Online and Adaptive Methods for Multitask Learning

Keerthiram Murugesan

August 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Avrim Blum, TTI-Chicago
Jaime Carbonell (*Chair*), CMU
Louis-Philippe Morency, CMU
Barnabás Póczos, CMU

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Language and Information Technologies.

Keywords: Multitask Learning, Lifelong Learning, Sequential Decision Making

To my parents

Abstract

The power of joint learning in multiple tasks arises from the transfer of relevant knowledge across said tasks, especially from information-rich tasks to information-poor ones. Lifelong learning, on the other hand, provides an efficient way to learn new tasks faster by utilizing the knowledge learned from the previous tasks and prevent catastrophic forgetting or significantly degrading performance on the old tasks. Despite several advantages on learning from related tasks, it poses considerable challenges in terms of effectiveness by minimizing prediction errors for all tasks and overall computational tractability for real-time performance, especially when the number of tasks is large. In contrast, human beings seem natural in accumulating and retaining the knowledge from the past and leverage this knowledge to acquire new skills and solve new problems efficiently.

In this thesis, we propose simple and efficient algorithms for learning from related tasks to address the aforementioned challenges. We present algorithms that feature probabilistic interpretation, efficient updating rules and flexible modulation on whether learners focus on their specific task or on jointly address all tasks. We develop a novel approach to active learning for sequential problems that first determines if the learner can acquire a label from its peers. If so, it saves the query for later use in more difficult cases, and if not it queries the human. We define a new machine learning paradigm based on a curriculum defined dynamically by the learner ("self-paced") instead of a fixed curriculum set *a-priori* by a teacher. The primary focus of this thesis is to scale the multitask and lifelong learning to practical applications where both the tasks and the examples of the tasks arrive in an online fashion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multitask methods leverage the relationships between tasks to jointly build a better model for each task. By exploiting the latent structure on task-relatedness, it introduces inductive bias in joint learning of multiple related tasks thereby improve generalization and prevent overfitting. Most existing work in multitask learning focuses on how to take advantage of these task relationships, either to share data directly [31] or to learn model parameters via cross-task regularization techniques [9, 114, 120]. In a broad sense, there are two settings to learn these task relationships 1) batch learning, in which an entire training set is available to the learner 2) online learning, in which the learner sees the data in a sequential fashion. In recent years, online multitask learning has attracted extensive research attention [1, 25, 35, 73, 96].

Unlike in the traditional multitask learning, where the tasks are presented simultaneously and an entire training set is available to the learner ([21]), in lifelong learning the tasks arrive sequentially ([106]). In this thesis, we are interested in a continuous lifelong learning setting in which one or both the tasks and the examples of the tasks arrive in an online fashion, without any predetermined order. Although the objectives of both online multitask learning and lifelong learning are similar, one of the main key differences is that the online multitask learning, unlike in the lifelong learning, may require that the number of tasks is specified beforehand.

There are many useful application areas for multitask and lifelong learning, including optimizing financial trading, email prioritization, personalized news, and spam filtering. Consider the latter, where some spam is universal to all users (e.g. financial scams), some messages might be useful to certain affinity groups, but spam to most others (e.g. announcements of meditation classes or other special interest activities), and some may depend on evolving user interests. In spam filtering each user is a task, and shared interests and dis-interests formulate the transfer of information between the tasks. If we can learn the underlying task relations as well as improving models from specific spam/not-spam decisions, we can perform mass customization of spam filtering, borrowing from spam/not-spam feedback from users with similar preferences.

1.1 Related Topics

In this section, we consider the topics closely related to this thesis. We briefly introduce the concepts here and explain them as when needed.

1.1.1 Multitask Learning

Multitask learning leverage the relationships between tasks to jointly build a better model for each task. Most existing work in multitask learning focuses on how to take advantage of these task relationships, either to share data directly [31] or to learn model parameters via cross-task regularization techniques [9, 114, 120]. In a broad sense, there are two settings to learn these task relationships: 1) batch learning, in which an entire training set is available to the learner 2) online learning, in which the learner sees the data in a sequential fashion. In recent years, online multitask learning has attracted extensive research attention [1, 25, 35, 73, 96]. Most of our work in this thesis focuses on online multitask learning. We briefly introduce the problem setup under consideration in the next section.

1.1.2 Lifelong Learning

Lifelong learning, inspired by established human learning principles, works by accumulating and retaining the knowledge from the past and leverages this knowledge to acquire new skills and solve new problems efficiently. It poses considerable challenges in terms of effectiveness (minimizing prediction errors for all tasks) and overall computational tractability for real-time performance. A lifelong learning agent must provide an efficient way to learn new tasks faster by utilizing the knowledge learned from the previous tasks and also not forgetting or significantly degrading performance on the old tasks. The goal of a lifelong learner is to minimize errors as compared to the full ideal hindsight learner, which has access to all the training data and no bounds on memory or computation.

1.1.3 Transfer Learning

Multitask and lifelong learning have been studied in part under a related research topic, *transfer learning* [16] under different assumptions. There are several key differences between those methods and multitask learning: i) While transfer learning tries to find a *single* hypothesis that works well for the target data with the assistance of one or more source tasks, multitask learning finds a hypothesis for each task by adaptively leveraging related tasks. ii) It is a typical assumption in transfer learning that the source tasks are label-rich and the target tasks are label-scarce. However, multitask learning assumes the scenario where there is a large number of tasks with very few examples available for each task.

1.2 Problem Setup

We are given K tasks where the j^{th} task is associated with N_j training examples. For brevity we consider a binary classification problem in this thesis for each task, but the methods generalize to multi-class and are also applicable to regression tasks. We denote by $[N]$ the consecutive integers ranging from 1 to N. Let $\left\{(x_j^{(i)}, y_j^{(i)})\right\}_{i=1}^{N_j}$ and $L_j(w) = \frac{1}{N_j}\sum_{i\in[N_j]}\left(1 - y_j^{(i)}\langle x_j^{(i)}, w\rangle\right)_+$ be the training set and batch empirical loss for task j, respectively, where $(z)_+ = \max(0, z)$, $x_j^{(i)} \in \mathbb{R}^d$ is the i^{th} instance from the j^{th} task and $y_j^{(i)}$ is its corresponding true label. When the notation is

clear from the context, we drop the index k and write $((x^{(i)}, k), y^{(i)})$. The general form of the objective function for multitask learning is given by:

$$\{\hat{w}_k\}_{k\in[K]} = \arg\min_{\{w_k\}_{k\in[K]}} \sum_{j\in[K]} L_j(\mathbf{w_j}) + \mathcal{R}(\{w_k\}_{k\in[K]}) \tag{1.1}$$

where $\mathcal{R}(.)$ is the regularization term on the model parameters, associated with a regularization parameter and w_k is the model parameter for the k^{th} task.

Now, we extend our batch multitask formulation to the online setting. The online multitask setting starts with a learner at each round t. The learner of the k^{th} task receives a training instance $x_k^{(t)}$, makes a prediction $\langle x_k^{(t)}, w_k^{(t)} \rangle$ and suffers a loss after $y^{(t)}$ is revealed. This sequence is repeated over the entire data, simulating a data stream. Let $\{w_k^*\}_{k\in[K]}$ be any set of arbitrary vectors where $w_k^* \in \mathbb{R}^d$. The hinge loss on the individual example $\left((x^{(t)}, k), y^{(t)}\right)$ is given by $\ell_{kk}^{(t)*} = \left(1 - y^{(t)}\langle x^{(t)}, w_k^* \rangle\right)_+$, with an inter-task loss given by $\ell_{km}^{(t)*} = \left(1 - y^{(t)}\langle x^{(t)}, w_m^* \rangle\right)_+$. Similarly, we define hinge losses $\ell_{kk}^{(t)}$ and $\ell_{km}^{(t)}$ for the linear predictors $\{w_k^{(t)}\}_{k\in[K]}$ learned at round t.

