	**50.29**‡	**73.28**	**1.949**
	ExTrD	55.31‡	**50.29**‡	73.26	1.951
RCV1	LR	78.43	60.37	80.16	0.534
	TrD	79.46‡	60.61	82.83	0.451
	NCA	**79.74**‡	60.76	**83.11**	**0.442**
	ExTrD	79.33‡	**61.74**†	82.91	0.466

Table 4.3: Performance comparison with hierarchical and imbalance cost.

		Micro-F_1 (↑)	Macro-F_1 (↑)	hF_1 (↑)	TE (↓)
CLEF	IMB + LR	79.52	53.11	85.19	1.002
	IMB + TrD	79.92	52.84	85.59	0.978
	IMB + NCA	79.62	51.89	85.34	0.994
	IMB + ExTrD	**80.32**	**58.45**	**85.69**	**0.966**
DMOZ SMALL	IMB + LR	48.55‡	32.72‡	68.62	3.406
	IMB + TrD	**49.03**‡	33.21‡	69.41	3.334
	IMB + NCA	48.87‡	33.27‡	69.37	3.335
	IMB + ExTrD	**49.03**‡	**33.34**‡	**69.54**	**3.322**
IPC	IMB + LR	55.04	49.00	72.82	1.974
	IMB + TrD	55.60‡	**50.45**†	73.56	1.933
	IMB + NCA	55.33	50.29	73.28	1.949
	IMB + ExTrD	**55.67**‡	50.42	**73.58**	**1.931**
RCV1	IMB + LR	78.59‡	60.77	81.27	0.511
	IMB + TrD	**79.63**‡	61.04	**83.13**	**0.435**
	IMB + NCA	79.61	61.04	82.65	0.458
	IMB + ExTrD	79.22	**61.33**	82.89	0.469

Dmoz 2010 and Dmoz 2012 datasets are blind and the predictions have to be uploaded to LSHTC website in order to obtain the scores. For Dmoz 2012, Tree Errors are not available and for Dmoz 2010, the hF_1 are not available. For HRLR, we do not have access to the predictions, hence, we could only report the values for Micro-F_1 and Macro-F_1 scores from [11]. Statistical significance tests compare the results of *HierCost* with LR. These tests could not be performed on LSHTC datasets due to non-availability of the true labels on test sets. As seen in Table 4.4, *HierCost* improves upon the baseline LR results as well as the results reported in [11], in most cases, especially the Macro-F_1 scores. The results of *HierCost* are better on most measures. TD performs worst on average on set-based measures. In fact, only on Dmoz 2012 dataset, TD is competitive, on the rest, the results are much worse than the *flat* LR classifier and its hierarchical extensions. On hierarchical measure, however, TD outperformed *flat* classifiers on some datasets.

In Table 4.5, we report the run-times comparisons of TD, LR and *HierCost*. We trained

Table 4.4: Performance comparison of HierCost with other baseline methods.

		Micro-F_1 (↑)	Macro-F_1 (↑)	hF_1 (↑)	TE (↓)
CLEF	TD	73.06	34.47	79.32	1.366
	LR	79.82	53.45	85.24	0.994
	HRLR	80.12	55.83	-	-
	HierCost	**80.32**	**58.45**†	**85.69**	**0.966**
DMOZ SMALL	TD	40.90	24.15	**69.99**	**3.147**
	LR	46.39	30.20	67.00	3.569
	HRLR	45.11	28.48	-	-
	HierCost	**49.03**‡	**33.34**‡	69.54	3.322
IPC	TD	50.22	43.87	69.33	2.210
	LR	55.04	48.99	72.82	1.974
	HRLR	55.37	49.60	-	-
	HierCost	**55.67**‡	**50.42**†	**73.58**	**1.931**
RCV1	TD	77.85	57.80	**88.78**	0.524
	LR	78.43	60.37	80.16	0.534
	HRLR	**81.23**	55.81	-	-
	HierCost	79.22‡	**61.33**	82.89	**0.469**
DMOZ 2010	TD	38.86	26.29	-	3.867
	LR	45.17	30.98	-	3.400
	HRLR	45.84	**32.42**	-	-
	HierCost	**45.87**	32.41	-	**3.321**
DMOZ 2012	TD	51.65	**30.48**	73.33	-
	LR	51.72	27.19	72.53	-
	HRLR	53.18	20.04	-	-
	HierCost	**53.36**	28.47	**73.79**	-

Table 4.5: Total training times (mins).

	TD-LR	LR	HierCost
CLEF	<1	<1	<1
DMOZ SMALL	4	41	40
IPC	27	643	453
RCV1	20	29	48
DMOZ 2010	196	15191	20174
DMOZ 2012	384	46044	50253

the models in parallel for different classes and computed the sum of run-times for each training instance. In theory, the run-times of LR and *HierCost* should be equivalent, because they solve similar optimization problems. However, minor variations in the run-times were observed because of the variations in optimal regularization penalties, which influences the convergence of the optimization algorithm. The run-times of flat methods were significantly worse than TD, which is efficient in terms of training, but at considerable loss in classification performance. Although, we do not measure the training times of HRLR, based on the experience from a similar problem [21], the recursive model take between 3-10 iterations for convergence. In each iteration, the models for all the terminal labels need to be trained hence each iteration is about as expensive as a single run of LR. In addition, the distributed recursive models require communication between the training machines which incurs an additional overhead.

4.6 Summary

In this chapter, we have argued that the methods that extend flat classification using hierarchical regularization, can be viewed in a complementary way as weighting the losses on the negative examples depending on dissimilarity between the positive and negative classes. The approach proposed in the current work incorporates this insight directly into the loss function by scaling the loss function according to the dissimilarity between the classes with

respect to the hierarchy, thus obviating the need for recursive regularization and iterative model training. At the same time, this approach also makes parallelization trivial. Our method also mitigates the adverse effects of imbalance in the training data by up-weighting the loss for examples from smaller classes, thus, significantly improving their classification results. Our experimental results show that the proposed method is able to efficiently incorporate hierarchical information by transforming the hierarchical classification problem into an example based cost sensitive classification problem. In future work, we would like to evaluate the benefits of cost sensitive classification using large margin classifiers such as support vector machines.

Chapter 5: Regularized Multi-task Learning for Classification with Dual Hierarchies

5.1 Introduction

Owing to the popularity of hierarchies for categorizing and labeling information in many databases, often more than one representations of the hierarchies can be encountered in many applications. Although these hierarchies differ in certain aspects such as the structure of the hierarchy, the number of entities classified and the meaning of the structural elements classified, the unifying aspect of these hierarchies is that they represent the same or similar entities. Several methods have tried to leverage the hierarchical information to improve classification from a single hierarchy, but one of the novel problems that the work presented in this chapter addresses is combining different hierarchies in learning process in order to take advantage of the additional perspectives presented by multiple schemas.

In this work, we have used multi-task learning (MTL) to combine the knowledge present across the two hierarchical protein classification databases. The motivation for our approach stems from the fact that different annotation databases strive to achieve the same objective (i.e., classify protein structures), but differ in terms of curation process, coverage and annotation errors. Further, the use of MTL approach allows us to capture the hierarchical structure that is prevalent across these databases. Within the hierarchies we may have several classes that have few training examples. As such, using MTL for training prediction models for related task assists in improving the generalization performance of individual tasks when the data is scarce [6].

The key contribution in this work is to combine the problem of hierarchical classification in two separate but related class hierarchies. Specifically, we explored different MTL formulations to leverage the combined knowledge present in SCOP and CATH hierarchical

databases to improve the classification performance. Each labeled class in SCOP and CATH defines a classification task. The hierarchical structure of the labeled classes explicitly defines the relationships between the classes within each hierarchy. These class relationships can be utilized in the MTL formulations which take task relationships into account. However, the relationships between one class from SCOP and the other from CATH are non-trivial, as there is no direct correspondence between the labeled classes defined by SCOP and CATH.

We performed a comprehensive set of experiments to assess the accuracy of classification when using the MTL approaches in comparison to single task learning (STL). We observed an improvement in the classification performance from the combined use of SCOP and CATH in protein structure classification. We found that using the graph-based structured MTL that explicitly defines the relationships between different tasks and the trace-based MTL formulation that seeks to find a low-dimensional common subspace across the tasks are suitable for our purpose. Specifically, we observed approximately $4-5\%$ improvement for the multiple hierarchical based MTL learning in comparison to the STL approach. Our current study suggests that on the consistent set of protein domains defined by both SCOP and CATH, MTL methods which combined both the hierarchical schemes had a significant advantage over MTL methods which were trained using only one hierarchy.

Notations We use T for the number of tasks and d for the number of input dimensions. n_t denotes the number of examples in task t. $W \in \mathbb{R}^{d \times T}$ denotes the matrix of task weights, where each column $w_t \in \mathbb{R}^d$ is the weight vector for tasks $t \in [1:T]$, the rows of W are denoted by $\tilde{w}_j \in \mathbb{R}^T$ for $j \in [1:d]$. The training examples for task t are denoted by $\{(x_{it}, y_{it})\}_{i=1}^{n_t}$.

5.2 Structural Classification of Proteins

Proteins are large biological macromolecules which perform a vast array of biological functions in all living organisms. Structurally they consist of one or more linear chains of

building blocks known as *amino acids*. Although only about 20 amino acids form the protein alphabet, proteins assume a vast range of three dimensional conformations which in turn determines their various biological roles. Since understanding their structure and the resultant functional roles is a crucial aspect for understanding biological organisms, several structural and functional databases have been created to organize proteins.

Two such early efforts towards hierarchical classification of protein structures are Structural Classification of Proteins (SCOP) [17] and Class, Architecture, Topology, and Homologous Superfamily (CATH) [85]. SCOP follows a predominantly manual process, relying on the expertise of its curators. CATH relies on automated approaches to a greater degree than SCOP and revises its database more frequently. SCOP as well as CATH classify entities known as *protein domains*, which can be considered as functional units of proteins, into four major levels. Protein domains may be viewed as subsequences (not necessarily contiguous) of proteins. A protein may be composed of one or more domains. Thus, each classification system assigns four labels to each of the domains it classifies. The levels defined by SCOP, listed in top to bottom order are Class, Fold, Superfamily and Family. The corresponding levels defined by CATH are Class, Architecture, Topology and Homologous Superfamily. These gold-standard structural classifications are performed through a laborious and expensive process of first determining the three-dimensional structure of proteins, and curation into a hierarchical database using manual /semi-automated schemes. The rapid growth in the number of sequenced proteins in recent years has created a need for automated methods to accurately classify protein sequences into structural classes. Several computational methods have been proposed to predict the structural classes from sequence information using a range of statistical methods [86–90].

SCOP and CATH have been extensively compared in several previous studies [91–93]. Despite their similar purpose, several discrepancies between these two databases have been reported [91, 93]. Inconsistencies primarily arise due to the differences in domain definition and disagreement in the classification of some domains.

Predictive methods developed in recent years have primarily focused on classifying sequences with respect to SCOP. Consequently, remote homology detection and fold recognition, where the objectives are to detect superfamily and fold given the protein sequence, have received considerable attention in the research community. Early predictive methods used sequence similarity as a surrogate for structural similarity and utilized sequence comparison methods such as SW-alignment, BLAST and PSI-BLAST [94, 95]. The later methods took a generative approach to model related sequences. The discriminative methods, primarily Support Vector Machines, which followed generative methods further improved classification performance. Much attention was then given to engineering better kernels [86, 89, 90, 96–98] to capture biologically meaningful similarities between sequences.

Protein structure classification is inherently a multi-class problem where the label of a protein domain is defined for classifying a particular level in the hierarchy (fold, superfamily etc.). Most early methods approached this problem by dividing the multi-class problem into several one-vs-rest or one-vs-one problems. If all the levels in the hierarchy are considered then the problem becomes a structured-output classification task [99] with interdependencies between the output labels [100, 101]. These methods are able to utilize the correlations between different labels within the hierarchy.

Our current work takes a slightly different approach. Here the focus is on leveraging combined knowledge across different hierarchical classification systems which enables us to learn better classifiers for all the tasks. To our knowledge this approach toward integrated learning and cross hierarchy transfer for protein structure classification has never been explored before.

5.3 Methods

Given two hierarchical sources of annotated protein domains (e.g., SCOP and CATH), our primary objective is to train a supervised learning model that can accurately classify new instance into classes within these hierarchies. In this study, we use MTL approaches to take advantage of the fact that the classes within the hierarchy have explicit relationships between

them and the two hierarchical source databases have implicit relationships across them due to similar objectives. We assume that each of the classes (nodes) within the hierarchies is associated with a binary classification problem, referred to as a task. Combining these tasks during learning should help improve the prediction performance. We assume that there are T tasks across the two hierarchies with independent training instances given by $\{(x_{it}, y_{it}) : i = 1 \dots n_t\}$. For a task t, we seek to learn a linear discriminant function per task given by $sign(w_t^T x_{it})$.

We use the logistic loss function given by,

$$\mathcal{L}\left(y, w^T x\right) = \log\left[1 + \exp^{-y\left(w^T x\right)}\right] \tag{5.1}$$

The regularized objective is minimized with respect to the classification constraints given by (5.2).

$$y_{it}\left(w_t^T x_{it}\right) \geq 0 \qquad \forall t \in \{1, \dots, T\}, \forall i \in n_t \tag{5.2}$$

The best loss function for binary classification problems is the natural 0-1 loss function. It is, however, non-convex and non-smooth which makes it intractable for optimization problems. Logistic loss is a smooth, convex loss function which is suited for classification. Other loss functions, such as hinge-loss might be better in practice [102] but may require specialized solutions. Regularization acts upon the weight parameters W only, and the bias parameters c are typically not regularized and hence not included in the joint regularization. In the following section we provide an overview of the MTL methods used in this work.