Based on the application at hand, there are two common settings in online multitask learning: 1) In the first setting, all tasks will be performed simultaneously by receiving K examples for K tasks at each round t. 2) In the second setting, the learner receives a single example from a task (along with a task identifier) and predicts it's output label. Following the prediction, the learner either receives the true label automatically or request an oracle for true label thereby incurring a cost. The learner then *updates* the model(s) as necessary. The goal of an online multitask learner in this setting is to minimize errors attempting to reach the performance of the full hindsight learner and optionally, reduce the total number of queries issued to the oracle. The cumulative regret for the online multitask learning algorithm can be given as follows:

$$R_k = \sum_{t\in[T]} \left(\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}\right), \forall k \in [K] \tag{1.2}$$

1.3 Thesis Overview

1. **Online Multitask Learning** (*NIPS'16* [79]) In chapter 2, we address the challenge of jointly learning both the per-task model parameters and the inter-task relationships in a multitask online learning setting. The proposed algorithm features probabilistic interpretation, efficient updating rules and flexible modulation on whether learners focus on their specific task or on jointly address all tasks. We also prove a sub-linear regret bound as compared to the best linear predictor in hindsight. Experiments over three multitask learning benchmark datasets show an advantageous performance of the proposed approach over several state-of-the-art online multitask learning baselines. we extend our learning setting to multiple kernel learning where each task is associated with multiple kernels and propose a two-stage learning approach to learn the model parameters, optimal kernel weights and the task relationship efficiently (*SDM'17* [77]).
2. **Learning from Peers** (NIPS'17 [76]) In chapter 3, we address the challenge of learning from peers in an online multitask setting. Instead of always requesting a label from a human

oracle, the proposed method first determines if the learner for each task can acquire that label with sufficient confidence from its peers either as a task-similarity weighted sum or from the single most similar task. If so, it saves the oracle query for later use in more difficult cases, and if not it queries the human oracle. This chapter develops the new algorithm to exhibit this behavior and proves a theoretical mistake bound for the method compared to the best linear predictor in hindsight. Experiments over three multitask learning benchmark datasets show clearly superior performance over baselines such as assuming task independence, learning only from the oracle and not learning from peer tasks.

3. **Self-paced Multitask Learning** (*IJCAI'17* [78]) In chapter 4, we introduce self-paced task selection to multitask learning, where instances from more closely related tasks are selected in a progression of easier-to-harder tasks, to emulate an effective human education strategy but applied to lifelong machine learning. We develop the mathematical foundation for the approach based on an iterative selection of the most appropriate task, learning the task parameters, and updating the shared knowledge, optimizing a new bi-convex loss function. This proposed method allows us to use some of the existing multitask learning formulations to lifelong learning setting. Results show that in each of the above formulations self-paced (easier-to-harder) task selection outperforms the baseline version of these methods in all the experiments.
4. **Efficient Co-Clustering for Multitask and Lifelong Learning** (*Arxiv'18* [80]) In chapter 4, we propose a co-clustering approach to multitask and lifelong learning. We provide a flexible way to cluster both the features and the tasks using the shared feature representation. In addition, our proposed models learn both the task relationship matrix and the feature relationship matrix along with the co-clustering of both the tasks and the features. We introduce an additional degree of freedom that allows the number of task clusters to differ from the number of features clusters. A key challenge in factoring with the extra degree of freedom is optimizing the resulting objective function. We propose an efficient algorithm that scales well to large-scale multitask learning and utilizes the structure of the objective function to learn the factorized task parameters. We develop a highly scalable and efficient learning algorithm using conjugate gradient descent and generalized Sylvester equations. Extensive empirical analysis on both synthetic and real datasets show that the proposed method outperforms the other state-of-the-art multitask baselines.
5. **Lifelong Learning** (*Arxiv'18*) A lifelong learning agent must provide an efficient way to learn new tasks faster by utilizing the knowledge learned from the previous tasks and also not forgetting or significantly degrading performance on the old tasks. The chapter 4 addresses continuous lifelong multitask learning by jointly re-estimating the inter-task relations (*output* kernel) and the per-task model parameters at each round, assuming data arrives in a streaming fashion. A new algorithm called *Online Output Kernel Learning Algorithm* (OOKLA) for lifelong learning setting is proposed. To avoid the memory explosion, a robust budget-limited version of the proposed algorithm is introduced, which efficiently utilizes the relationships between the tasks to bound the total number of representative examples in the support set. In addition, a two-stage budgeted scheme for efficiently tackling the task-specific budget constraints in lifelong learning is proposed. Empirical results over three datasets indicate superior AUC performance for OOKLA and its budget-limited cousins

over strong baselines.

Chapter 2

Online Multitask Learning

In online learning, the learner sees the data in a sequential fashion. The key challenge in online learning of multiple tasks with a large number of tasks is to adaptively learn the model parameters and the task relationships, which potentially change over time. Without manageable efficient updates at each round, learning the task relationship matrix automatically may impose a severe computational burden. In this chapter, we address these concerns in order to make predictions and update the models in an efficient real-time manner.

We propose an online learning framework that *efficiently* learns multiple related tasks by estimating the task relationship matrix from the data, along with the model parameters for each task. Unlike in the previous works, we learn the model for each task by *sharing* data from related task directly [79]. Our model provides a natural way to specify the trade-off between learning the hypothesis from each task's own (possibly quite limited) data and data from multiple related tasks. Later in this chapter, we extend our learning setting to multiple kernel learning where each task is associated with multiple kernels and propose a two-stage learning approach to learn the model parameters, optimal kernel weights and the task relationship efficiently [77].

2.1 Smooth Multitask Learning

The power of joint learning in multiple tasks arises from the transfer of relevant knowledge across said tasks, especially from information-rich tasks to information-poor ones. Instead of learning individual models, multitask methods leverage the relationships between tasks to jointly build a better model for each task. Most existing work in multitask learning focuses on how to take advantage of these task relationships, either to share data directly [31] or to learn model parameters via cross-task regularization techniques [9, 114, 120]. In a broad sense, there are two settings to learn these task relationships 1) batch learning, in which an entire training set is available to the learner 2) online learning, in which the learner sees the data in a sequential fashion. In this chapter, we consider the online learning approaches to multitask problems. In recent years, online multitask learning has attracted extensive research attention due to its practical applications [1, 25, 35, 73, 96].

Following the online setting, particularly from [35, 73], at each round t, the learner *receives* a set of K observations from K tasks and *predicts* the output label for each of these observations.

Subsequently, the learner receives the true labels and *updates* the model(s) as necessary. This sequence is repeated over the entire data, simulating a data stream. Our approach follows an error-driven update rule in which the model for a given task is updated only when the prediction for that task is in error. The goal of an online learner is to minimize errors compared to the full hindsight learner. The key challenge in online learning with large number of tasks is to adaptively learn the model parameters and the task relationships, which potentially change over time. Without manageable efficient updates at each round, learning the task relationship matrix automatically may impose a severe computational burden. In other words, we need to make predictions and update the models in an efficient real time manner.

We propose an online learning framework that *efficiently* learns multiple related tasks by estimating the task relationship matrix from the data, along with the model parameters for each task. We learn the model for each task by sharing data from related task directly. Our model provides a natural way to specify the trade-off between learning the hypothesis from each task's own (possibly quite limited) data and data from multiple related tasks. We propose an iterative algorithm to learn the task parameters and the task-relationship matrix alternatively. We first describe our proposed approach under a batch setting and then extend it to the online learning paradigm. In addition, we provide a theoretical analysis for our online algorithm and show that it can achieve a sub-linear regret compared to the best linear predictor in hindsight. We evaluate our model with several state-of-the-art online learning algorithms for multiple tasks.

There are many useful application areas for online multitask learning, including optimizing financial trading, email prioritization, personalized news, and spam filtering. Consider the latter, where some spam is universal to all users (e.g. financial scams), some messages might be useful to certain affinity groups, but spam to most others (e.g. announcements of meditation classes or other special interest activities), and some may depend on evolving user interests. In spam filtering each user is a task, and shared interests and dis-interests formulate the inter-task relationship matrix. If we can learn the matrix as well as improving models from specific spam/not-spam decisions, we can perform mass customization of spam filtering, borrowing from spam/not-spam feedback from users with similar preferences.

2.1.1 Related Work

While there is considerable literature in online multitask learning, many crucial aspects remain largely unexplored. Most existing work in online multitask learning focuses on how to take advantage of task relationships. To achieve this, Lugosi et al. [73] imposed a hard constraint on the K simultaneous actions taken by the learner in the expert setting, Agarwal et al. [3] used matrix regularization, and Dekel et al. [35] proposed a global loss function, as an absolute norm, to tie together the loss values of the individual tasks. Different from existing online multitask learning models, we propose an intuitive and efficient way to learn the task relationship matrix automatically from the data, and to explicitly take into account the learned relationships during model updates.

Cavallanti et al. [25] assumes that task relationships are available *a priori*. Kshirsagar et al. [61] does the same but in a more adaptive manner. However such task-relation prior knowledge is either unavailable or infeasible to obtain for many applications especially when the number of tasks K is large [112] and/or when the manual annotation of task relationships is expensive

[62]. Saha et al. [96] formulated the learning of task relationship matrix as a Bregman-divergence minimization problem w.r.t. positive definite matrices. The model suffers from high computational complexity as semi-definite programming is required when updating the task relationship matrix at each online round. We show that with a different formulation, we can obtain a similar but much cheaper updating rule for learning the inter-task weights.

Multitask learning has been studied in part under a related research topic, *Domain Adaptation* (DA) [16] under different assumptions. There are several key differences between those methods and ours: i) While DA tries to find a *single* hypothesis that works well for both the source and the target data, this work finds a hypothesis for each task by adaptively leveraging related tasks. ii) It is a typical assumption in DA that the source domains are label-rich and the target domains are label-scarce. However, we are more interested in the scenario where there is a large number of tasks with very few examples available for each task. iii) DA uses predefined uniform weights or weights induced from VC-convergence theory during training, while our method allows cross-task weights to dynamically evolve in an adaptive manner.

The most related work to ours is *Shared Hypothesis* model (*SHAMO*) from Crammer and Mansour [31], where the key idea is to use a K-means-like procedure that simultaneously clusters different tasks and learns a small pool of $m \ll K$ shared hypotheses. Specifically, each task is free to choose a hypothesis from the pool that better classifies its own data, and each hypothesis is learned from pooling together all the training data that belongs to the same cluster. A similar idea was explored by Abernathy et al. [1] under expert settings.

2.1.2 Setup

Suppose we are given K tasks where the j^{th} task is associated with N_j training examples. For brevity we consider a binary classification problem for each task, but the methods generalize to multi-class and are also applicable to regression tasks. We denote by $[N]$ the consecutive integers ranging from 1 to N. Let $\left\{(x_j^{(i)}, y_j^{(i)})\right\}_{i=1}^{N_j}$ and $L_j(w) = \frac{1}{N_j}\sum_{i\in[N_j]}\left(1 - y_j^{(i)}\langle x_j^{(i)}, w\rangle\right)_+$ be the training set and batch empirical loss for task j, respectively, where $(z)_+ = \max(0, z), x_j^{(i)} \in \mathbb{R}^d$ is the i^{th} instance from the j^{th} task and $y_j^{(i)}$ is its corresponding true label.

We start from the motivation of our formulation in Section 2.1.2, based on which we first propose a batch formulation in Section 2.1.3. Then, we extend the method to the online setting in Section 2.1.4.

Motivation

Learning tasks may be addressed independently via $w_k^* = \text{argmin}_{w_k} L_k(w_k), \forall k \in [K]$. However, when each task has limited training data, it is often beneficial to allow information sharing among the tasks, which can be achieved via the following optimization:

$$w_k^* = \text{argmin}_{w_k} \sum_{j\in[K]} \eta_{kj} L_j(w_k) \quad \forall k \in [K] \tag{2.1}$$

Beyond each task k, optimization (2.1) encourages hypothesis w_k^* to do well on the remaining $K-1$ tasks thus allowing tasks to borrow information from each other. In the extreme case where

the K tasks have an identical data distribution, optimization (2.1) amounts to using $\sum_{j\in[K]} N_j$ examples for training as compared to N_k in independent learning.

The weight matrix $\boldsymbol{\eta}$ is in essence a task relationship matrix, and a prior may be manually specified according to domain knowledge about the tasks. For instance, η_{kj} would typically be set to a large value if tasks k and j share similar nature. If $\boldsymbol{\eta} = \boldsymbol{I}$, (2.1) reduces to learning tasks independently. It is clear that manual specification of $\boldsymbol{\eta}$ is feasible only when K is small. Moreover, tasks may be statistically correlated even if a domain expert is unavailable to identify an explicit relation, or if the effort required is too great. Hence, it is often desirable to automatically estimate the optimal $\boldsymbol{\eta}$ adapted to the inter-task problem structure.

We propose to learn η in a data-driven manner. For the k^{th} task, we optimize

$$w_k^*, \eta_k^* = \text{argmin}_{w_k, \eta_k \in \Theta} \sum_{j\in[K]} \eta_{kj} L_j(w_k) + \lambda r(\eta_k) \tag{2.2}$$

where Θ defines the feasible domain of η_k, and regularizer r prevents degenerate cases, e.g., where η_k becomes an all-zero vector. Optimization (2.2) shares the same underlying insight with Self-Paced Learning (SPL) [55, 65] where the algorithm automatically learns the weights over data points during training. However, the process and scope in the two methods differ fundamentally: SPL minimizes the weighted loss over *datapoints* within a single domain, while optimization (2.2) minimizes the weighted loss over multiple *tasks* across possibly heterogeneous domains.

A common choice of Θ and $r(\eta_k)$ in SPL is $\Theta = [0, 1]^K$ and $r(\eta_k) = -\|\eta_k\|_1$. There are several drawbacks of naively applying this type of settings to the multitask scenario: (i) *Lack of focus*: there is no guarantee that the k^{th} learner will put more focus on the k^{th} task itself. When task k is intrinsically difficult, η_{kk}^* could simply be set near zero and w_k^* becomes almost independent of the k^{th} task. (ii) *Weak interpretability*, the learned η_k^* may not be interpretable as it is not directly tied to any physical meanings (iii) *Lack of worst-case guarantee* in the online setting. All those issues will be addressed by our proposed model in the following.

2.1.3 Batch Formulation

We parametrize the aforementioned task relationship matrix $\boldsymbol{\eta} \in \mathbb{R}^{K\times K}$ as follows:

$$\boldsymbol{\eta} = \alpha \boldsymbol{I}_K + (1 - \alpha)\, \boldsymbol{P} \tag{2.3}$$

where $\boldsymbol{I}_K \in \mathbb{R}^{K\times K}$ is an identity matrix, $\boldsymbol{P} \in \mathbb{R}^{K\times K}$ is a *row-stochastic matrix* and α is a scalar in $[0, 1]$. Task relationship matrix $\boldsymbol{\eta}$ defined as above has the following interpretations:

1. *Concentration Factor* α quantifies the learners' "concentration" on their own tasks. Setting $\alpha = 1$ amounts to independent learning. We will see from the forthcoming Theorem 1 how to specify α to ensure the optimality of the online regret bound.
2. *Smoothed Attention Matrix* $\boldsymbol{P}$ quantifies to which degree the learners are attentive to all tasks. Specifically, define the k^{th} row of $\boldsymbol{P}$, namely $p_k \in \Delta^{K-1}$, as a probability distribution over all tasks where Δ^{K-1} denotes the probability simplex. Our goal of learning a data-adaptive $\boldsymbol{\eta}$ now becomes learning a data-adaptive attention matrix $\boldsymbol{P}$.

Algorithm 1: Batch Algorithm (*SMTL*-e)

while *not converge* **do**
  for $k \in [K]$ **do**
    $w_k^{(t)} \leftarrow \text{argmin}_{w_k} \ \alpha L_k(w_k) + (1-\alpha) \sum_{j\in[K]} p_{kj}^{(t)} L_j(w_k)$;
    for $j \in [K]$ **do**
      $p_{kj}^{(t+1)} \leftarrow \frac{e^{-\frac{1-\alpha}{\lambda} L_j(w_k^{(t)})}}{\sum_{j'=1}^{K} e^{-\frac{1-\alpha}{\lambda} L_{j'}(w_k^{(t)})}}$;
    end
  end
  $t \leftarrow t+1$;
end

Common choices about $\boldsymbol{\eta}$ in several existing algorithms are special cases of (2.3). For instance, domain adaptation assumes $\alpha = 0$ and a fixed row-stochastic matrix $\boldsymbol{P}$; in multitask learning, we obtain the effective heuristics of specifying $\boldsymbol{\eta}$ by Cavallanti et al. [25] when $\alpha = \frac{1}{1+K}$ and $\boldsymbol{P} = \frac{1}{K}\mathbf{1}\mathbf{1}^\top$. When there are $m \ll K$ unique distributions p_k, then the problem reduces to *SHAMO* model [31].