5.3.1 MTL Methods

Sparse learning through joint feature selection

It is well known that the l_1-norm regularization [38] encourages sparsity in STL models. This idea has been extended to multi-task learning [8, 28, 103] to select sparse features across all the tasks. In these models, it is assumed that across all the tasks only a subset of the feature space is critical for classification. The objective function for sparse learning in MTL can be written as,

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}\left(y_{it}, w_t^T x_{it}\right) + \lambda \left\|W\right\|_{2,1} \tag{5.3}$$

where $\left\|W\right\|_{2,1} \equiv \sum_{t=1}^{T} \left\|\tilde{w}_l\right\|_2$ and $\tilde{w}_l \in \mathbb{R}^D$ denotes the l^{th} row of W associated with the input feature l. The $l_{2,1}$-norm here is essentially the l_1-norm of the vector of l_2-norm over the weights associated with a particular input dimension. However, the above optimization problem is difficult to solve due to the non-smoothness of $l_{2,1}$-norm. Obozinski and Taskar [103] proposed a solution based on block-wise boosting scheme, and Liu et al. [28] proposed two equivalent formulations for the problem that can be efficiently solved. Argyriou et al. [8] formulated the problem of learning sparse linear features as a convex optimization problem. In our current work we have used the sparse feature selection method described by Argyriou et al. [8]. However, the presence of sparse feature space assumed by this model is too strong for the application studied in the current work and consequently does not perform well. We refer to this method as *Sparse* MTL.

Graph Regularization

As discussed above, many MTL formulations assume uniform correlations between all the tasks. Evgeniou and Pontil [10] extended SVM to MTL by augmenting the SVM objective with an additional term to control the inter-task regularization. In a more general setting,

the objective function can be written as,

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}\left(y_{it}, w_t^T x_{it}\right) + \lambda_1 \sum_{t=1}^{T} \|w_t\|_2^2 + \lambda_2 \sum_{t=1}^{T} \left\| w_t - \frac{1}{T} \sum_{s=1}^{T} w_s \right\|_2^2 \tag{5.4}$$

The first regularization term controls the magnitude of the parameters for each task and the second regularization term imposes the constraint that the parameters for each task should be close to the average parameters of all the tasks. This assumption is too strong in many cases where some tasks may be more closely related than others.

When relationships between tasks are known a-priori, inter-task regularization can be controlled such that only the related tasks influence each other. Graph-based MTL formulations [13, 35, 104] are able to exploit such relationships. The relationships between different tasks can be encoded as a set of edges $\mathcal{E} = (e_1, e_2, \ldots, e_m)$, where each edge $e_i \equiv (p, q)$ connects a pair of tasks indexed by p and q. The graph regularized MTL utilizes the edge relationships in the following manner,

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}\left(y_{it}, w_t^T x_{it}\right) + \lambda_1 \sum_{t=1}^{T} \|w_t\|_2^2 + \lambda_2 \sum_{(p,q)\in\mathcal{E}} \|w_p - w_q\|_2^2 \tag{5.5}$$

Unlike the previous formulation in (5.4), the regularization term in (5.5) minimizes the difference between all pairs of related tasks only and does not bias the learned parameters towards the average of all tasks. Therefore, only the parameter vectors of tasks connected by an edge are forced to be similar to each other.

Rank minimization using Trace norm

The problem of rank minimization arises in many optimization problems in machine learning, image compression and automatic control. In the context of multi-task learning minimizing the rank of W forces it to share a low-dimensional subspace, therefore inducing correlations

between the tasks. The objective function with rank minimization as regularization can be written as,

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}\left(y_{it}, w_t^T x_{it}\right) + \lambda \; rank(W) \tag{5.6}$$

However, the rank minimization problem is known to be NP-hard [72]. Rank minimization objective can be approximated by using the trace norm (nuclear norm) which gives a convex solution [105]. The regularized objective function with trace norm can therefore be written as,

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}\left(y_{it}, w_t^T x_{it}\right) + \lambda \left\|W\right\|_* \tag{5.7}$$

This formulation has been extensively studied in the context of multi-task learning [43, 106, 107]. In the current work we have used the solution described by Ji et al. [43].

Graph and Trace joint norm minimization

In addition to the previous formulations, we implemented a novel multi-task learning formulation combining the Graph and Trace norm minimization which uses the following objective function

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}\left(y_{it}, w_t^T x_{it}\right) + \lambda_1 \sum_{(p,q)\in\mathcal{E}} \left\|w_p - w_q\right\|_2^2 + \lambda_2 \left\|W\right\|_* \tag{5.8}$$

In all the models described above, an l_2-norm is added to the regularization objective however it is only explicitly represented for Graph formulation. The l_2-norm essentially

controls the magnitude of the weight vectors.

We used the publicly available MALSAR toolkit [108] to implement all methods discussed above. The optimization algorithms within MALSAR are implemented using the accelerated gradient methods [109]. This optimization technique is shown to be faster than traditional gradient descent methods, and involves finding a proximal operator for a non-smooth regularization term [108].

5.3.2 Protein Structure Classification using MTL

Both SCOP and CATH classify protein domains hierarchically. As noted previously, the levels of SCOP from top to bottom are Class, Fold, Superfamily, and Family, the levels of CATH are Class, Architecture, Topology and Homologous Superfamily. The learning problem is to build classifiers for each level of the hierarchy, which in turn has multiple classes. Therefore this problem is hierarchical multi-class classification problem. We transform this into one-vs-rest type binary problems for each node in the hierarchy. In this work, we only focus on the results of classification of Superfamily and Topology levels which we call L3 levels from SCOP and CATH respectively. In our main experiments, we also include the L4 level tasks, which correspond to Family (SCOP) and Homologous Superfamily (CATH), as auxiliary tasks. In an additional set of experiments, we also included the Fold (SCOP) level nodes.

5.3.3 Dataset Pre-processing

For this study, we used SCOP 1.75 and CATH 3.4 databases, both of which provide the domain definition, domain classification at each taxonomic level and the taxonomy as a hierarchical structure for the protein domains classified in their databases.

To apply MTL across tasks from different hierarchies we extracted a set of domains which are defined consistently in both the hierarchies. Determination of domains in a protein is a non-trivial process, and SCOP and CATH differ significantly in defining domains for multi-domain proteins. The inconsistencies in these two schemes may lead to errors in the

models learned from these classification systems [91]. To reduce the inconsistencies, we extracted domains which are defined similarly in both the schemes, which we call *consistent domains*. As very few domain definitions agree exactly in both SCOP and CATH, we used an approximate similarity threshold of 0.8 to define similar domains. The domains derived from the same protein sequence are considered to be similar if $\geq 80\%$ of the amino-acids in their sequences overlap. Out of 96,885 domains classified by SCOP and 149,306 domains classified by CATH, we could extract 61,931 domains where the domain definitions agreed according to our similarity criterion.

Many pairs of the sequences classified in SCOP are identical or nearly identical. This is also the case with CATH. Before applying classification methods, it is a good practice to eliminate redundant sequences by clustering sequences and retaining only the cluster representatives. Instead of performing the clustering on the extracted domains, we have used the sequence subsets at different similarity thresholds, determined by sequence alignment, already provided by both SCOP and CATH databases. We used the 95% sequences similarity clusters to reduce the redundancies and mapped the 61,931 domains to their respective cluster representatives in both SCOP and CATH, which produced 16,712 SCOP domains and 21,325 CATH domains.

All the nodes in the hierarchy can not be used for training and evaluating classifiers because only a few nodes have sufficient number of examples for a meaningful generation of testing and training sets as described later in section 5.4.2. To create the dataset, we first eliminate all the L4 nodes with < 5 instances. Then we eliminate the L3 nodes with ≤ 2 children because we partition the training and test instances by L4 nodes. This final filtering process resulted in 5,587 SCOP domains and 9,642 CATH domains. The statistics of the pruned dataset are provided in Table 5.1.

Once the tasks for each relevant node are defined, the application of MTL to the set of tasks is straightforward. As described in previous section, our main goal is to model the tasks corresponding to superfamilies and topologies and families and homologous superfamilies are included as auxiliary tasks. Therefore, the number of main tasks in the MTL formulations

correspond to the number of superfamilies and topologies in Table 5.1. To repeat each experiment multiple times, we created multiple random splits of our data using the strategy described in section 5.4.2. The number of auxiliary sub-tasks corresponding to the lowest levels i.e. families and homologous superfamilies vary according to the split.

We also extracted the task relationships for Graph-regularized MTL. These relationships are extracted from explicit hierarchical structure of each taxonomy as well as the knowledge of common domains across different hierarchies. The process of relationship extraction is detailed in the following section.

Finally, the raw protein sequences must be represented as feature vectors of fixed length, so that we can use standard machine learning methods working in vector-space model. Various methods have been proposed in literature for extracting meaningful features from biological sequences [86, 87, 97, 110]. For majority of our analysis we use spectrum kernel features [88], which uses contiguous subsequences of some fixed length k, also known as k-mers. We performed most of our analysis using $k = 3$, which results in 8000 features, but we also experimented with larger values of k (4, 5, and 6), which resulted in poorer performance. We performed additional experiments using more expressive set of features known as Sparse Spatial Samples Kernel (SSSK)[111], which have been experimentally shown to perform well. The features in SSSK are t contiguous words each of length k separated by up to $d-1$ intervening characters. Specifically, we used two sets of SSSK features SSSK($t = 2, k = 1, d = 5$) also known as (1,5)-double in the original reference and SSSK($t = 3, k = 1, d = 3$) also known as (1,3)-triple. We applied Kernel Principal Component analysis (KPCA) [112], to (1,3)-triple features to reduce dimensionality from 72000 features to 8000 features. 8000 features were chosen to capture $> 95\%$ of the variance of the centered kernel matrix.

5.3.4 Extracting Task Relationships

In order to utilize the relationships between different tasks in graph regularized formulation, we define edges between different nodes in SCOP and CATH. Edges can be categorized based on whether they span only one hierarchy or both the hierarchies. We refer to the

Table 5.1: Dataset statistics.

SCOP		CATH	
instances	5587	instances	9642
families	318	homologous superfamilies	259
superfamilies	102	topologies	51
folds	82	architectures	17
classes	4	classes	3

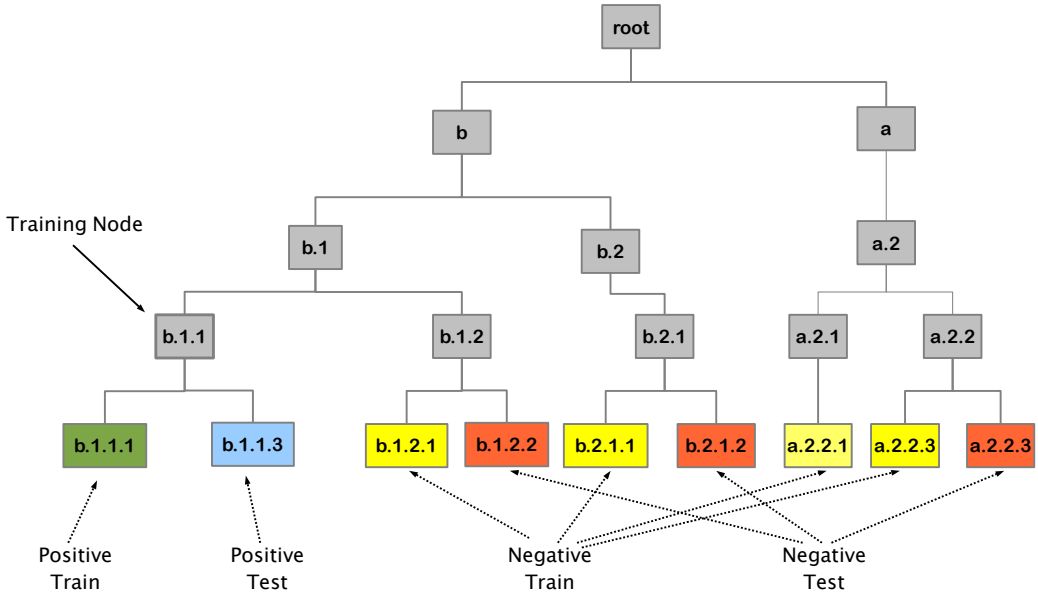


Figure 5.1: Partial hierarchy illustrating training and test partitions for SCOP superfamily *b.1.1*

Table 5.2: Counts of each type of relationship used in Graph MTL

	SCOP	CATH
Parent Links	318	259
Sibling Links	1112	3808
Cross Links	274	274

edges that connect nodes in SCOP and CATH as *cross links*. The links connecting SCOP nodes to SCOP nodes or CATH nodes to CATH nodes, are referred to as *within links*. The *within links* can be further categorized based on the type of nodes they connect. Below, we list the three types of links use in the current work.

Parent Links These edges join a child node to its parent node *i.e.,* superfamily-family (SCOP), topology-homologous superfamily (CATH). These are explicitly defined by the hierarchical structure of SCOP and CATH.

Sibling Links These edges join siblings *i.e.,* links between a node and other children of its parent. An edge between a node and itself is not defined. These edges are defined between superfamily-superfamily (SCOP), topology-topology (CATH), family-family (SCOP), and homologous superfamily-homologous superfamily (CATH).

Cross Links These edges span both the hierarchies and connect nodes at the same level for both the hierarchies *i.e.,* superfamily (SCOP) - topology (CATH) and family (SCOP) - homologous superfamily (CATH). Edges are not defined between superfamily (SCOP - homologous superfamily (CATH) and family (SCOP) - topology (CATH). These edges are derived using *consistent domains.* If a domain is classified by two nodes we add an edge between them. Specifically, we add an edge between a SCOP node N_s and CATH node N_c , if there are are protein domains $p \in N_s$ and $q \in N_c$ such that p and q are similar as described previously.