Equation (2.3) implies the task relationship matrix $\boldsymbol{\eta}$ is also row-stochastic, where we always reserve probability α for the k^{th} task itself as $\eta_{kk} \geq \alpha$. For each learner, the presence of α entails a trade off between learning from other tasks and concentrating on its own task. Note that we do not require $\boldsymbol{P}$ to be symmetric due to the asymmetric nature of information transferability—while classifiers trained on a resource-rich task can be well transferred to a resource-scarce task, the inverse is not usually true. Motivated by the above discussion, our batch formulation instantiates (2.2) as follows

$$w_k^*, p_k^* = \text{argmin}_{w_k, p_k \in \Delta^{K-1}} \sum_{j\in[K]} \eta_{kj}(p_k) L_j(w_k) - \lambda \mathcal{H}(p_k) \tag{2.4}$$

$$= \text{argmin}_{w_k, p_k \in \Delta^{K-1}} \ \mathbb{E}_{j\sim Multinomial(\eta_k(p_k))} L_j(w_k) - \lambda \mathcal{H}(p_k) \tag{2.5}$$

where $\mathcal{H}(p_k) = -\sum_{j\in[K]} p_{kj} \log p_{kj}$ denotes the entropy of distribution p_k. Optimization (2.4) can be viewed as to balance between minimizing the cross-task loss with mixture weights η_k and maximizing the smoothness of cross-task attentions. The max-entropy regularization favours a uniform attention over all tasks and leads to analytical updating rules for p_k (and η_k).

The pseudo-code is in Algorithm 1. Optimization (2.4) is biconvex over w_k and p_k. With $p_k^{(t)}$ fixed, solution for w_k can be obtained using off-the-shelf solvers. With $w_k^{(t)}$ fixed, solution for p_k is given in closed-form:

$$p_{kj}^{(t+1)} = \frac{e^{-\frac{1-\alpha}{\lambda} L_j(w_k^{(t)})}}{\sum_{j'=1}^{K} e^{-\frac{1-\alpha}{\lambda} L_{j'}(w_k^{(t)})}} \quad \forall j \in [K] \tag{2.6}$$

The exponential updating rule in (2.6) has an intuitive interpretation. That is, our algorithm attempts to use hypothesis $w_k^{(t)}$ obtained from the k^{th} task to classify training examples in all other

tasks. Task j will be treated as related to task k if its training examples can be well classified by w_k. The intuition is that two tasks are likely to relate to each other if they share similar decision boundaries, thus combining their associated data should yield to a stronger model, trained over larger data.

2.1.4 Online Formulation

In this section, we extend our batch formulation to the online setting. We assume that all tasks will be performed at each round, though the assumption can be relaxed with some added complexity to the method. At time t, the k^{th} task receives a training instance $x_k^{(t)}$, makes a prediction $\langle x_k^{(t)}, w_k^{(t)} \rangle$ and suffers a loss after $y^{(t)}$ is revealed. Our algorithm follows a error-driven update rule in which the model is updated only when a task makes a mistake.

Let $\ell_{kj}^{(t)}(w) = 1 - y_j^{(t)} \langle x_j^{(t)}, w \rangle$ if $y_j^{(t)} \langle x_j^{(t)}, w_k^{(t)} \rangle < 1$ and $\ell_{kj}(w) = 0$ otherwise. For brevity, we introduce shorthands $\ell_{kj}^{(t)} = \ell_{kj}^{(t)}(w_k^{(t)})$ and $\eta_{kj}^{(t)} = \eta_{kj}(p_k^{(t)})$.

For the k^{th} task we consider the following optimization problem at each time:

$$w_k^{(t+1)}, p_k^{(t+1)} = \underset{w_k, p_k \in \Delta^{K-1}}{\operatorname{argmin}} \; C \sum_{j \in [K]} \eta_{kj}(p_k) \ell_{kj}^{(t)}(w_k) + \|w_k - w_k^{(t)}\|^2 + \lambda D_{\mathrm{KL}}\big(p_k \| p_k^{(t)}\big) \tag{2.7}$$

where $\sum_{j \in [K]} \eta_{kj}(p_k) \ell_{kj}^{(t)}(w_k) = \mathbb{E}_{j \sim Multi(\eta_k(p_k))} \ell_{kj}^{(t)}(w_k)$, and $D_{\mathrm{KL}}\big(p_k \| p_k^{(t)}\big)$ denotes the Kullback–Leibler (KL) divergence between current and previous soft-attention distributions. The presence of last two terms in (2.7) allows the model parameters to evolve smoothly over time. Optimization (2.7) is naturally analogous to the batch optimization (2.4), where the batch loss $L_j(w_k)$ is replaced by its noisy version $\ell_{kj}^{(t)}(w_k)$ at time t, and negative entropy $-\mathcal{H}(p_k) = \sum_j p_{kj} \log p_{kj}$ is replaced by $D_{\mathrm{KL}}(p_k \| p_k^{(t)})$ also known as the relative entropy. We will show the above formulation leads to analytical updating rules for both w_k and p_k, a desirable property particularly as an online algorithm.

Solution for $w_k^{(t+1)}$ conditioned on $p_k^{(t)}$ is given in closed-form by the proximal operator

$$w_k^{(t+1)} = \mathbf{prox}(w_k^{(t)}) = \operatorname{argmin}_{w_k} \; C \sum_{j \in [K]} \eta_{kj}(p_k^{(t)}) \ell_{kj}^{(t)}(w_k) + \|w_k - w_k^{(t)}\|^2 \tag{2.8}$$

$$= w_k^{(t)} + C \sum_{j : y_j^{(t)} \langle x_j^{(t)}, w_k^{(t)} \rangle < 1} \eta_{kj}(p_k^{(t)}) y_j^{(t)} x_j^{(t)} \tag{2.9}$$

Solution for $p_k^{(t+1)}$ conditioned on $w_k^{(t)}$ is also given in closed-form, analogous to mirror descent [81]

$$p_k^{(t+1)} = \operatorname{argmin}_{p_k \in \Delta^{K-1}} \; C(1-\alpha) \sum_{j \in [K]} p_{kj} \ell_{kj}^{(t)} + \lambda D_{\mathrm{KL}}\big(p_k \| p_k^{(t)}\big) \tag{2.10}$$

$$\implies p_{kj}^{(t+1)} = \frac{p_{kj}^{(t)} e^{-\frac{C(1-\alpha)}{\lambda} \ell_{kj}^{(t)}}}{\sum_{j'} p_{kj'}^{(t)} e^{-\frac{C(1-\alpha)}{\lambda} \ell_{kj'}^{(t)}}} \quad j \in [K] \tag{2.11}$$

Algorithm 2: Online Algorithm (*OSMTL*-e)

for $t \in [T]$ **do**
  for $k \in [K]$ **do**
    if $y_k^{(t)} \langle x_k^{(t)}, w_k^{(t)} \rangle < 1$ **then**
      $w_k^{(t+1)} \leftarrow$
      $w_k^{(t)} + C\alpha \mathbf{1}_{\ell_{kk}^{(t)} > 0} y_k^{(t)} x_k^{(t)} + C(1-\alpha) \sum_{j:\ell_{kj}^{(t)} > 0} p_{kj}^{(t)} y_j^{(t)} x_j^{(t)}$
      ;
      for $j \in [K]$ **do**
        $p_{kj}^{(t+1)} \leftarrow \frac{p_{kj}^{(t)} e^{-\frac{C(1-\alpha)}{\lambda} \ell_{kj}^{(t)}}}{\sum_{j'=1}^{K} p_{kj'}^{(t)} e^{-\frac{C(1-\alpha)}{\lambda} \ell_{kj'}^{(t)}}}$;
      end
    else
      $w_k^{(t+1)}, p_k^{(t+1)} \leftarrow w_k^{(t)}, p_k^{(t)}$;
    end
  end
end

The pseudo-code is in Algorithm 2[1]. Our algorithm is "passive" in the sense that updates are carried out only when a classification error occurs, namely when $\hat{y}_k^{(t)} \neq y_k^{(t)}$. An alternative is to perform "aggressive" updates only when the active set $\{j : y_j^{(t)} \langle x_j^{(t)}, w_k^{(t)} \rangle < 1\}$ is non-empty.