In Table 5.2, we summarize the number of links of each type extracted from the dataset.

5.4 Experimental Setup

5.4.1 Models Evaluated

To test the benefit of combining various tasks, we evaluated different MTL formulations using the following partitions of the datasets.

Single Task Learning (STL)

Each task is trained separately. We performed STL experiments using the elastic net penalty which uses both l_1 and l_2 norms in regularization.

Single Hierarchy MTL (SHMTL)

Only the tasks belonging to the same hierarchy are learned together. These experiments are conducted for Sparse, Graph and Trace regularization. Graph regularization for SHMTL uses the parent links and sibling links.

Multiple Hierarchy MTL (MHMTL)

We grouped the tasks belonging to both the hierarchies and trained them together. These experiments were conducted for Sparse, Graph and Trace regularization. Graph regularization for MHMTL uses the cross links between the two hierarchies in addition to the sibling and parent links.

5.4.2 Validation Protocols

Partitioning the data into training and test sets is one of the important issues in the application of prediction methods to protein structure classification problem. Random partitioning of the training examples without taking the class labels into accounts may lead to over estimation of the accuracy or equivalently, under estimation of the test error. A common strategy employed to overcome this problem divides the training and test examples by L4

nodes (Family in SCOP and Homologous Superfamily in CATH). This ensures that sufficiently similar examples do not appear in both training and test examples, giving the classifiers unfair advantage.

This method of partitioning the examples by L4 nodes is illustrated in Fig. 5.1, using the partial hierarchy, for the partitioning of training and test examples for the task corresponding to SCOP node *b.1.1*. Considering the task of training classifier for the node *b.1.1*, we assign the examples belonging to some of its children, *b.1.1.1* in the figure, to positive train and the remaining, *b.1.1.3* in the figure, to positive test. Remaining L4 nodes are randomly assigned to negative train and negative test. By assigning a node to a particular set, we mean allocating the instances classified under that node, to the aforementioned set.

Similarly, the test and train datasets are constructed for the all L3 nodes in both SCOP and CATH.

To determine the best parameters, we performed grid search on a single random training and test partition with the parameters approximately in the range $\{10^{-1}, 10^{-2}, 10^{-3}, 0\}$. With the best parameters selected for each method, we repeated the experiments with 30 random training and test partitions.

Further, to test whether there was any benefit of adding the family (SCOP) and homologous superfamily (CATH) levels, we removed the tasks belonging to these levels in one of our datasets (Spectrum $k = 3$) and tested them using the same setup. We will call this partition L3-only, which stands for level 3 in the hierarchies as opposed to L3+L4 which includes both level 3 and level 4 in the hierarchies.

Accuracy of classification is not a good measure of performance when the class distribution is highly unbalanced. In these cases Area Under Curve (AUC) of Receiver Operating Characteristics curve (ROC) [113] gives a better indication of classification performance. As stated above we repeated the training and testing with 30 randomly derived partitions. For each task we collected the AUC scores from each run and calculated their mean and standard error. The AUC scores averaged across all the tasks, separated by the two hierarchies, are reported in Table 5.3. The standard error value for each score is reported inside parenthesis,

Table 5.3: AUC scores using L3 along with auxiliary L4 tasks in training

Dataset	Method	SCOP score	SCOP stderr	CATH score	CATH stderr
Spectrum($k = 3$) # features = 8000	Sparse-MHMTL	0.790	0.001	0.723	0.002
	Sparse-SHMTL	0.792	0.001	0.724	0.003
	Sparse-STL	0.791	0.001	0.725	0.003
	Graph-MHMTL	0.839	0.002	0.782	0.002
	Graph-SHMTL	0.795	0.001	0.746	0.004
	Trace-MHMTL	0.851	0.004	0.764	0.003
	Trace-SHMTL	0.801	0.002	0.747	0.003
	Graph-Trace-MHMTL	0.853	0.002	0.782	0.003
SSSK-double($k = 1, d = 5$) # features = 2000	Sparse-MHMTL	0.831	0.001	0.775	0.002
	Sparse-SHMTL	0.824	0.002	0.752	0.003
	Sparse-STL	0.832	0.001	0.769	0.002
	Graph-MHMTL	0.853	0.001	0.787	0.003
	Graph-SHMTL	0.833	0.001	0.783	0.003
	Trace-MHMTL	0.841	0.005	0.779	0.004
	Trace-SHMTL	0.831	0.001	0.773	0.002
	Graph-Trace-MHMTL	0.841	0.005	0.781	0.005
SSSK-triple($k = 1, d = 3$) with KPCA # features = 8000	Sparse-MHMTL	0.840	0.001	0.766	0.003
	Sparse-SHMTL	0.838	0.002	0.762	0.003
	Sparse-STL	0.840	0.001	0.764	0.002
	Graph-MHMTL	0.871	0.004	0.810	0.003
	Graph-SHMTL	0.842	0.001	0.781	0.003
	Trace-MHMTL	0.883	0.002	0.785	0.003
	Trace-SHMTL	0.839	0.001	0.761	0.003
	Graph-Trace-MHMTL	0.871	0.004	0.810	0.003

Table 5.4: AUC scores using only L3 tasks

Dataset	Method	SCOP score	SCOP stderr	CATH score	CATH stderr
Spectrum($k = 3$) # features = 8000	Sparse-MHMTL	0.789	0.001	0.723	0.002
	Sparse-SHMTL	0.792	0.002	0.722	0.002
	Sparse-STL	0.792	0.002	0.725	0.003
	Graph-MHMTL	0.844	0.001	0.777	0.004
	Graph-SHMTL	0.797	0.002	0.743	0.005
	Trace-MHMTL	0.840	0.004	0.766	0.001
	Trace-SHMTL	0.792	0.003	0.746	0.003
	Graph-Trace-MHMTL	0.845	0.002	0.776	0.003

next to the AUC score.

5.5 Results

5.5.1 Performance of MHMTL

We performed a comprehensive set of experiments evaluating the performance of different MTL approaches under various settings. Table 5.3 shows the average AUC scores for the different MTL models evaluated across the SCOP and CATH nodes for the datasets where both the L3 tasks as well as the auxiliary L4 tasks are used. Similiar results for the dataset which uses only L3 tasks are reported in Table 5.4 for the dataset using Spectrum($k = 3$) features. Detailed results with AUC scores for each task have been made available on the supplementary website [1]. The following observations can be made from the results.

Sparse MTL based on $l_{2,1}$-norm regularization does not improve the results in SHMTL and MHMTL compared to STL. The sparse formulation used here, attempts to extract sparse features across all the tasks. It assumes a uniform relationship between all the tasks. However, it is unlikely that any subset for the features derived from the amino-acid sequences are more important for the classification. During the parameter optimization with

[1] `http://www.cs.gmu.edu/~mlbio/supplements/mhmtl/`

$l_{2,1}$ regularization we found that better results were produced when we lowered the value of the parameter which controls sparsity. Therefore, the best performing models are not sparse. However, these experiments act as a control set to verify that no bias is being introduced by the selection of tasks within the datasets.

Fig. 5.2 compares the AUC scores of SHMTL and STL methods. As can be seen from Fig. 5.2 and Table 5.3, SHMTL with the Trace and Graph based regularization performs better than the STL. We notice that improvement for CATH nodes is significantly greater ($\approx 2-3\%$) than that for SCOP nodes ($\leq 1\%$) which can also be noticed in Fig. 5.2. We also see that the Trace regularization performs slightly better in comparison to the Graph regularization for SHMTL with the L3+L4 dataset. This difference is also evident from Fig. 5.4(a) which compares the improvement in AUC scores in SHMTL over STL for Graph and Trace regularization. We also notice that for some CATH nodes the improvement is significantly greater for Trace than for Graph.

We see an improvement of approximately 5% in the AUC scores for SCOP using both the Trace and Graph regularizations for the MHMTL in comparison to SHMTL. For the CATH dataset we see an improvement of $2-3\%$ (refer to L3+L4 dataset results in Table 5.3). The percentage improvement observed for MHMTL in comparison to STL is $5-6\%$ and $4-5\%$ for the SCOP and CATH data, respectively. Fig. 5.3 compares the AUC scores for MHMTL and SHMTL for both Trace and Graph formulations. This suggests that there is an advantage in combined MTL over both the hierarchies.

Fig. 5.4(b) compares the AUC improvement from MHMTL over SHMTL for the Graph and Trace formulations. There is a linear correlation in the improvements in AUC scores from Graph and Trace formulations in MHMTL when compared to SHMTL, *i.e.,* the same L3 nodes are improving and by roughly the same amount using two independent methods. Therefore, it is very unlikely that the improvement is random. Interestingly, as can be seen in Fig. 5.4(b) CATH nodes benefit more from Graph than from Trace. Whereas in the AUC improvement of SHMTL over STL case shown in Fig. 5.4(a), Trace formulation helps CATH nodes more than Graph formulation.

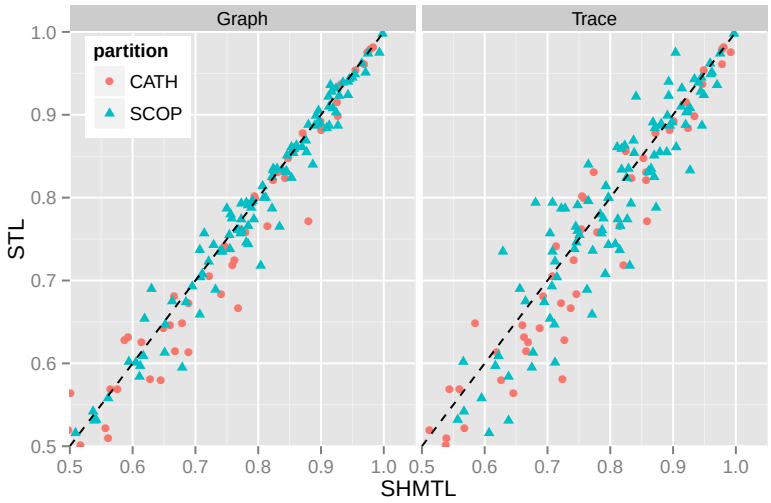


Figure 5.2: AUC SHMTL vs STL for L3 tasks

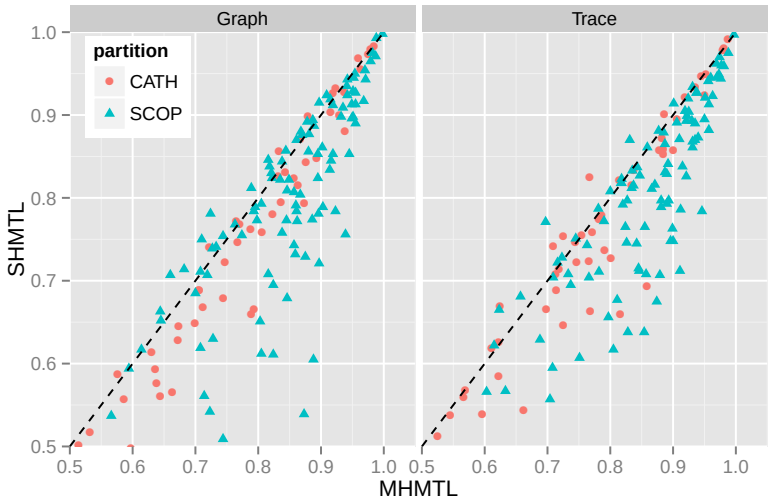
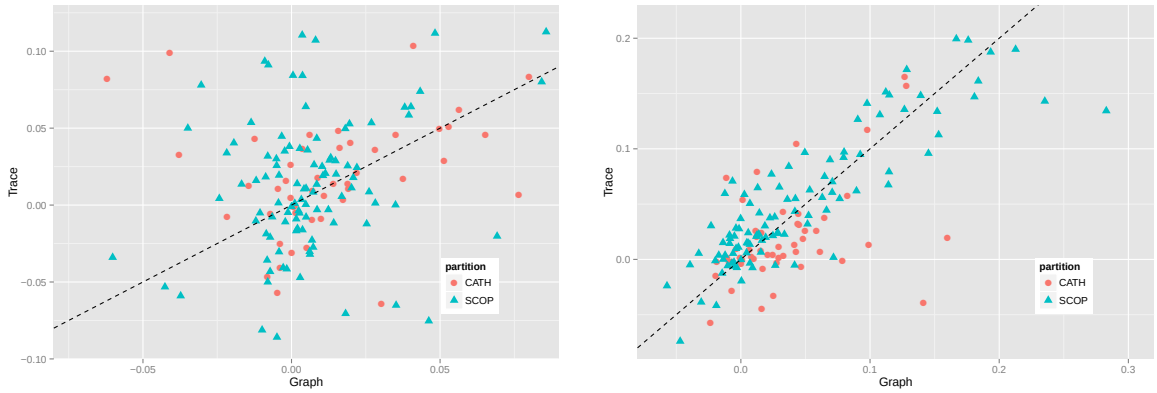


Figure 5.3: AUC MHMTL vs SHMTL for L3 tasks



(a) SHMTL - STL

(b) MHMTL - SHMTL

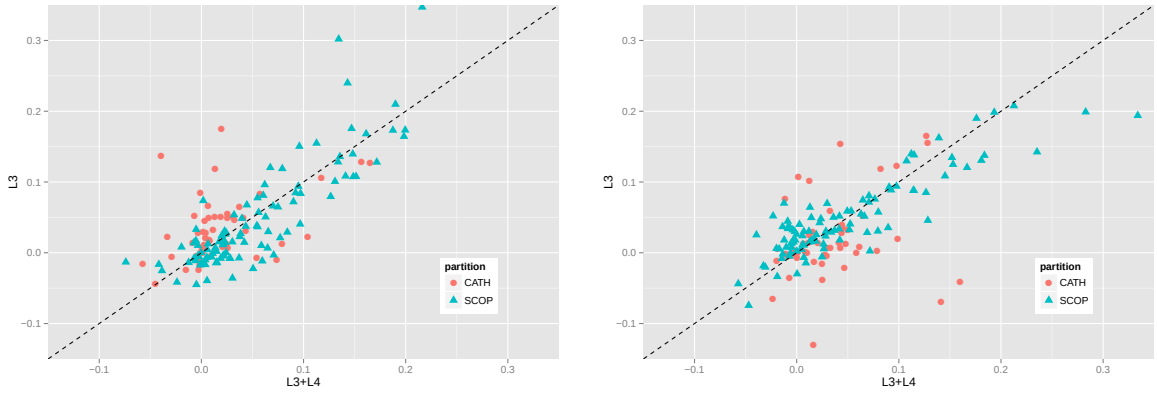
Figure 5.4: AUC improvement for Graph vs Trace

For our datasets, we observed that combining the Graph and Trace norm in the objective function did not improve the classification performance over the individual norms used separately. From the parameter tuning, we noticed that one of the norms

Although, on an average, we see a performance improvement in most cases, on certain tasks the combined SCOP-CATH setting under-performs. We conjecture that MTL could be introducing negative transfer in these cases whereby the generalization performance is negatively affected by incorporating MTL bias.