2.1.5 Regret Bound

Theorem 1. *$\forall k \in [K]$, let $S_k = \left\{ \left(x_k^{(t)}, y_k^{(t)} \right) \right\}_{t=1}^{T}$ be a sequence of T examples for the k^{th} task where $x_k^{(t)} \in \mathbb{R}^d$, $y_k^{(t)} \in \{-1, +1\}$ and $\|x_k^{(t)}\|_2 \leq R$, $\forall t \in [T]$. Let C be a positive constant and let α be some predefined parameter in $[0, 1]$. Let $\{w_k^*\}_{k \in [K]}$ be any arbitrary vectors where $w_k^* \in \mathbb{R}^d$ and its hinge loss on the examples $\left(x_k^{(t)}, y_k^{(t)} \right)$ and $\left(x_j^{(t)}, y_j^{(t)} \right)_{j \neq k}$ are given by $\ell_{kk}^{(t)*} = \left(1 - y_k^{(t)} \langle x_k^{(t)}, w_k^* \rangle \right)_+$ and $\ell_{kj}^{(t)*} = \left(1 - y_j^{(t)} \langle x_j^{(t)}, w_k^* \rangle \right)_+$, respectively.*

If $\{S_k\}_{k \in [K]}$ is presented to OSMTL algorithm, then $\forall k \in [K]$ we have

$$\sum_{t \in [T]} (\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}) \leq \frac{1}{2C\alpha} \|w_k^*\|^2 + \frac{(1-\alpha)T}{\alpha} \left(\ell_{kk}^{(t)*} + \max_{j \in [K], j \neq k} \ell_{kj}^{(t)*} \right) + \frac{CR^2 T}{2\alpha} \tag{2.12}$$

Notice when $\alpha \to 1$, the above reduces to the perceptron mistake bound [97].

Corollary 2. *Let $\alpha = \frac{\sqrt{T}}{1+\sqrt{T}}$ and $C = \frac{1+\sqrt{T}}{T}$ in Theorem 1, we have*

$$\sum_{t \in [T]} (\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}) \leq \sqrt{T} \left(\frac{1}{2} \|w_k^*\|^2 + \ell_{kk}^{(t)*} + \max_{j \in [K], j \neq k} \ell_{kj}^{(t)*} + 2R^2 \right) \tag{2.13}$$

[1] It is recommended to set $\alpha \propto \frac{\sqrt{T}}{1+\sqrt{T}}$ and $C \propto \frac{1+\sqrt{T}}{T}$, as suggested by Corollary 2.

Proofs are given in the supplementary. Theorem 1 and Corollary 2 have several implications:

1. Quality of the bound depends on both $\ell_{kk}^{(t)*}$ and the maximum of $\{\ell_{kj}^{(t)*}\}_{j\in[K],j\neq k}$. In other words, the worst-case regret will be lower if the k^{th} true hypothesis w_k^* can well distinguish training examples in both the k^{th} task itself as well as those in all the other tasks.
2. Corollary 2 indicates the difference between the cumulative loss achieved by our algorithm and by any fixed hypothesis for task k is bounded by a term growing sub-linearly in T.
3. Corollary 2 provides a principled way to set hyperparameters to achieve the sub-linear regret bound. Specifically, recall α quantifies the self-concentration of each task. Therefore, $\alpha = \frac{\sqrt{T}}{1+\sqrt{T}} \overset{T\to\infty}{\longrightarrow} 1$ implies for large horizon it would be less necessary to rely on other tasks as available supervision for the task itself is already plenty; $C = \frac{1+\sqrt{T}}{T} \overset{T\to\infty}{\longrightarrow} 0$ suggests diminishing learning rate over the horizon length.

2.1.6 Experiments

We evaluate the performance of our algorithm under batch and online settings. All reported results in this section are averaged over 30 random runs or permutations of the training data. Unless otherwise specified, all model parameters are chosen via 5-fold cross validation.

Benchmark Datasets

We use three datasets for our experiments. Details are given below:

Landmine Detection[2] consists of 19 tasks collected from different landmine fields. Each task is a binary classification problem: landmines $(+)$ or clutter $(-)$ and each example consists of 9 features extracted from radar images with four moment-based features, three correlation-based features, one energy ratio feature and a spatial variance feature. Landmine data is collected from two different terrains: tasks 1-10 are from highly foliated regions and tasks 11-19 are from desert regions, therefore tasks naturally form two clusters. Any hypothesis learned from a task should be able to utilize the information available from other tasks belonging to the same cluster.

Spam Detection[3] We use the dataset obtained from ECML PAKDD 2006 Discovery challenge for the spam detection task. We used the task B challenge dataset which consists of labeled training data from the inboxes of 15 users. We consider each user as a single task and the goal is to build a personalized spam filter for each user. Each task is a binary classification problem: spam $(+)$ or non-spam $(-)$ and each example consists of approximately $150K$ features representing term frequency of the word occurrences. Since some spam is universal to all users (e.g. financial scams), some messages might be useful to certain affinity groups, but spam to most others. Such adaptive behavior of user's interests and dis-interests can be modeled efficiently by utilizing the data from other users to learn per-user model parameters.

Sentiment Analysis[4] We evaluated our algorithm on product reviews from amazon. The dataset contains product reviews from 24 domains. We consider each domain as a binary classifi-

[2] http://www.ee.duke.edu/~lcarin/LandmineData.zip
[3] http://ecmlpkdd2006.org/challenge.html
[4] http://www.cs.jhu.edu/~mdredze/datasets/sentiment

cation task. Reviews with rating > 3 were labeled positive $(+)$, those with rating < 3 were labeled negative $(-)$, reviews with rating = 3 are discarded as the sentiments were ambiguous and hard to predict. Similar to the previous dataset, each example consists of approximately $350K$ features representing term frequency of the word occurrences.

We choose 3040 examples (160 training examples per task) for *landmine*, 1500 emails for *spam* (100 emails per user inbox) and 2400 reviews for *sentiment* (100 reviews per domain) for our experiments. Note that we intentionally kept the size of the training data small to drive the need for learning from other tasks, which diminishes as the training sets per task become large. Since all these datasets have a class-imbalance issue (with few $(+)$ examples as compared to $(-)$ examples), we use average Area Under the ROC Curve (AUC) as the performance measure.

Batch Setting

We briefly conduct an experiment on landmine detection dataset for our batch learning to demonstrate the advantages of learning from shared data. We implement two versions of our proposed algorithm with different updates: *SMTL-t* (SMTL with thresholding updates) where $p_{kj}^{(t+1)} \propto (\lambda - \ell_{kj}^{(t)})_+$[5] and *SMTL-e* (SMTL with exponential updates) as in Algorithm 1. We compare our *SMTL** with two standard baseline methods for our batch setting: Independent Task Learning (ITL)—learning a single model for each task and Single Task Learning (STL)—learning a single classification model for pooled data from all the tasks. In addition we compare our models with *SHAMO*, which is closest in spirit with our proposed models. We select the value for λ and α for *SMTL** and M for *SHAMO* using cross validation.

Figure 2.1 (left) shows the average AUC calculated for different training size on *landmine*. We can see that the baseline results are similar to the ones reported by Xue et al. [114]. Our proposed algorithm (*SMTL**) outperforms the other baselines but when we have very few training examples (say 20 per task), the performance of STL improves as it has more examples than the others. Since η depends on the loss incurred on the data from related tasks, this loss-based measure can be unreliable for a small training sample size. To our surprise, *SHAMO* performs worse than the other models which tells us that assuming two tasks are exactly same (in the sense of hypothesis) may be inappropriate in real-world applications. Figure 2.1 (middle & left) show the task relationship matrix η for *SMTL-t* and *SMTL-e* on *landmine* when the number of training instances is 160 per task.

Online Setting

To evaluate the performance of our algorithm in the online setting, we use all three datasets (*landmine*, *spam* and *sentiment*) and compare our proposed methods to 5 baselines. We implemented two variations of Passive-Aggressive algorithm (*PA*) [33]. *PA-ITL* learns independent model for each task and *PA-ONE* builds a single model for all the tasks. We also implemented the algorithm proposed by Dekel et al. for online multitask learning with shared loss (*OSGL*) [35]. These three baselines do not exploit the task-relationship or the data from other tasks during model update. Next, we implemented two online multitask learning related to our approach: *FOML* – initializes

[5]Our algorithm and theorem can be easily generalized to other types of updating rules by replacing $\exp$ in (2.6) with other functions. In latter cases, however, $\boldsymbol{\eta}$ may no longer have probabilistic interpretations.