5.5.2 L3-only versus L3+L4

When we remove the L4 nodes from training the results are not significantly affected. Fig. 5.5 compares the improvement in AUC scores across the L3-only and L3+L4 datasets (difference of MHMTL over SHMTL given by "MHMTL - SHMTL") for the Graph and Trace formulations. The performance of Graph is very similar in both the cases, but Trace performed slightly better, especially for SCOP nodes with L4 nodes included in training. Removing L4 nodes significantly reduces the training times, therefore the cost to benefit ratio for including L4 nodes is very high.



(a) Graph (b) Trace

Figure 5.5: AUC improvement (MHMTL - SHMTL) for different set of nodes in training L3 vs L3+L4

5.5.3 Graph Regularized MTL using Inner Node Mappings

In the previous set of experiments using Graph MHMTL, we created the cross links between the same level of the hierarchy. However, the nodes at different levels between SCOP and CATH are not necessarily equivalent to each other . In their comparison of SCOP and CATH, Casaba et al. [91] consider the following levels to be mappable to each other - SCOP superfamily/family map to CATH homologous superfamily and SCOP fold map to CATH topology and SCOP class to CATH class. Their evaluation reveals that this mapping of levels is not strict. To evaluate the performance of Graph MHMTL with respect to Graph SHMTL when the cross links between SCOP and CATH are defined by node mappings based on shared protein domains we created another dataset with tasks corresponding to fold, superfamily and family nodes from SCOP and topology and homologous superfamily nodes from CATH.

To map the nodes, we computed an F1-score metric between pair of nodes similar to the mapping described by Casaba et al. [91]. This mapping is not symmetric, hence, we consider two maps, one from SCOP→ CATH and similarly from CATH → SCOP. To map the nodes we use the set of similar domains. Let S_1 represent the set of domains belonging

a SCOP node N_1 and S_2 be the set of domains belonging to a CATH node N_2. To compute the F1-score of mapping from $N_1 \rightarrow N_2$, we first determine the set of SCOP domains similar to the CATH domains in S_2, and denote this set by $\bar{S}_2$.

The F1-score of mapping is calculated as

$$F1 - score = \frac{2 * sensitivity * specificity}{sensitivity + specificity} \tag{5.9}$$

Where $sensitivity = \left|S_1 \cap \bar{S}_2\right| / |S_1|$ and $specificity = \left|S_1 \cap \bar{S}_2\right| / \left|S_1 \cup \bar{S}_2\right|$. The mapping of nodes from CATH to SCOP is similarly calculated. Finally, we created an edge between the nodes where the F1-score of a pair of nodes in the mappings in both the directions is ≥ 0.5. We extracted a total of 154 such cross links.

The experiments with this dataset did not show any statistically significant improvement compared to the previous dataset where the edges were defined between a pair of nodes if they shared even a single domain. The detailed results are available on the supplementary website.

5.5.4 Graph-based Link Analysis

In the context of Graph based MTL, we also studied the effect of sibling relationships on AUC improvement of SHMTL over STL (SHMTL - STL) and the effect of number of cross links on AUC improvement of MHMTL over STL (MHMTL - STL). Fig. 5.6 and Fig. 5.7 depict the improvements with respect to the number of links. In Fig. 5.6-Graph, which shows the improvements for Graph formulation, we see a uniform improvement in all the nodes with non-zero cross links. CATH seems to benefit more from the presence of cross edge and Graph formulation which can also be seen from Fig. 5.4(b). This suggests that Graph formulation is able to leverage the relatedness between tasks across the hierarchies. The Trace formulation does not take into account these relationship, Fig. 5.6-Trace does not seem to suggest this pattern.

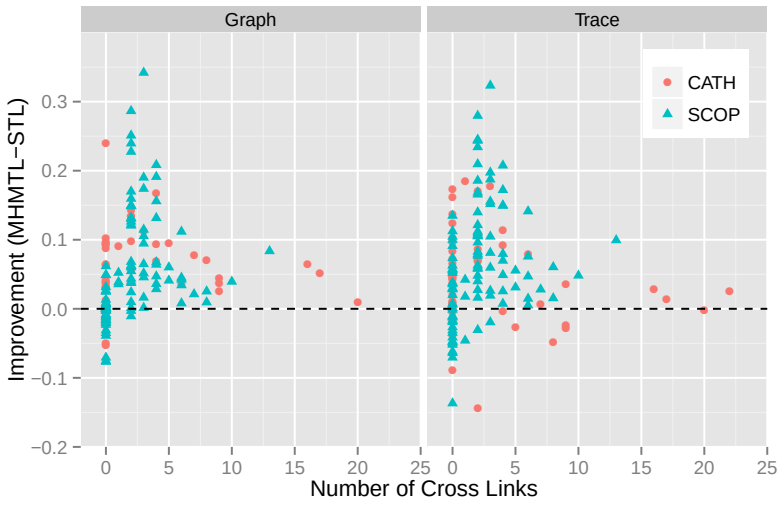


Figure 5.6: AUC improvement (MHMTL - STL) with number of Cross Links

Fig. 5.7 shows the improvement in AUC for SHMTL over STL (SHMTL - STL) with respect to the number of siblings. Both Fig. 5.6 and Fig. 5.7 suggest a positive inductive bias being introduced by related tasks and Graph based formulation is doing slightly better when the relationships between tasks are known.

To study the effect of different relationships within the Graph-based formulation for SHMTL and MHMTL during the training phase, we repeated the experiments with various combinations of link types. The bar chart of average AUC scores are show in Fig. 5.8. P, S, and C denote the parent, sibling, and cross links respectively. For example *MHMTL PC* indicates that the MHMTL experiment was conducted only with parent links and sibling links were removed. We observed that for both CATH and SCOP, the sibling links helped more than the parent links. However, for MHMTL the effects on SCOP and CATH due to the removal of sibling or parent links was different. SCOP performance improved with removal of sibling links and decreased with removal of parent links while the effect on CATH was the opposite.

Figure 5.7: AUC improvement (SHMTL - STL) with number of siblings

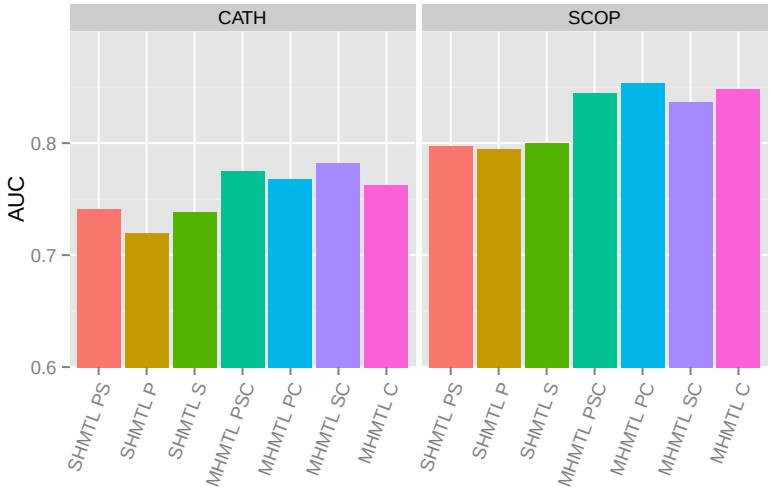


Figure 5.8: Average AUC for SCOP and CATH with different links in training set for Graph MTL

Table 5.5: Training times (in sec.)

		STL	SHMTL	MHMTL
L3	Sparse	873	214	227
	Graph	-	371	895
	Trace	-	1,233	1,721
L3+L4	Sparse	876	536	656
	Graph	-	1,159	3,463
	Trace	-	5,498	7,049

5.5.5 Run Time Analysis

Table 5.5 reports the run-times in seconds for the STL, SHMTL and MHMTL with different regularization penalties and for the L3-only and L3+L4 dataset. We can observe that the Graph-MHMTL takes longer in comparison to Graph-SHMTL. This is because there are more relationships (edges) that are added for the MHMTL method in comparison to the SHMTL method. With the addition of more tasks in L3+L4 dataset in comparison to the L3-only dataset, we notice a 3-5 fold increase in run time. Our analysis of AUC performance suggests that use of L4 nodes may not be necessary given the increase in run time.

5.6 Summary

In this work, we have shown the benefit of combining two hierarchical classification schemes to achieve better classification performance using the protein structure classification problem. Due to the nature of protein structure classification, the Sparse feature learning MTL formulation could not achieve any performance improvement over the baseline STL method. However, both the Graph and Trace MTL methods were able to learn better classifiers in both single hierarchy settings as well as multiple hierarchy settings. The combined Graph-Trace setting for MTL did not improve the classification performance over the individual Graph or Trace MTL setting. In our analysis we found that introducing the Family (SCOP) and Homologous Superfamily (CATH) tasks did not noticeably improve the classification,

but incurred some increase in runtime. For Graph based MTL formulation, which explicitly utilizes task relationships, we also examined the effect of various relationships on the classification performance.

Chapter 6: Convex Multi-task Relationship Learning using Hinge Loss

6.1 Introduction

Multi-task learning improves generalization performance by learning several related tasks jointly. Several methods have been proposed for multi-task learning in recent years. Many methods make strong assumptions about symmetric task relationships while some are able to utilize externally provided task relationships. In hierarchical classification as well, we are provided with an external hierarchy on the tasks which can be utilized in learning the models for tasks. In many real world settings, the relatedness among tasks is not explicitly provided and it needs to be inferred from the data. This task relationship inference adds an additional level of complexity in the learning process. In addition, even if it is known that certain tasks are related, the degree of to which these tasks can be said to be related might vary significantly. Therefore, in addition to the mere fact that two tasks are related, knowing the strength of the relationship can have a significant impact on the modeling of these relationships. Besides, most multi-task learning settings are capable of exploiting only positive correlations between tasks, but sometimes even negative correlations can help by restricting the possible search space for models. In terms of modeling the tasks as classification vectors, the assumption is that the linear weight vectors are similar but with some task specific variations. However, there might be instances where tasks are negatively correlated, in the sense that the model vectors are similar in the magnitude of values but opposite in sign. The standard multi-task settings find it difficult to model these kind of correlations.

In this chapter, we present a model for joint learning of tasks that also concurrently models task relationships. The main objective of the method presented here is to extend the

relationship learning formulation to max-margin based hinge loss formulation. Due to the non-smooth nature of hinge-loss, the optimization of the objective function is a non-trivial task. Therefore, we propose an efficient optimization algorithm using bundle methods for regularized risk minimization. We have validated our method on one simulated and two real world datasets and have compared its performance to competitive baseline single-task and multi-task methods.

Notations We use T for the number of tasks and d for the number of input dimensions. n_t denotes the number of examples in task t. $W \in \mathbb{R}^{d \times T}$ denotes the matrix of task weights, where each column $w_t \in \mathbb{R}^d$ is the weight vector for tasks $t \in [1:T]$, the rows of W are denoted by $\tilde{w}_j \in \mathbb{R}^T$ for $j \in [1:d]$. $\Omega \in \mathbb{R}^{T \times T}$ denotes the task covariance matrix. To simplify notation, we also define $\Sigma \equiv \Omega^{-1}$, where Ω^{-1} is the inverse of Ω.

6.2 Related Work

Multi-task learning has been empirically and theoretically shown to perform well in joint learning settings. However, a common assumption made by many formulations is that the tasks are equally related to each other [10,13] or some form of external task relationships are made available. These formulations try to enforce the constraints that tasks which are related to each other should be similar with respect to their parameter vectors. This assumption breaks down in several problem domains where the learning algorithm does not have access to relationships over the tasks. In such cases, joint learning of tasks can significantly impact the generalization performance of the learned models due to *negative transfer*. Cluster multi-task learning formulations [41,42] try to overcome the adverse affects of negative transfer by clustering tasks. The essential idea is that tasks can be grouped into clusters such that tasks within each cluster share a greater degree of similarity within the cluster and are relatively dissimilar to tasks belonging to other cluster. Therefore, tasks belonging to the same cluster are forced to learn similar model parameters and consequently only the tasks which are related fall into the same cluster can influence each other. However, this approach has two

main drawbacks. Firstly, it can limit the transfer of information between tasks belonging to the same cluster and secondly, it is unable to model negative correlations between tasks.

In general joint learning setting, the common assumption of symmetric task relationships and clustering can only be a crude approximation of the actual interactions or interrelationships between tasks. Generalizing the assumption of relationships between tasks from a binary relationship exist or does not exist type scenarios, we can further argue that tasks can positively correlated, negatively correlated or unrelated [31]. The knowledge of positive correlations can be enforced as constraint such that the tasks are forced to learn similar model parameters. Similarly learning can be formulated such that the learning algorithm promotes dissimilarity between negatively correlated tasks and decouples the learning of uncorrelated tasks. The multi-task relationship learning (MTRL) for smooth squared loss was proposed by Zang and Yeung [31]. The key feature of this formulation is that it does not require the task relationships to be known beforehand and it can exploit both the positive and negative task correlations. In order to extend the formulation to classification problems, it can be noted that the same regularizer can be utilized in order to leverage its advantages while using a loss function well suited for classification.

6.3 Methods

6.3.1 Multi-task Relationship Learning

Regularized risk minimization has been a dominant theme in machine learning. The general idea of regularized risk minimization has been implemented in several learning algorithms such as regularized least squares, regularized logistic regression, support vector machines and many other learning formulations. The choice of the appropriate regularizer and the loss function play an essential role in the learning algorithm. The bias on the model is imposed through the regularization term and in fact, in many successful MTL methods, the regularizer [28, 31, 39, 43] imposes the multi-task learning bias. However, the role of the loss function is also non-trivial and can play a significant part in the generalization

ability of the learned models [102]. For classification, the perfect loss function would be 0-1 loss, however, due to its non-convexity, it poses several challenges for optimization and consequently is not used in practice. Several surrogates of 0-1 loss with desirable properties for optimization, such as convexity and smoothness, are used in practice, the most common of which are logistic loss and hinge loss. Although logistic loss function yields a smooth optimization problem, hinge loss has certain theoretical and practical desirable properties, such as promoting max-margin solution and statistical guarantees of error bounds [102]. Hence, we chose to extend the MTRL formulation using SVM hinge loss [51]. The new model for MTLR using hinge loss is given by the optimization problem in (6.1).

$$\begin{aligned} \min_{W,\Omega} \; J = & \sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \left|1 - y_{it}\left(w_t^T x_{it}\right)\right|_+ + \frac{\lambda_1}{2} tr\left(WW^T\right) \\ & + \frac{\lambda_2}{2} tr\left(W\Omega^{-1}W^T\right) \\ s.t. \; & \Omega \succeq 0 \\ & tr\left(\Omega\right) = T \end{aligned} \tag{6.1}$$

In problem (6.1), the objective consists of the hinge loss term which decomposes over all the examples and all the tasks. The regularizer is composed of two parts — the first term $\frac{\lambda_1}{2} tr\left(WW^T\right)$ is the Frobenius norm of W, which penalizes the complexity of the weight matrix, and the second term $\frac{\lambda_2}{2} tr\left(W\Omega^{-1}W^T\right)$ penalizes the deviations between correlated tasks based on the task covariance matrix Ω. Due to the replacement of a smooth squared loss with non-smooth hinge loss the problem becomes significantly more difficult to solve. Hence, we propose the following optimization procedure to solve it efficiently.

6.3.2 Optimization

Theorem 1. *Problem (6.1) is jointly convex in both W and Ω*

Proof. To prove that (6.1) is convex we need to prove that the objective function is convex and the constraints define a convex set. Since the sum of convex functions preserves convexity, we only need to show that the individual terms of the objective function are convex. It is obvious that the first two terms in the objective function in (6.1) are convex in terms of W, the first being the sum of hinge losses and the second being a norm. The last term $tr\left(W\Omega^{-1}W^T\right)$ can be rewritten as $\sum_{j=1}^{D} \tilde{w}_j\Omega^{-1}\tilde{w}_j^T$ where $\tilde{w}_j$ denotes the j^{th} row of W. The function $f(x, Y) = x^TY^{-1}x$, where $x \in R^n$ and $Y \succeq 0$, known as the *matrix fractional function*, is convex (for proof refer to [72] page 76). Once again, due to the convexity preserving nature of sum, the last term is also convex. Finally, the constraint $\Omega \succeq 0$ defines a positive semi-definite cone and the $tr(M)$ is just an affine function of the matrix M, and hence, the intersection of both the constraints defines a convex set. □

Although problem (6.1) is jointly convex in W and Ω, simultaneous optimization over both W and Ω is difficult. Therefore, we use an alternating optimization strategy [114] and iterate between optimizing W with Ω fixed and vice-versa. The complete optimization procedure is summarized in Algorithm 6.

Optimizing Ω with fixed W

With W fixed, we can ignore the terms which are not dependent on Ω. The reduced problem involving only terms dependent on Ω can be rewritten as (6.2).

$$\min_{W,\Omega} tr\left(W\Omega^{-1}W^T\right) \tag{6.2}$$

$$s.t.\ \Omega \succeq 0$$

$$tr\left(\Omega\right) = T$$

The analytical solution to the above problem can be obtained in the closed form given

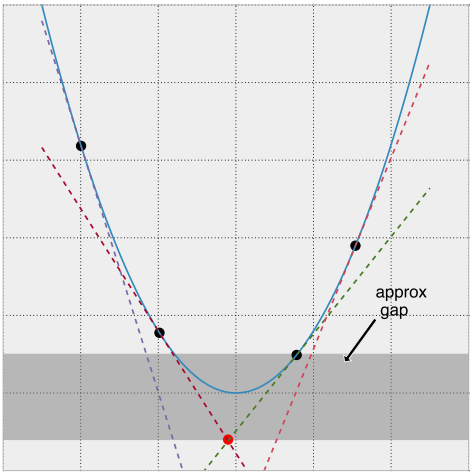


Figure 6.1: Illustration of convex function lower bounded by cutting planes. Cutting planes are tangents to the curve. The black dots represent the points at which the cutting planes are defined. The piecewise linear approximation is defined by the collective maxima of the cutting planes. The red dot represents the current minima of J_k^{CP}

by (6.3),

$$\Omega = \frac{T(W^T W)^{\frac{1}{2}}}{tr\left((W^T W)^{\frac{1}{2}}\right)} \tag{6.3}$$

This can be proved using the Cauchy-Schwarz inequality for the Frobenius norm. For a detailed proof, please refer to [31].

Optimizing W with fixed Ω

To solve this problem, we use the Bundle Methods for Regularized Risk Minimization (BMRM) proposed by Teo *et al.* [115]. BMRM is an optimization scheme for efficiently solving convex optimization problems commonly encountered in regularized risk minimization framework. This method converges in $\mathcal{O}(\epsilon)$ steps for general convex problems and in $\mathcal{O}(\log(1/\epsilon))$ steps for smooth convex problems.

The central idea of BMRM is based on the cutting plane and bundle methods, which is

to bound a convex objective function using a piecewise linear approximations. The piecewise linear approximation in iteration k is denoted by J_k^{CP}. This is illustrated in Fig. 6.1 for a single variable convex function. The cutting planes are defined by the (sub)gradients to the curve at each of the points marked by black circles. The collective maximum of the cutting planes, $J_k^{CP} := \max_{1 \leq j \leq k} \left\{ tr\left(W^T A_j\right) + b_j \right\}$, provides an approximation of the objective function. Due to convexity of the objective function, the cutting planes, and as a consequence J_k^{CP}, always lower bound the objective. New cutting planes are added to the approximation as the algorithm progresses to make the lower bound progressively tighter. The highlighted gray area in Fig. 6.1 marks the approximation gap, which is defined as the difference between the current best minimum of the objective function and the minimum of the cutting plane approximation J_k^{CP}

Even though the cutting plane method is convergent [116], it generally suffers from instability and extremely slow convergence [117] when new iterates move far away from the previous ones. To mitigate this problem, bundle methods add a proximal term to prevent the zig-zag behavior caused by next iterates moving too far away from the current iterates. There are three popular variants of bundle methods [115].

proximal:

$$w_k = \mathbf{argmin}_w \left\{ \frac{\xi_k}{2} \|w - \hat{w}_{k-1}\|^2 + J_k^{CP}(w) \right\} \tag{6.4a}$$

trust region:

$$w_k = \mathbf{argmin}_w \left\{ J_k^{CP}(w) \mid \frac{1}{2} \|w - \hat{w}_{k-1}\|^2 \leq \kappa_k \right\} \tag{6.4b}$$

level set:

$$w_k = \mathbf{argmin}_w \left\{ \frac{1}{2} \|w - \hat{w}_{k-1}\|^2 \mid J_k^{CP}(w) \leq \tau_k \right\} \tag{6.4c}$$

As it can be seen for (6.4a), (6.4b), and (6.4c), each of the variants penalize large steps in some form. However tuning the exact parameters for the proximal terms for achieving good rate of convergence can be difficult. The main insight of BMRM, which is based on *proximal* bundle methods, is to note that the objective function in regularized risk minimization consists of a loss term and a regularization term, and typically the regularization term is a norm. Therefore, the regularization term can be used as the proximal term, obviating the need for a specialized proximal term and the additional inconvenience associated with its parameter tuning.

To apply BMRM to Problem (6.1) we split the objective into two parts — the loss term and the regularization term.

$$J = R_{emp} + \Psi$$

where

$$R_{emp} = \sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \left|1 - y_{it} \left(w_t^T x_{it}\right)\right|_+$$

$$\Psi = \frac{\lambda_1}{2} tr\left(WW^T\right) + \frac{\lambda_2}{2} tr\left(W\Sigma W^T\right)$$

We proceed by defining a cutting plane approximation of R_{emp} and use Ψ as the proximal term. The cutting plane approximation is defined by

$$R_k^{CP}(W) = \max_{1 \le i \le k} \left\{tr\left(W^T A_j\right) + b_j\right\} \tag{6.5}$$

where $A_k \in \partial R_{emp}(W_{k-1})$ and $b_k = R_{emp}(W_{k-1}) - tr\left(W_{k-1}^T A_k\right)$. Therefore, we minimize problem (6.6) to get the next iterate

$$\begin{aligned} J_k(W) &= \Psi(W) + \max_{1 \le i \le k} \left\{ tr\left(W^T A_j\right) + b_j \right\} \\ W_k &= \arg\min_W J_k(W) \end{aligned} \tag{6.6}$$

The algorithm terminates when the gap between the approximation and the original objective function falls below a specified tolerance level ϵ, i.e. $min_{0 \le i \le k} J(W_j) - J_k(W_k) \le \epsilon$.

We solve the approximate problem (6.6) in its dual form using following result from [115] restated in a modified form for the current problem.

Theorem 2. *Let $\{A_j\}_{j=1}^k$ be the set of (sub)gradients and $b = (b_1, b_2, \ldots, b_k)^T$ be defined such that $b_k = R_{emp}(W_{k-1}) - tr\left(W_{k-1}^T A_k\right)$. The dual problem of*

$$W_k = \arg\min_W \{J_k(W)\}$$

$$J_k(W) \equiv \Psi(W) + \max_{1 \le j \le t} \left\{ tr\left(W^T A_j\right) + b_j \right\}$$

is

$$\alpha_k = \arg\max_{\alpha \in \mathbb{R}^t} \left\{ J_k^*(\alpha) = \Psi^* \left(-\sum_{j=1}^k A_j \alpha_j \right) + \alpha^T b \right\}$$

subject to:

$$\alpha \ge 0$$

$$\|\alpha\|_1 = 1$$

where Ψ^* *denotes the Fenchel Dual of* Ψ*. Furthermore,* W_k *and* α_k *are related by the dual connection* $W_k = \partial\Psi^*\left(-\sum_j A_j\alpha_j\right)$

Please note that our statement differs slightly, in notation, from the original theorem presented in [115] because the variables in the original theorem are presented in a vector form, whereas in our case the optimization variables are presented in matrix format.

Fenchel Dual of Ψ

In order to provide a concrete instantiation of the dual problem we have to compute the Fenchel dual of Ψ.