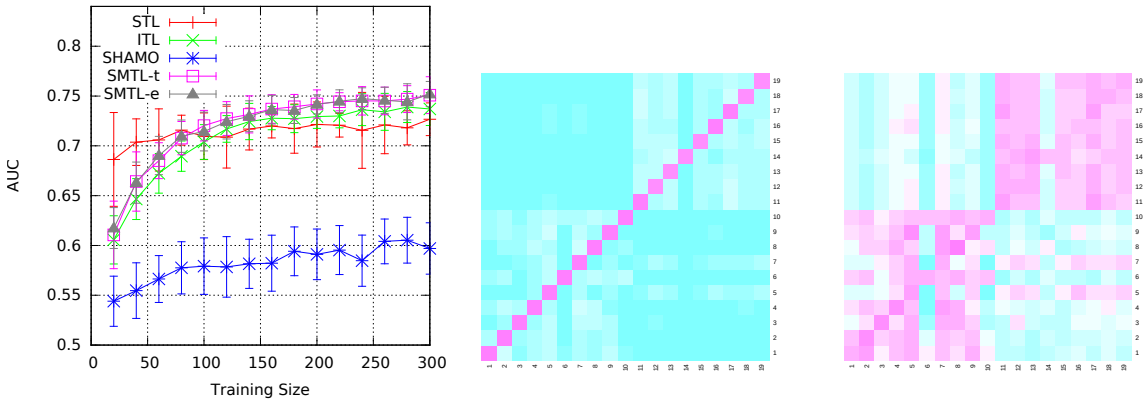

Figure 2.1: Average AUC calculated for compared models (left). A visualization of the task relationship matrix in *Landmine* learned by *SMTL-t* (middle) and *SMTL-e* (right). The probabilistic formulation of *SMTL-e* allows it to discover more interesting patterns than *SMTL-t*.

η with fixed weights [25], Online Multitask Relationship Learning (*OMTRL*) [96] – learns a task covariance matrix along with task parameters. We could not find a better way to implement the online version of the SHAMO algorithm, since the number of shared hypotheses or clusters varies over time.

Table 2.1 summarizes the performance of all the above algorithms on the three datasets. In addition to the AUC scores, we report the average total number of support vectors (*nSV*) and the CPU time taken for learning from one instance (*Time*). From the table, it is evident that *OSMTL** outperforms all the baselines in terms of both AUC and *nSV*. This is expected for the two default baselines (*PA-ITL* and *PA-ONE*). We believe that *PA-ONE* shows better result than *PA-ITL* in *spam* because the former learns the global information (common spam emails) that is quite dominant in spam detection problem. The update rule for *FOML* is similar to ours but using fixed weights. The results justify our claim that making the weights adaptive leads to improved performance.

In addition to better results, our algorithm consumes less or comparable CPU time than the baselines which take into account inter-task relationships. Compared to the *OMTRL* algorithm that recomputes the task covariance matrix every iteration using expensive SVD routines, the adaptive weights in our are updated independently for each task. As specified in [96], we learn the task weight vectors for *OMTRL* separately as K independent perceptron for the first half of the training data available (*EPOCH*=0.5). *OMTRL* potentially looses half the data without learning task-relationship matrix as it depends on the quality of the task weight vectors.

It is evident from the table that algorithms which use loss-based update weights $\boldsymbol{\eta}$ (*OSGL*, *OSMTL**) considerably outperform the ones that do not use it (*FOML*,*OMTRL*). We believe that loss incurred per instance gives us valuable information for the algorithm to learn from that instance, as well as to evaluate the inter-dependencies among tasks. That said, task relationship information does help by learning from the related tasks' data, but we demonstrate that combining both the task relationship and the loss information can give us a better algorithm, as is evident from our experiments.

We would like to note that our proposed algorithm *OSMTL** does exceptionally better in

Table 2.1: Average performance on three datasets: means and standard errors over 30 random shuffles.

Models	Landmine Detection			Spam Detection			Sentiment Analysis		
	AUC	*nSV*	*Time (s)*	*AUC*	*nSV*	*Time (s)*	*AUC*	*nSV*	*Time (s)*
PA-ONE	0.5473 (0.12)	2902.9 (4.21)	0.01	0.8739 (0.01)	1455.0 (4.64)	0.16	0.7193 (0.03)	2350.7 (6.36)	0.19
PA-ITL	0.5986 (0.04)	618.1 (27.31)	0.01	0.8350 (0.01)	1499.9 (0.37)	0.16	0.7364 (0.02)	2399.9 (0.25)	0.16
OSGL	0.6482 (0.03)	740.8 (42.03)	0.01	0.9551 (0.007)	1402.6 (13.57)	0.17	0.8375 (0.02)	2369.3 (14.63)	0.17
FOML	0.6322 (0.04)	426.5 (36.91)	0.11	0.9347 (0.009)	819.8 (18.57)	1.5	0.8472 (0.02)	1356.0 (78.49)	1.20
OMTRL	0.6409 (0.05)	432.2 (123.81)	6.9	0.9343 (0.008)	840.4 (22.67)	53.6	0.7831 (0.02)	1346.2 (85.99)	128
OSMTL-t	**0.6776** (0.03)	333.6 (40.66)	0.18	0.9509 (0.007)	809.5 (19.35)	1.4	0.9354 0.01	1312.8 (79.15)	2.15
OSMTL-e	0.6404 (0.04)	458 (36.79)	0.19	**0.9596** (0.006)	804.2 (19.05)	1.3	**0.9465** (0.01)	1322.2 (80.27)	2.16

sentiment, which has been used as a standard benchmark application for domain adaptation experiments in the existing literature [18]. We believe the advantageous results on *sentiment* dataset implies that even with relatively few examples, effectively knowledge transfer among the tasks/domains can be achieved by adaptively choosing the (probabilistic) inter-task relationships from the data.

2.2 Multitask Multiple Kernel Relationship Learning

There have been two main line of work in multitask learning: First, learn a shared feature representation across all the tasks, leveraging low-dimensional subspaces in the feature space [9, 52, 72, 104]. Second, learn the relationship between the tasks to improve the performance of the related tasks [92, 114, 119, 120]. Pairwise task relationships such as positive task correlation, negative task correlation and task independence provide very useful information for characterizing and transferring information to similar tasks.

Despite the expressive power of these two different research directions, the learning space is restricted to a single kernel (per task), chosen by the user, that corresponds to a $RKHS$ space. Multiple Kernel Learning (*MKL*), on the other hand, allows the user to specify a family of base kernels related to an application, and to use the training data to automatically learn the optimal combination of these kernels. We learn the weights of the base kernels along with the model parameters in a single joint optimization problem. There is a large body of work in the recent years addressing several aspects of this problem, such as efficient learning of the kernel weights, fast optimization and providing better theoretical guarantees [10, 28, 58, 59, 66, 88, 103].

Recent work in multiple kernel learning in a multitask framework focuses on sharing common representations and assumes that the tasks are all related [53]. The motivation for this approach stems from multitask feature learning that learns joint feature representation shared across multiple

tasks [9, 104]. Unfortunately, the assumption that all the tasks are related and share a common feature representation is too restrictive for many real-world applications. Similarly, based on previous work [120], one can extend the traditional multitask relationship learning *MTRL* with multiple task-specific base kernels. There are two main problems with such naive approach: First, the unknown variables (task model parameters, kernel weights and task relationship matrix) are intertwined in the optimization problem, and thus making it difficult to learn for large-scale applications. Furthermore, the task relationship matrix is learned in the original feature space rather than in the kernel spaces. We show in this chapter, that learning the relationship between the kernel spaces empirically performs better than learning the relations among the original feature spaces.

There have been a few attempts to imposing higher-order relationship between kernel spaces using kernel weights. Kloft et al. [59] propose a non-isotropic norm such as $\sqrt{\boldsymbol{\beta}^\top \Sigma^{-1} \boldsymbol{\beta}}$ on kernels weights $\boldsymbol{\beta}$ to induce the relationship between the base kernels in Reproducing Kernel Hilbert Spaces. For example, in neuroimaging, a set of base kernels are derived from several medical imaging modalities such as MRI, PET etc., or image processing methods such as morphometric or anatomical modeling, etc. Since some of the kernel functions share similar parameters such as patient information, disease progression stage, etc., we can expect that these base kernels are correlated based on how they were constructed. Such information can be obtained from medical domain experts as a part of the disease prognosis which then can be used as a prior knowledge Σ. Previous work either assumes Σ as a diagonal matrix or requires prior knowledge from the experts on the interaction of kernels [50, 59]. Unfortunately, such prior knowledge is not easily available in many applications either because it is time-consuming or it is expensive to elicit. [62]. In such applications, we want to induce this relationship matrix from the data along with the kernel weights and model parameters.

In this section, we address these problems with a novel regularization-based approach for multitask multiple kernel learning framework, called *multitask multiple kernel relationship learning* (*MK-MTRL*), which models the task relationship matrix from the weights learned from the latent feature spaces of task-specific base kernels. The idea is to automatically infer task relationships in (*RKHS*) spaces from their base kernels. We first propose an alternating minimization algorithm to learn the model parameters, kernel weights and task relationship matrix. The method uses a *wrapper* approach which efficiently uses any off-the-shelf *SVM* solver (or any kernel machine) to learn the task model parameters. However, like previous work, the proposed iterative algorithm suffers from scalability challenges. The run-time complexity of the algorithm increases with the number of tasks and the number of base kernels per task, as it needs these base kernels in memory to learn the kernel weights and the task relationship matrix.

For large-scale applications such as object detection, we introduce a novel two-stage online learning algorithm following the recent work [64] that learns the kernel weights independently from the model parameters. The first stage learns a good combination of base kernels in an online setting and the second stage uses the learned weights to estimate a linear combination of the base kernels, which can be readily used with a standard kernel method such as *SVM* or kernel ridge regression [29, 34]. We provide strong empirical evidence that learning the task relationship matrix in the RKHS space is beneficial for many applications such as stock market prediction, visual object categorization, etc. On all these applications, our proposed approach outperforms

several state-of-the-art multitask learning baselines. It is worth noting that the proposed multitask multiple kernel relationship learning can be readily applied for heterogeneous and multi-view data with no modification to the proposed framework [49, 118].

The rest of the chapter is organized as follows: we provide a brief overview of multitask multiple kernel learning in the next section. In section 2.2.2, we discuss the proposed model *MK-MTRL*, followed by our two-stage online learning approach in section 2.2.5. We then show comprehensive evaluations of the proposed model against the six baselines on several benchmark datasets in section 2.2.7.

2.2.1 Preliminaries

Before introducing our approach, we briefly review the multitask multiple kernel learning framework in this section. Suppose there are T learning tasks available with the training set $\mathcal{D} = \{(\mathbf{x}_{ti}, \boldsymbol{y}_{ti}), i = 1 \dots N_t, t = 1 \dots T\}$, where $\boldsymbol{x}_{ti}$ is the i^{th} samples from the task t and it's corresponding output $\boldsymbol{y}_{ti}$. Let $\{\mathcal{K}_{tk}\}^{1 \leq k \leq K}$ be a set of task-specific base kernels, induced by the kernel mapping function $\phi_k(\cdot)$ on t^{th} task data. The objective of multitask multiple kernel learning problem is to learn a *good* linear combination of the task-specific base kernels $\sum_k \beta_{tk}\mathcal{K}_{tk}, \beta_{tk} \geq 0$ using the relationship between the tasks.

In addition to the non-negative constraints on β_{tk}, we need to impose an additional constraint or penalty to ensure that the units in which the margins are measured are meaningful (assuming that the base kernels are properly normalized). Recent work in *MKL* employs $\|\boldsymbol{\beta}\|_2^2$ to constrain the kernel weights. A direct extension of ℓ_2 regularized *MKL* to multitask framework is given as follows [6]:

$$\begin{aligned} &\min_{\mathbf{B} \geq 0} \min_{\mathbf{W}, \mathbf{c}, \xi} \sum_{t=1}^{T} \left(\frac{1}{2} \sum_{k=1}^{K} \frac{\|\mathbf{w}_{tk}\|_{\mathcal{H}_k}^2}{\beta_{tk}} + C \sum_{i=1}^{N_t} \xi_{ti} + \frac{\mu}{2} \|\boldsymbol{\beta}_t\|_2^2 \right) \\ &\text{s.t., } y_{ti}(\sum_{k=1}^{K} w_{tk}^{\top} \phi_k(\mathbf{x}_{ti}) + c_t) \geq 1 - \xi_{ti}, \ \xi_{ti} \geq 0 \end{aligned} \tag{2.14}$$

Similarly, we can use a general ℓ_p norm constraint with $p > 1$ on the kernel weights ($\|\boldsymbol{\beta}\|_p^2$). This can be thought of as a simple extension of ℓ_p-*MKL* to multitask setting [58]. Without any additional structural constraints on β_{tk}, the kernel weights are learned independently for each task and thus does not efficiently use the relationship between the tasks. Hence, we call the model in equation (2.14) as Independent Multiple Kernel Learning (*IMKL*).

Jawanpuria and Nath [53] proposed Multitask Multiple Kernel Feature Learning (*MK-MTFL*), that employs mixed $(\ell_1, \ell_p), p \geq 2$ norm regularizer over the *RKHS* norm of the feature loadings corresponding to the tasks and the base kernels. The mixed norm regularization promotes a shared feature representations to combine the given set of task-specific base kernels. The ℓ_p-norm regularizer learns the unequal weighting across the tasks, where as, ℓ_1-norm regularizer over the ℓ_p-norm leads to learning the shared kernel among the tasks.

[6] For clarity, we use binary classification tasks to explain the preliminaries and the proposed approach. They can be easily applied to multiclass tasks and also to regression tasks via kernel ridge regression.

The objective function for *MK-MTFL* is given as follows:

$$\begin{aligned} &\min_{\mathbf{W},\mathbf{c},\xi} \left(\frac{1}{2} \sum_{k=1}^{K} \big(\sum_{t=1}^{T} \|\mathbf{w}_{tk}\|_2^p \big)^{\frac{1}{p}} \right)^2 + C \sum_{t=1}^{T} \sum_{i=1}^{N_t} \xi_{ti} \\ &\text{s.t., } y_{ti} \big(\sum_{k=1}^{K} w_{tk}^\top \phi_k(\mathbf{x}_{ti}) + c_t \big) \geq 1 - \xi_{ti}, \xi_{ti} \geq 0 \end{aligned} \tag{2.15}$$

Note that the above objective function employs an ℓ_1-norm across the base kernels and ℓ_p norm across tasks. The above optimization problem can be equivalently written in the dual space as follows:

$$\begin{aligned} &\min_{\gamma \in \Delta_K} \max_{\lambda_j \in \Delta_{T,\tilde{p}}} \max_{0 \leq \alpha \leq C} g(\lambda, \alpha, \gamma) \\ &\text{s.t., } \alpha_t^\top \mathbf{y}_t = 0, \end{aligned} \tag{2.16}$$

where,

$$g(\lambda, \alpha, \gamma) = \sum_{t=1}^{T} \Big\{ \mathbf{1}^\top \alpha_t - \frac{1}{2} \alpha_t^\top \mathbf{Y}_t \big[\sum_{k=1}^{K} \frac{\gamma_k \mathcal{K}_{tk}}{\lambda_{tk}} \big] \mathbf{Y}_t \alpha_t \Big\}$$

Here $\boldsymbol{\alpha}_t$ is a vector of Langragian multipliers for the t^{th} task, and corresponds to N_t constraints on the task data. $\boldsymbol{Y}_t$ is a diagonal matrix with entries as $\boldsymbol{y}_t$ and $\mathcal{K}_{tk}$ is the gram matrix of the t^{th} task data w.r.t the k^{th} kernel function. More specifically, $\boldsymbol{\gamma}$ selects the base kernels that are important for all the tasks, where as $\boldsymbol{\lambda}$ selects the base kernels that are specific to individual tasks. With this representation, *MK-MTFL* can be seen as a multiple kernel generalization to the multi-level multitask learning proposed by Lozano and Swirszcz (2012) [104].

2.2.2 Proposed Method (*MK-MTRL*)

This section presents the details of the proposed model *MK-MTRL*. Since multitask learning seeks to improve performance of each task with the help of other *related* tasks, it is desirable in multiple kernel learning for the multitask framework to have a structural constraints on the task kernel weights β_{tk} to promote sharing of information from other related tasks. Note that the proposed approach is significantly different from the traditional *MTRL*, as explained in the introduction.

When prior knowledge on task relationship is available, the multiple kernel multitask learning model should incorporate this information for simultaneously learning several related tasks. Neither the *IMKL* or *MK-MTFL* consider the pairwise task relationship such as positive task correlation, negative task correlation, and task independence when learning the kernel weights for combining the base kernels. Based on the assumption that similar tasks are likely to give similar importance to their base kernels (and thereby, their respective *RKHS* spaces), we consider a regularization on the task kernel weights $tr(\boldsymbol{B}\Omega^{-1}\boldsymbol{B}^\top)$, where, for notational convenience, we write $\boldsymbol{B} = \{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_T\}$. Mathematically, the proposed *MK-MTRL* formulation is written as

follows:

$$\begin{aligned}\min_{\mathbf{\Omega},\mathbf{B}\succeq 0}\min_{\mathbf{W},\mathbf{c},\xi}&\sum_{t=1}^{T}\left(\frac{1}{2}\sum_{k=1}^{K}\frac{\|\mathbf{w}_{tk}\|^2_{\mathcal{H}_k}}{\beta_{tk}}+C\sum_{i=1}^{N_t}\xi_{ti}\right)\\&+\frac{\mu}{2}tr(\mathbf{B}\mathbf{\Omega}^{-1}\mathbf{B}^\top)\\\text{s.t., }y_{ti}(&\sum_{k=1}^{K}w_{tk}^\top\phi_k(\mathbf{x}_{ti})+c_t)\geq 1-\xi_{ti},\xi_{ti}\geq 0\\&\mathbf{\Omega}\succeq 0,\\&tr(\mathbf{\Omega})\leq 1\end{aligned}\tag{2.17}$$

The key difference from the *IMKL* model is that the standard (squared) ℓ_p norm on $\boldsymbol{\beta}_t$ is replaced with a more meaningful structural penalty that incorporates the task relationship. Unlike in *MK-MTFL*, the shared information among the task is separate from the core problem (T SVMs). Here, $\mathbf{\Omega}$ encodes the task relationship such that similar tasks are forced to have similar kernel weights. It is easy to see that when $\mathbf{\Omega}=\mathbb{I}_{T\times T}$, the above problem reduces to equation (2.14).