Definition 1. *Fenchel Dual: Let* $\phi : \mathcal{W} \to \mathbb{R}$ *be a convex function on a convex set* $\mathcal{W}$*. Then the dual* ϕ^* *of* ϕ *is defined as*

$$\phi^*(\mu) := \sup_{w\in\mathcal{W}} w^T\mu - \phi(w) \tag{6.7}$$

The matrix equivalent of dot product for vectors is the trace of the matrix product. Hence, the Fenchel dual of $\Psi(\cdot)$ is computed as follows.

$$\begin{aligned}\Psi^* &= \sup_W tr\left(U^TW\right) - \Psi(W) \\ &= \sup_W tr\left(U^TW\right) - \frac{\lambda_1}{2}tr\left(WW^T\right) - \frac{\lambda_2}{2}tr\left(W\Sigma W^T\right)\end{aligned} \tag{6.8}$$

To maximize R.H.S., we take its derivative w.r.t. W and equate it to zero to find the stationary point, noting that Σ is symmetric.

$$\left|\frac{\partial F\left(W,U\right)}{\partial W}\right|_{W=W*}=0$$

$$U-\lambda_1 W^*-\lambda_2 W^*\Sigma=0$$

$$\implies W^*=\ U\left(\lambda_1 I+\lambda_2\Sigma\right)^{-1}\equiv UB,$$

where we have defined $B=\left(\lambda_1 I+\lambda_2\Sigma\right)^{-1}$. Substituting back into (6.8), we get the closed form expression for Fenchel dual, which we further simplify using the cyclic property of trace [118].

$$\begin{aligned}\Psi^*=&tr\left(U^TUB\right)-\frac{\lambda_1}{2}tr\left(UBB^TU\right)-\frac{\lambda_2}{2}tr\left(UB\Sigma B^TU^T\right)\\=&\underline{tr\left(UBU^T\right)}-\frac{\lambda_1}{2}tr\left(UBB^TU\right)-\frac{\lambda_2}{2}tr\left(UB\Sigma B^TU^T\right)\\=&tr\left\{U\left(B-\frac{\lambda_1}{2}BB^T-\frac{\lambda_2}{2}B\Sigma B^T\right)U^T\right\}\\=&tr\left\{UGU^T\right\},\end{aligned}$$

where we define $G:=\left(B-\frac{\lambda_1}{2}BB^T-\frac{\lambda_2}{2}B\Sigma B^T\right)$

Finally, the connection between primal and dual variables according to Theorem 2 is given by

Algorithm 6 Main Algorithm (MTRL-hinge)

Input: $X,\ Y,\ \lambda_1,\ \lambda_2$
Output: W, Ω
$W_0 \leftarrow 0;\ \Omega_0 \leftarrow T * I$
$\epsilon \leftarrow 10^{-3}\ max\_iter \leftarrow 30;\ k \leftarrow 0;\ \epsilon_k \leftarrow \infty$
for $k = 1, \ldots, max\_iter$ **do**
  $W_k := optimizeW\ (X, Y, W, \Omega_k, \lambda_1, \lambda_2)$
  $\Omega_k := T * \frac{(W_k^T W_k)^{\frac{1}{2}}}{tr\left((W_k^T W_k)^{\frac{1}{2}}\right)}$
  if $\frac{\|W_k - W_{k-1}\|_2}{\|W_k\|_2} > \epsilon$ **then**
    break;
  end
end
return W_k, Ω_k

$$
\begin{aligned}
W_k &= \partial\Psi^* \left(-\sum_j A_j \alpha_j \right) \\
&= \left| \frac{\partial tr\left\{ UGU^T \right\}}{\partial U} \right|_{U=-\sum_j A_j \alpha_j} \\
&= |2UG|_{U=-\sum_j A_j \alpha_j} \\
&= -2 \left(\sum_j A_j \alpha_j \right) G
\end{aligned}
\tag{6.9}
$$

The dual problem in this form can be solved using any constrained quadratic solver. This optimization procedure is summarized in Algorithm 7.

6.4 Experimental Setup

6.4.1 Validation Protocols

In this section we present the results of experimental evaluation of our method on a simulated dataset and two real world datasets commonly used in multi-task learning literature and

Algorithm 7 OptimizeW (BMRM)

Input: $W_1, X, Y, \Omega, \lambda_1, \lambda_2$
Output: W_{best}
$k \leftarrow 1, \epsilon \leftarrow 10^{-4}, max\_iter \leftarrow 100$
$B = \left(\lambda_1 I + \lambda_2 \Omega^{-1}\right)^{-1}$
$G = \left(B - \frac{\lambda_1}{2} BB^T - \frac{\lambda_2}{2} B\Omega^{-1} B^T\right)$
for $k = 1, \ldots, max\_iter$ **do**
  $A_k \in \partial R_{emp}\left(W_{k-1}\right)$
  $b_k := R_{emp}\left(W_{k-1}\right) - tr\left(W_{k-1}^T A_k\right)$
  M such that $M_{pq} = tr\left\{A_p G A_q^T\right\}$
  $\alpha_k = \arg\min_{\alpha \in \mathbb{R}^k} \left\{\alpha^T M \alpha - \alpha^T b \mid \alpha \geq 0, \mathbf{1}^T \alpha = 1\right\}$
  $W_k = -2\left(\sum_{j=1}^{k} A_j \alpha_j\right) G$
  $W_{best} = \mathbf{argmin}_{W \in \{W_1, \ldots, W_k\}} \left\{J\left(W\right)\right\}$
  $\epsilon_k \leftarrow J\left(W_{best}\right) - J_k\left(W_k\right)$
  if $\epsilon_k < \epsilon$ **then**
    break
  end
end
return W_{best}

compare the results with a single task model and two competitive multi-task learning models described below.

MTRL: Our method proposed in this chapter.

SVM: STL baseline using support vector machines. SVM is the most competitive single task learning baseline because our method uses hinge loss in its objective.

TRACE: MTL formulation using Trace Norm minimization [43, 107] tries to enforce a low rank constraint on the combined weights matrix for all the tasks, thereby favoring the weight vectors for the tasks that lie in a low dimensional subspace. Due to the difficulties involved in directly minimizing the rank of a matrix, trace norm is used as a surrogate.

L21: MTL formulation uses L_{21} norm as the regularizer [8, 119]. This has the effect of joint feature selection on all the tasks.

Both TRACE and L21 MTL formulations use a smooth logistic loss which is different from the non-smooth hinge loss used in our method. For our experiments we used the publicly available implementation of libSVM for SVM [1]. TRACE and L21 are available in MALSAR package [108], which provides implementations for several well known multi-task methods using logistic loss and squared loss.

6.5 Results

6.5.1 Simulated Toy Dataset

In this section, we discuss the results on a simulated dataset. The purpose of testing our method on this dataset is to verify that our method is indeed able to extract the task relationships as encoded in the data. By controlling the dataset generation process we know the ground truth about the task relationship, which can be then compared with the results of the method execution. To generate this dataset, we create two groups of four tasks each by first generating two orthonormal vectors which form the bases for the two groups. Then, we add random noise to the weight vectors of the group weight vectors to generate four independent task weight vectors in each group. Therefore, by construction we have one group of four tasks which has high correlation within the group and another group of four tasks with similarly high correlation within the group. Finally, we generate random examples and assign positive and negative classes using the generated weight vectors. Any correlation, positive or otherwise, with tasks from the other group are mere chance occurrences. We chose the number of input dimensions as 4. Each task has equal number of positive and negative training examples. We used a validation set for tuning the model parameters in the range $\left\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\right\}$ and measured the performance on an independent test set.

In Table 6.1, we compare the accuracy of our MTRL formulation with that of STL using SVM. Our MTRL model outperforms SVM by a huge margin when the amount of training

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

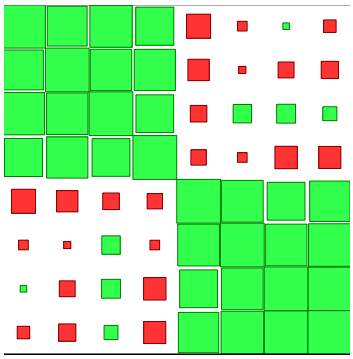


Figure 6.2: Learned task correlations for simulated data using 10 training examples. Each square represents the correlation between a pair of tasks. The size of the squares represents the magnitude of the value with positive values shown in green and negative values in red.

Table 6.1: Simulated dataset accuracy

Train Size (# Examples)	SVM	MTRL
5	0.8158	0.8785
10	0.8979	0.9186
30	0.9815	0.9822
50	0.9941	0.9948

data per task is small. Given sufficient amount of training data, SVM performs just as well as our MTRL formulation. In Fig. 6.2 we show the task correlations for our MTRL model learned with 10 training examples. The figure shows two distinct groups of tasks uncovered by our method.

6.5.2 Landmine Detection

The Landmine detection dataset[2] consists of 29 tasks. Each task is a binary classification problem of learning a classification model to discriminate between instances of landmines and clutters using features extracted from radar images. The 9 features in this dataset consist of four moment-based features, three correlation-based features, one energy ratio feature and one spatial variance feature [120]. Tasks 1-15 are taken from regions with high

[2]http://people.ee.duke.edu/~lcarin/LandmineData.zip

foliage and the rest are taken from bare earth or desert regions. Therefore, it is reasonable to assume that the tasks form two distinct groups. The number of examples per task vary between 445 and 690, whereas the number of positive examples per task vary in the range from 15 to 48.

We selected 25% of the examples for test set and 25% for validation set and using the remaining 50% of the examples we created datasets of varying sizes to evaluate classification performance at different training sizes. We used the validation set to select the best parameters for all the methods in the range $\left\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\right\}$.

Due to the imbalance in the number of positive and negative examples in this dataset, accuracy is not a good measure of performance. Hence, we evaluate the performance of different models using area under the receiver operating characteristic curve (AUC). The results for this dataset are provided in Table 6.2. In general, with increasing training size the performance of the classifier improves, as can be expected. Our MTRL model performs better than the STL using SVM, but for this dataset, the performance using hinge loss based models was considerably worse than the models using logistic loss.

As mentioned earlier, the tasks in this dataset are naturally clustered into two groups. In order to observe our method's ability to extract these groups at various training sizes, we plotted the correlations of the weight vectors of the models extracted from the learned model parameter Ω in Fig. 6.3, for different training sizes. We observed that the first group of tasks, consisting of tasks 1-15, is more strongly correlated than the second group of tasks, consisting of task 16-29. With sufficient training sample sizes, the correlation pattern between tasks recovered by MTRL converges to the expected pattern consisting of two distinct groups.

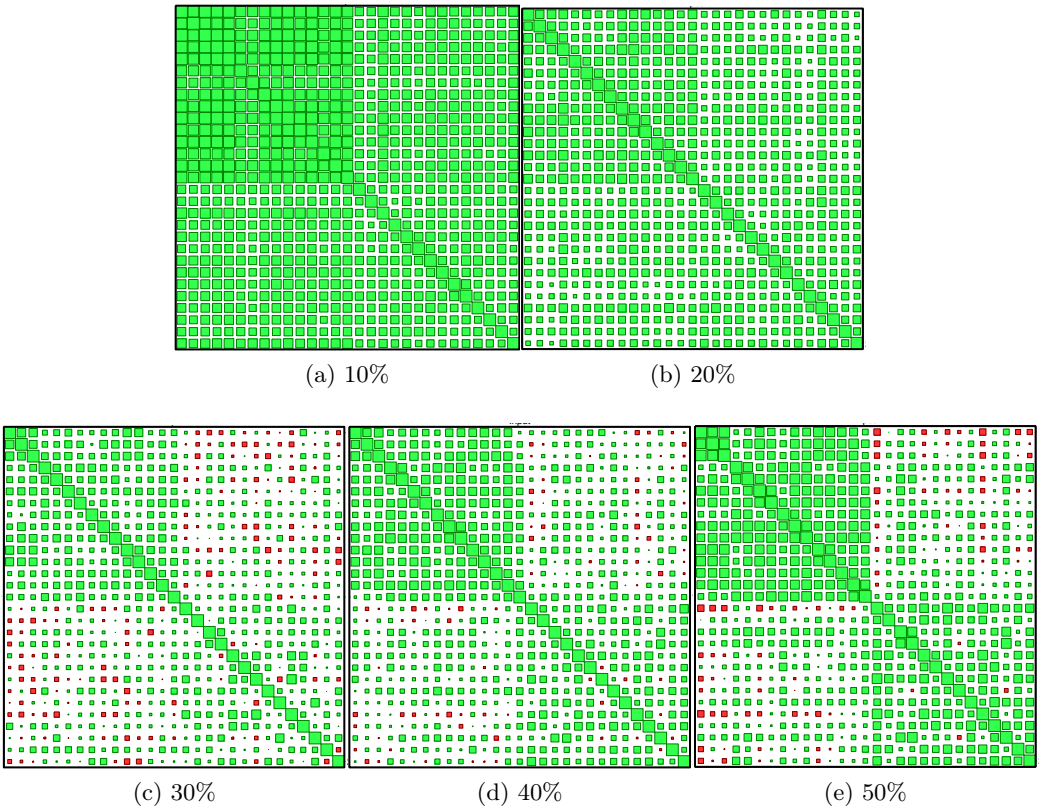
6.5.3 Amazon Sentiment Classification

The tasks in Amazon sentiment classification dataset [3] are to classify the the polarity of different product reviews using their text. This dataset was originally provided by Blitzer

[3] http://www.cs.jhu.edu/~mdredze/datasets/sentiment/

Table 6.2: Landmine AUC scores

Train Size (%)	SVM	MTRL	TRACE	L21
10	0.6902	0.6708	0.7512	0.7533
20	0.6697	0.6688	0.7393	0.7384
30	0.6841	0.6914	0.7711	0.7676
40	0.6983	0.7107	0.7708	0.7646
50	0.7016	0.7225	0.7724	0.7682



(a) 10% (b) 20%

(c) 30% (d) 40% (e) 50%

Figure 6.3: Task correlations of best models for Landmine dataset. Sub-figures correspond to different percentages of data used for training. Each square represents the correlation between a pair of tasks. The size of the squares represents the magnitude of the value with positive values shown in green and negative values in red.

et al. [121] and used in the context of domain adaptation. The instances consist of product reviews, and ratings, which are provided in terms of 1 to 5 stars. The ratings are converted into positive and negative reviews for classification tasks. Each task corresponds to a particular product category - books, DVDs, electronics, and kitchen appliances. 1000 positive and 1000 negative examples are available for each task. The instances are represented by a term frequency vector of 473856 feature dimensions. We selected 25% examples for testing and 25% examples for validation and to assess the performance with different training sizes, we created different training split with 10%, 20%, 30%, 40%, and 50% of the original data. We validated the models using parameters in the range $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$ on a validation set and selected the best model for final test on a held out test set.

Accuracies on the test set for various methods are reported in Table 6.3. Both MTRL and Trace perform considerably better than STL and L21. We expected L21 to perform well, given the high dimensionality of this dataset, but, it performed worse than STL for this dataset. The overall performance of our model was better than all other models.

In Table 6.4, we show the task correlations recovered by our model for different tasks. We observed a strong correlation between the Books-DVDs and Electronics-Kitchen Appliances task pairs.

6.6 Summary

In this work, we have presented a multi-task model using non-smooth hinge loss which is capable of learning the task relationships between different tasks. Our method can not only use positive task relationships, but it can also leverage negative task relationships. We demonstrated the effectiveness of the method in improving generalization performance using a simulated dataset and two real world datasets. Our method compared favorably with competing multi-task learning methods and consistently outperformed them on one of the datasets. We also compared the performance of our method to competing STL and MTL methods at different training sizes and found that the MTL methods significantly outperform STL methods and the performance gains are more pronounced for smaller training sizes.

Table 6.3: Per task accuracy - Amazon Sentiment Classification

Train Size (%)	Method	Books	DVDs	Electronic	Kitchen Appliances
10	SVM	0.6360	0.6800	0.7720	0.7560
	MTRL	**0.7380**	0.7500	**0.8280**	**0.8100**
	TRACE	0.7000	**0.7620**	0.8180	0.7920
	L21	0.6380	0.6960	0.7640	0.7620
20	SVM	0.7040	0.7680	0.8080	0.8020
	MTRL	0.7580	**0.7860**	0.8640	**0.8420**
	TRACE	**0.7600**	0.7680	**0.8660**	**0.8420**
	L21	0.7000	0.7400	0.8200	0.7620
30	SVM	0.7540	0.7880	0.8060	0.8160
	MTRL	0.7860	**0.8020**	0.8600	**0.8520**
	TRACE	**0.7940**	0.7960	**0.8620**	0.8420
	L21	0.7280	0.7440	0.8320	0.8320
40	SVM	0.7420	0.8180	0.8200	0.8240
	MTRL	**0.7980**	**0.8380**	**0.8640**	**0.8340**
	TRACE	0.7860	0.8260	0.8560	0.8280
	L21	0.7320	0.8020	0.8160	0.8220
50	SVM	0.7740	0.8160	0.8240	0.8540
	MTRL	**0.8080**	**0.8440**	**0.8820**	**0.8640**
	TRACE	0.7940	0.8300	0.8720	0.8460
	L21	0.7360	0.8140	0.8420	0.8400

Table 6.4: Task correlation - Amazon Sentiment Classification

Train Size(%)	Correlation Tables			
	Books	DVDs	Electronic	Kitchen Appliances
10	1.0000	0.7830	0.6192	0.5629
	0.7830	1.0000	0.7291	0.7000
	0.6192	0.7291	1.0000	0.9413
	0.5629	0.7000	0.9413	1.0000
20	1.0000	0.5848	0.5492	0.5516
	0.5848	1.0000	0.4589	0.5263
	0.5492	0.4589	1.0000	0.7397
	0.5516	0.5263	0.7397	1.0000
30	1.0000	0.5898	0.5774	0.6009
	0.5898	1.0000	0.5557	0.5885
	0.5774	0.5557	1.0000	0.7925
	0.6009	0.5885	0.7925	1.0000
40	1.0000	0.6084	0.4683	0.5578
	0.6084	1.0000	0.4847	0.5593
	0.4683	0.4847	1.0000	0.7450
	0.5578	0.5593	0.7450	1.0000
50	1.0000	0.6835	0.5638	0.6386
	0.6835	1.0000	0.5782	0.6214
	0.5638	0.5782	1.0000	0.8179
	0.6386	0.6214	0.8179	1.0000

With respect to extracting task correlations we observed that some amount of training data is required to extract robust task correlations.

Chapter 7: Future Work

In this thesis, we have presented several methods to improve the efficiency and effectiveness in classification with several tasks in hierarchical and multi-task learning scenarios. There are several interesting directions to explore for future work.

7.1 Large Scale Multi-task Learning

Majority of the current work in multi-task learning deals with small scale problems compared to the web-scale problems encountered in the real world. To be practically applicable, these methods should be able to scale to scenarios involving thousands or even millions of tasks. For example modeling the news preferences of subscribers of an online news website, or the buying preferences of the users of a web-retailer using current methods is impractical. The primary challenge is that these methods involve complex regularization penalties that involve all the model variables. Hence, using the current optimization methods limits us to the memory and computational capability of current commodity computers or mid-sized servers. However, with further understanding into various regularization penalties that can be advantageous to multiple tasks and efficient implementation of distributed optimization methods, these multi-task methods can be scaled to extremely large problems. Promising advances in distributed optimization methods have been made in recent years, such as parallel block-coordinate descent methods [122]. Understanding the applicability of these methods to current multi-task formulations and designing efficient implementations in distributed computing frameworks such as map-reduce or spark is a challenging and interesting direction.

7.2 Hierarchical Classification Targeting Application Specific Losses

Most binary, multi-class and hierarchical classification methods are geared towards minimizing 0/1 misclassification error, albeit using a convex surrogate instead of using 0/1 loss directly. Most of the current algorithms only achieve minimization/maximization on other metrics as a by product of minimizing misclassification error. In many application domains it might be more beneficial to optimize other metrics such as tree-error or hierarchical f1-score rather than targeting misclassification error in order to maximize the utility of the classifier. Recently, Chen *et al.* [75] proposed a method to maximize revenue generation targeting specific business goals. The large margin method proposed by Dekel *et al.* [59] is geared towards minimizing the tree induced error instead of the misclassification error. However, such work targeting specific classification metrics is sparse in comparison to methods focusing on misclassification error. More work needs to be done to understand the specific properties of different loss functions and their applicability and feasibility in learning problems.

One promising line of attack can be inspired from Search based structured prediction (SEARN) proposed by Daume et al. [123]. SEARN is applicable in more general context of structure output learning. It is a meta-algorithm which reduces the problem of learning over structured output spaces to binary classification. The general idea is to start with an initial policy for classification over the output space and iteratively refine it using the losses incurred by the classification policy of the previous iteration. The algorithm is able to incorporate any user defined loss function and relatively efficient in practice.

7.3 Top-down Classification with Feature Selection

Hierarchical classification methods that train one-vs-rest type of models for all the terminal class labels have been noted to empirically outperform their hierarchical counterparts that train top down models [11, 79]. However, one-vs-rest flat classifiers are considerably more expensive in terms of both training times and testing times. In the domains where this

trade-off of slight decrease in classification performance is acceptable in order to achieve significant improvement in training and testing efficiency, top-down classification can be a feasible alternative.

In large scale classification problems where feature space as well as the number of classes are large, the number of features that are relevant to any local classifier in top-down approach tend to small. However, traditional approaches tend to use all the features in learning the models. Such problems are commonly encountered in large scale text classification with large vocabularies. The motivation for this method is based on the observation that although the number of features in the entire vocabulary of English language is enormous, for any given classification problem the majority of the words are not relevant for the classification problem of a particular node in the hierarchy. The discriminative features for classifiers at different nodes might be different. In the case of text classification, for example, the important features at a node related to computer science might be words such as compilers, programs, and so on, whereas words like seeds, crop, and wheat might be more important for agriculture. Words which are discriminative at a more general level might cease to be so at a more specific level. Although, the entire set of features might be used by the overall hierarchical classification problem, only a subset of features will be used by the different independent models.

Determining the relevant features for each local classifier can significantly improve efficiency while simultaneously mitigating the impact of noisy features. Towards this end, we make use of the advances in Multi-task learning feature selection and feature learning methods. Considering the classification at an internal node with c children, we are presented with the problem of multi-class classification to distinguish between the c children. Without any feature selection, we are can use any multi-class classifier such as binary SVM in one-vs-rest settings or direct multi-class SVM formulation [52]. However our requirement in the current case is to learn a set of features that are useful for all the classes. It has two fold advantage, on one hand, feature selection reduces the number of parameters of the model therefore the models tend to be compact; secondly for several categories in large scale classification suffer

from insufficient data problem as illustrated on Yahoo! directory dataset by Lui *et al.* [1], therefore by incorporating the multi-task bias of selecting or learning relevant features for related tasks should supposedly help.

For a given set of related learning problem multi-task feature selection solves the following problem given in (7.1)

$$\min_{W} \sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}\left(y_{it}, w_t^T x_{it}\right) + \gamma \left\|W\right\|_{2,1} \tag{7.1}$$

T is the number of tasks, and for task t, the number of examples is represented by n_t and weight vector by w_t. (y_{it}, x_{it}) denote the i^{th} example for task t, where $y_{it} \in \pm 1$. Although, the feature learning problem was originally proposed in [8]. Two methods to solve the problem efficiently using accelerated proximal gradient descent were proposed by Liu et al. [28].

7.4 Imbalance Classification Methods for Large Scale Hierarchical Classification

The limited success of hierarchical classification methods on large scale problems, has been attributed to three main causes: sparsity of training data at deeper levels in the hierarchy, complex decision surfaces at higher levels because they are composed of several more specific classes, and error propagation where mistakes at higher levels can not be corrected at lower levels [1,68]. While the third reason is applicable mainly to the top-down local classification methods, the first two are inherent to the nature of these problems. Typically, for majority of the classes, the ratio of number of positive examples to that of the negative examples is very small. This problem can be seen in many standard text classification datasets. Going from top to down in the hierarchy the classes increasingly become sparser. In Fig. 7.1 this phenomenon can be seen for for Yahoo! document classification hierarchy where the category

size exhibits a power-law distribution. Some methods have tried to indirectly deal with this issue by biasing the parameters towards the more general, subsuming categories [11, 16]. Although the performance tends to be high on measures such as Micro-F_1, the measures such as Macro-F_1 which give equal importance to all the categories, the performance tends to be very poor.

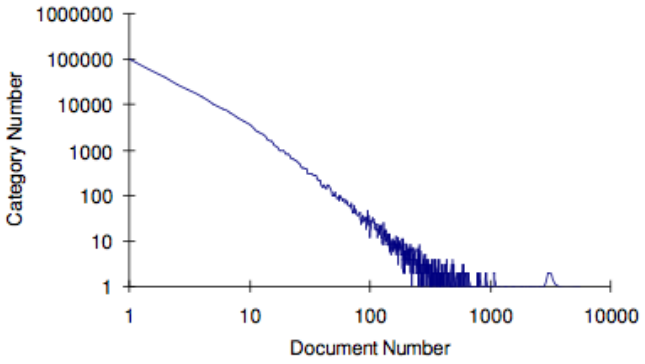


Figure 7.1: Power law distribution of class sizes for Yahoo! documents dataset [1]

Considering this extremely skewed distribution of examples, methods need to be developed to address the sparsity of training data. In recent years, *Imbalance classification* has gained considerable attention. The task addressed by Imbalance classification is concerned with improvements in the performance of learning algorithms when the class distribution is highly skewed [124]. The major challenge here is that the induction rules that describe the minority concepts are often fewer and weaker than those of majority concepts, since the minority class is often both outnumbered and under-represented. Two major directions in which the problem has been addressed are sampling methods and cost sensitive methods. In chapter 4, we have tried to mitigate the effect of imbalance by incorporating higher misclassification cost for smaller categories. In addition, one-class classification methods have also been applied to the problem with some degree of success [125]. Since, one class classification models the characteristics of a single class, their application imbalance classification has been successful because characterizing the specifics of minority classes takes the center

stage. It would be interesting to investigate the applicability and adaptation of imbalance classification methods to the problem of hierarchical classification, especially to deal with the imbalance in large scale hierarchies.

Bibliography

Bibliography

[1] Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *ACM SIGKDD Explorations Newsletter*, 7(1):36–43, 2005.

[2] Carlos N Silla Jr and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[3] M.S. Sorower. A literature survey on algorithms for multi-label learning.

[4] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

[5] Alina Beygelzimer, John Langford, and Bianca Zadrozny. Machine learning techniques—reductions between prediction quality metrics. In *Performance Modeling and Engineering*, pages 3–28. Springer, 2008.

[6] J. Baxter. A model of inductive bias learning. *JAIR*, 12:149–198, 2000.

[7] S. Thrun. Is learning the n-th thing any easier than learning the first? *NIPS*, pages 640–646, 1996.

[8] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. *NIPS*, 19:41, 2007.

[9] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[10] T. Evgeniou and M. Pontil. Regularized multitask learning. *KDD*, pages 109–117, 2004.

[11] Siddharth Gopal and Yiming Yang. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In *Proceedings of the 19th ACM SIGKDD*, pages 257–265. ACM, 2013.

[12] Siddharth Gopal, Yiming Yang, Bing Bai, and Alexandru Niculescu-Mizil. Bayesian models for large-scale hierarchical classification. In *NIPS*, pages 2420–2428, 2012.

[13] T. Evgeniou, C.A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6(1):615, 2006.

[14] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[15] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[16] Andrew McCallum, Ronald Rosenfeld, Tom M Mitchell, and Andrew Y Ng. Improving text classification by shrinkage in a hierarchy of classes. In *ICML*, volume 98, pages 359–367, 1998.

[17] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of molecular biology*, 247(4):536–540, 1995.

[18] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

[19] C.A. Orengo, AD Michie, S. Jones, D.T. Jones, MB Swindells, and J.M. Thornton. Cath-a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1109, 1997.

[20] Anveshi Charuvaka and Huzefa Rangwala. Multi-task learning for classifying proteins using dual hierarchies. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 834–839. IEEE, 2012.

[21] Anveshi Charuvaka and Huzefa Rangwala. Approximate block coordinate descent for large scale hierarchical classification. In *Proceedings of The 30th ACM SIGAPP Symposium On Applied Computing*, 2015.

[22] Anveshi Charuvaka and Huzefa Rangwala. Convex multi-task relationship learning using hinge loss. In *Computational Intelligence and Data Mining (CIDM), 2014 IEEE Symposium on*, pages 63–70. IEEE, 2014.

[23] Anveshi Charuvaka and Huzefa Rangwala. Multi-task learning for classifying proteins using dual hierarchies. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE, 2012.

[24] Anveshi Charuvaka and Huzefa Rangwala. Classifying protein sequences using regularized multi-task learning. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2014.

[25] S.J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 2009.

[26] S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. *Learning Theory and Kernel Machines*, pages 567–580, 2003.

[27] A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008.

[28] J. Liu, S. Ji, and J. Ye. Multi-task feature learning via efficient l 2, 1-norm minimization. *UIA*, pages 339–348, 2009.

[29] S. Thrun and J. O'Sullivan. Clustering learning tasks and the selective cross-task transfer of knowledge. *Learning to learn*, pages 181–209, 1998.

[30] E. Bonilla, K.M. Chai, and C. Williams. Multi-task gaussian process prediction. *NIPS*, 20(October), 2008.

[31] Y. Zhang and D.Y. Yeung. A convex formulation for learning task relationships in multi-task learning. In *Proceedings of the Twenty-fourth Conference on Uncertainty in AI (UAI)*, 2010.

[32] Y. Qi, D. Liu, D. Dunson, and L. Carin. Multi-task compressive sensing with dirichlet process priors. In *Proceedings of the 25th international conference on Machine learning*, pages 768–775. ACM, 2008.

[33] Jian Zhang, Zoubin Ghahramani, and Yiming Yang. Flexible latent variable models for multi-task learning. *Machine Learning*, 73(3):221–242, 2008.

[34] T. Jebara. Multi-task feature and kernel selection for SVMs. *ICML*, page 55, 2004.

[35] T. Kato, H. Kashima, M. Sugiyama, and K. Asai. Multi-task learning via conic programming. *Advances in Neural Information Processing Systems*, 20:737–744, 2008.

[36] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[37] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

[38] D.L. Donoho. For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution. *Communications on pure and applied mathematics*, 59(6):797–829, 2006.

[39] G. Obozinski, B. Taskar, and M.I. Jordan. Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20(2):231–252, 2010.

[40] Seyoung Kim and Eric P. Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In *ICML*, pages 543–550, 2010.

[41] J. Zhou, J. Chen, and J. Ye. Clustered multi-task learning via alternating structure optimization. *NIPS*, 2011.

[42] Laurent Jacob, Francis Bach, and Jean-Philippe Vert. Clustered multi-task learning: A convex formulation. *CoRR*, abs/0809.2085, 2008.

[43] S. Ji and J. Ye. An accelerated gradient method for trace norm minimization. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 457–464. ACM, 2009.

[44] Andreas Argyriou, Andreas Maurer, and Massimiliano Pontil. An algorithm for transfer learning in a heterogeneous environment. In *Machine Learning and Knowledge Discovery in Databases*, pages 71–85. Springer, 2008.

[45] Ke Wang, Senqiang Zhou, and Shiang Chen Liew. Building hierarchical classifiers using class proximity. In *VLDB*, volume 99, pages 363–374, 1999.

[46] Celine Vens, Jan Struyf, Leander Schietgat, Sašo Džeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

[47] Miguel E Ruiz and Padmini Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5(1):87–118, 2002.

[48] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. In *ICML*, pages 170–178, 1997.

[49] Alexander Genkin, David D Lewis, and David Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[50] Lin Xiao, Dengyong Zhou, and Mingrui Wu. Hierarchical classification via orthogonal transfer. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 801–808, 2011.

[51] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[52] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2):201–233, 2002.

[53] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004.

[54] Aris Kosmopoulos, Ioannis Partalas, Eric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. Evaluation measures for hierarchical classification: a unified view and novel approaches. *arXiv preprint arXiv:1306.6802*, 2013.

[55] W. Bi and J.T. Kwok. Multi-label classification on tree-and dag-structured hierarchies. ICML, 2011.

[56] Farbound Tai and Hsuan-Tien Lin. Multilabel classification with principal label space transformation. *Neural Computation*, 24(9):2508–2542, 2012.

[57] Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In *NIPS*, volume 22, pages 772–780, 108, 2009.

[58] Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830, 2006.

[59] Ofer Dekel, Joseph Keshet, and Yoram Singer. Large margin hierarchical classification. In *Proceedings of the twenty-first international conference on Machine learning*, page 27. ACM, 2004.

[60] Artem Sokolov and Asa Ben-Hur. Hierarchical classification of gene ontology terms using the gostruct method. *Journal of Bioinformatics and Computational Biology*, 8(02):357–376, 2010.

[61] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 78–87. ACM, 2004.

[62] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[63] S. Dumais and H. Chen. Hierarchical classification of Web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, page 263. ACM, 2000.

[64] Nicolò Cesa-Bianchi, Claudio Gentile, Luca Zaniboni, et al. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7(1), 2006.

[65] Michelangelo Ceci and Donato Malerba. Classifying web documents in a hierarchy of categories: a comprehensive study. *Journal of Intelligent Information Systems*, 28(1):37–78, 2007.

[66] Aris Kosmopoulos, Eric Gaussier, Georgios Paliouras, and Sujeevan Aseervatham. The ecir 2010 large scale hierarchical classification workshop. In *ACM SIGIR Forum*, volume 44, pages 23–32. ACM, 2010.

[67] Yiming Yang, Jian Zhang, and Bryan Kisiel. A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 96–103. ACM, 2003.

[68] Paul N Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 11–18. ACM, 2009.

[69] Gui-Rong Xue, Dikan Xing, Qiang Yang, and Yong Yu. Deep classification in large-scale text hierarchies. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 619–626. ACM, 2008.

[70] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

[71] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th international conference on Machine learning*, pages 408–415. ACM, 2008.

[72] S.P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge Univ Pr, 2004.

[73] Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[74] Ivica Dimitrovski, Dragi Kocev, Suzana Loskovska, and Sašo Džeroski. Hierarchical annotation of medical images. *Pattern Recognition*, 44(10):2436–2449, 2011.

[75] Jianfu Chen and David Warren. Cost-sensitive learning for large-scale hierarchical classification. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1351–1360. ACM, 2013.

[76] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. In *Journal of Machine Learning Research*, pages 1453–1484, 2005.

[77] Babak Shahbaba, Radford M Neal, et al. Improving classification when a class hierarchy is available using a hierarchy-based prior. *Bayesian Analysis*, 2(1):221–237, 2007.

[78] Budhaditya Saha, Sunil Gupta, Dinh Phung, and Svetha Venkatesh. Multiple task transfer learning with small sample sizes. *Knowledge and Information Systems*, pages 1–28, 2015.

[79] Rohit Babbar, Ioannis Partalas, Eric Gaussier, and Massih-Reza Amini. On flat versus hierarchical classification in large-scale taxonomies. In *Advances in Neural Information Processing Systems*, pages 1824–1832, 2013.

[80] Hamed Masnadi-Shirazi and Nuno Vasconcelos. Risk minimization, probability elicitation, and cost-sensitive svms. In *ICML*, pages 759–766, 2010.

[81] Yurii Nesterov. *Introductory lectures on convex optimization*, volume 87. Springer Science & Business Media, 2004.

[82] Yiming Yang. A study of thresholding strategies for text categorization. In *Proceedings of the 24th annual international ACM SIGIR*, pages 137–145. ACM, 2001.

[83] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[84] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR*, pages 42–49. ACM, 1999.

[85] C.A. Orengo, AD Michie, S. Jones, D.T. Jones, MB Swindells, and J.M. Thornton. CATH-a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1109, 1997.

[86] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.

[87] Hiroto Saigo, Jean-Philippe Vert, Nobuhisa Ueda, and Tatsuya Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20(11):1682–9, 2004.

[88] C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: A string kernel for svm protein classification. In *Proceedings of the pacific symposium on biocomputing*, volume 7, pages 566–575. Hawaii, USA., 2002.

[89] H. Rangwala and G. Karypis. Profile-based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, 21(23):4239–4247, 2005.

[90] J. Weston, C. Leslie, E. Ie, D. Zhou, A. Elisseeff, and W.S. Noble. Semi-supervised protein classification using cluster kernels. *Bioinformatics*, 21(15):3241, 2005.

[91] G. Csaba, F. Birzele, and R. Zimmer. Systematic comparison of SCOP and CATH: a new gold standard for protein structure analysis. *BMC Structural Biology*, 9(1):23, 2009.

[92] R. Day, D.A.C. Beck, R.S. Armen, and V. Daggett. A consensus view of fold space: Combining SCOP, CATH, and the Dali Domain Dictionary. *Protein Science*, 12(10):2150–2160, 2003.

[93] C. Hadley and D.T. Jones. A systematic comparison of protein structure classifications: Scop, cath and fssp. *Structure*, 7(9):1099–1112, 1999.

[94] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[95] SF Altschul, TL Madden, AA Schaffer, J. Zhang, Z. Zhang, W. Miller, and DJ Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389, 1997.

[96] C. Leslie and R. Kuang. Fast kernels for inexact string matching. *Learning Theory and Kernel Machines*, pages 114–128, 2003.

[97] R. Kuang, IE EUGENE, K. Wang, K. Wang, M. Siddiqi, Y. Freund, and C. Leslie. Profile-based string kernels for remote homology detection and motif extraction. *Journal of bioinformatics and computational biology*, 3(03):527–550, 2005.

[98] Pavel P. Kuksa, Pai-Hsi Huang, and Vladimir Pavlovic. Scalable algorithms for string kernels with inexact matching. In *Advances in Neural Information Processing Systems 20*, pages 881–888, 2008.

[99] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

[100] H. Rangwala and G. Karypis. Building multiclass classifiers for remote homology detection and fold recognition. *BMC bioinformatics*, 7(1):455, 2006.

[101] I. Melvin, E. Ie, J. Weston, W.S. Noble, and C. Leslie. Multi-class protein classification using adaptive codes. *Journal of Machine Learning Research*, 8(1557-1581):6, 2007.

[102] L. Rosasco, E.D. Vito, A. Caponnetto, M. Piana, and A. Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

[103] G. Obozinski, B. Taskar, and M. Jordan. Multi-task feature selection. *ICML*, 2006.

[104] T. Kato, H. Kashima, M. Sugiyama, and K. Asai. Conic programming for multitask learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(7):957–968, 2010.

[105] M. Fazel, H. Hindi, and S.P. Boyd. A rank minimization heuristic with application to minimum order system approximation. In *American Control Conference, 2001. Proceedings of the 2001*, volume 6, pages 4734–4739. IEEE, 2001.

[106] Jacob Abernethy, Francis Bach, Theodoros Evgeniou, and Jean-Philippe Vert. Low-rank matrix factorization with attributes. *CoRR*, abs/cs/0611124, 2006.

[107] T.K. Pong, P. Tseng, S. Ji, and J. Ye. Trace norm regularization: Reformulations, algorithms, and multi-task learning. *SIAM Journal on Optimization*, 2009.

[108] J.Zhou, J.Chen, and J. Ye. *MALSAR: Multi-tAsk Learning via StructurAl Regularization*. Arizona State University, 2011.

[109] Yu. NESTEROV. Gradient methods for minimizing composite objective function. CORE Discussion Papers 2007076, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), September 2007.

[110] C.S. Leslie, E. Eskin, A. Cohen, J. Weston, and W.S. Noble. Mismatch string kernels for discriminative protein classification. *Bioinformatics*, 20(4):467–476, 2004.

[111] P. Kuksa, P.H. Huang, and V. Pavlovic. Fast protein homology and fold detection with sparse spatial sample kernels. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, 2008.

[112] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *Artificial Neural Networks—ICANN'97*, pages 583–588. Springer, 1997.

[113] M.H. Zweig and G. Campbell. Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine. *Clinical chemistry*, 39(4):561–577, 1993.

[114] James Bezdek and Richard Hathaway. Some notes on alternating optimization. *Advances in Soft Computing—AFSS 2002*, pages 187–195, 2002.

[115] Choon Hui Teo, SVN Vishwanthan, Alex J Smola, and Quoc V Le. Bundle methods for regularized risk minimization. *The Journal of Machine Learning Research*, 11:311–365, 2010.

[116] James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial & Applied Mathematics*, 8(4):703–712, 1960.

[117] Krzysztof C Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46(1-3):105–122, 1990.

[118] K.B. Petersen and M.S. Pedersen. The matrix cookbook. *Technical University of Denmark*, 2006.

[119] X. Chen, W. Pan, J.T. Kwok, and J.G. Carbonell. Accelerated gradient method for multi-task sparse learning problem. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 746–751. IEEE, 2009.

[120] Y. Xue, X. Liao, L. Carin, and B. Krishnapuram. Multi-task learning for classification with dirichlet process priors. *The Journal of Machine Learning Research*, 8:35–63, 2007.

[121] J. Blitzer, M. Dredze, and F. Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Annual Meeting-Association For Computational Linguistics*, volume 45, page 440, 2007.

[122] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *arXiv preprint arXiv:1212.0873*, 2012.

[123] Hal Daumé Iii, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[124] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, 2009.

[125] Larry M Manevitz and Malik Yousef. One-class svms for document classification. *the Journal of machine Learning research*, 2:139–154, 2002.

Curriculum Vitae

Anveshi Charuvaka was born and raised in India. Growing up, he lived in several places across India, thanks to his father's job, which gave him the opportunity to witness India's unity in diversity first hand. He did his schooling from various schools of the Kendriya Vidyala Sanghathan, and obtained a degree of Bachelors of Technology in Information Science and Technology, with first class distinction, from Koneru Lakshmaiah College of Engineering. He joined Wipro Technologies as a Project Engineer straight out of college, but soon afterwards moved to Oracle India Pvt. Ltd. as an Applications Engineer to pursue more challenging undertakings. Here, he got the opportunity to work with several very talented people in helping develop *Oracle Fusion Applications*. After spending about 3 years in software development, he decided to pursue a doctoral degree in Computer Science at George Mason University, and moved to USA in August 2009. Upon graduation, he will be joining General Electric's Global Research as a Senior Data Scientist at their Software Development Center in San Ramon, California.