2.2.3 *MK-MTRL* in Dual Space

In this section, we consider the proposed approach in the dual space. By writing the above objective function in Lagrangian form and introducing Lagrangian multiplier α_{tk} for the constraints, we can write the corresponding dual objective function as:

$$\begin{aligned}\min_{\mathbf{\Omega},\mathbf{B}\succeq 0}\max_{0\leq\alpha\leq C}&h(\alpha,\mathbf{B})+\frac{\mu}{2}tr(\mathbf{B}\mathbf{\Omega}^{-1}\mathbf{B}^\top)\\&\text{s.t., }\alpha_t^\top\mathbf{y}_t=0,\\&\mathbf{\Omega}\succeq 0,\\&tr(\mathbf{\Omega})\leq 1\end{aligned}\tag{2.18}$$

where,

$$h(\alpha,\mathbf{B})=\sum_{t=1}^{T}\left\{\mathbf{1}^\top\alpha_t-\frac{1}{2}\alpha_t^\top\mathbf{Y}_t\Big(\sum_{k=1}^{K}\beta_{tk}\mathcal{K}_{tk}\Big)\mathbf{Y}_t\alpha_t\right\}$$

Note that we can further reduce the problem by eliminating α_{tk}, then the dual problem becomes:

$$\begin{aligned}\min_{\mathbf{\Omega}}\max_{0\leq\alpha\leq C}&\sum_{t=1}^{T}\left\{\mathbf{1}^\top\alpha_t-\frac{1}{2}\|\mathcal{G}\|_\Omega\right\}\\&\text{s.t., }\alpha_t^\top\mathbf{y}_t=0,\\&\mathbf{\Omega}\succeq 0,\\&tr(\mathbf{\Omega})\leq 1\end{aligned}\tag{2.19}$$

where $\mathcal{G}_{tk}=\beta_{tk}\alpha_t^\top\mathbf{Y}_t\mathcal{K}_{tk}\mathbf{Y}_t\alpha_t$ which corresponds to $\frac{\|\boldsymbol{w}_{tk}\|_2^2}{\beta_{tk}}$ in the primal space and we write $\|\mathcal{G}\|_\Omega=\sqrt{tr(\mathcal{G}\mathbf{\Omega}\mathcal{G}^\top)}$. We will use this representation for deriving closed-form solution for the task kernel weights $\boldsymbol{B}$

2.2.4 Optimization

We use an alternating minimization procedure for learning the kernel weights and the model parameters iteratively. We implement a two-layer *wrapper* approach commonly used in these *MKL* solvers for our problem. The wrapper methods alternate between minimizing the primal problem (2.17) w.r.t $\boldsymbol{\beta}_t$ via a simple analytical update step and minimizing all other variables in terms of the dual variables $\boldsymbol{\alpha}_t$ from equation (2.18).

When $\{\boldsymbol{B}, \boldsymbol{\Omega}\}$ are fixed, *MK-MTRL* equation (2.18) reduces to T independent sub-problems. One can use any conventional *SVM* solver (or any kernel method) to optimize for $\boldsymbol{\alpha}_t$ independently. We focus on optimizing the kernel coefficients $\boldsymbol{B}$ and $\boldsymbol{\Omega}$ next.

Optimizing w.r.t $\boldsymbol{B}$ when $\{\boldsymbol{\alpha}, \boldsymbol{\Omega}\}$ are fixed

Given $\{\boldsymbol{\alpha}, \boldsymbol{\Omega}\}$, we find $\boldsymbol{B}$ by setting the gradient of equation (2.17) w.r.t $\boldsymbol{B}$ to zero and we get:

$$\boldsymbol{B} = \frac{1}{\mu}(\mathcal{W} \circ \boldsymbol{B}^{-2})\boldsymbol{\Omega} \tag{2.20}$$

where $\boldsymbol{B}^{-2} = \{\beta_{kt}^{-2}, 1 \leq k \leq K, 1 \leq t \leq T\}$, $\mathcal{W}_{tk} = \|\mathbf{w}_{tk}\|^2_{\mathcal{H}_k}$ and $\boldsymbol{A} \circ \boldsymbol{B}$ is an element-wise product operation.

By incorporating the last term in equation (2.17) into the constraint set, we can eliminate the regularization parameter μ to obtain an analytical solution for $\boldsymbol{B}$. Because $\boldsymbol{\Omega} \succeq 0$ and $\boldsymbol{B} \geq 0$, the constraint $tr(\boldsymbol{B}\Omega^{-1}\boldsymbol{B}^\top) \leq 1$ must be active at optimality. We can now use the above equation to solve for μ.

$$\boldsymbol{B} = \frac{(\mathcal{W} \circ \boldsymbol{B}^{-2})\boldsymbol{\Omega}}{\sqrt{tr((\mathcal{W} \circ \boldsymbol{B}^{-2})\boldsymbol{\Omega}(\mathcal{W} \circ \boldsymbol{B}^{-2})^\top)}} \tag{2.21}$$

Since the task relationship matrix is independent of the number of base kernels K, one may use the above closed-form solution when the number of tasks is small. For some applications, it may be desirable to employ an iterative approach such as first-order method (*FISTA*) or second-order method (*Newton's*). The parameter μ can be easily learned by cross-validation.

Optimizing w.r.t $\boldsymbol{\Omega}$ when $\{\boldsymbol{\alpha}, \boldsymbol{B}\}$ are fixed

In the final step of the optimization, we fix $\boldsymbol{\alpha}$ and $\boldsymbol{B}$ and solve the problem w.r.t $\boldsymbol{\Omega}$. By taking the partial derivative of the objective function with respect to $\boldsymbol{\Omega}$ and setting it zero, we get an analytical solution for $\boldsymbol{\Omega}$[120]:

$$\boldsymbol{\Omega} = \frac{(\boldsymbol{B}^\top \boldsymbol{B})^{\frac{1}{2}}}{tr((\boldsymbol{B}^\top \boldsymbol{B})^{\frac{1}{2}})} \tag{2.22}$$

Substituting the above solution in equation 2.17, we can see that the the objective function of *MK-MTRL* is related to the trace norm regularization. Instead of ℓ_p norm regularization (as in L_p-MKL) or mixed-norm regularization (as in *MK-MTFL*), our model seeks a low-rank $\boldsymbol{B}$, using $\|\boldsymbol{B}\|_*$, such that similar base kernels are selected among the similar tasks.

2.2.5 Two-Stage *MK-MTRL*

The proposed optimization procedure in the previous section involves T independent SVM (or *kernel ridge regression*) calls, followed by two closed-form expressions for jointly learning the kernel weights $\boldsymbol{B}$, task relationship matrix $\boldsymbol{\Sigma}$ and the task parameters $\boldsymbol{\alpha}$. Even though this approach is simple and easy to implement, it requires the precomputed kernel matrices to be loaded into memory for learning the kernel weights. This could add a serious computational burden especially when the number of tasks T is large [112].

In this section, we consider an alternative approach to address this problem inspired by [29, 34]. It follows a two-stage approach: first, we independently learn the weights of the given task-specific base kernels using the training data and then, we use the weighted sum of these base kernels in a standard kernel machines such as SVM or *kernel ridge regression* to obtain a classifier. This approach significantly reduces the amount of computational overhead involved in the traditional multiple kernel learning algorithms that estimate the kernel weights and the classifier by solving a joint optimization problem.

We propose an efficient binary classification framework for learning the weights of these task-specific base kernels, based on target alignment [34]. The proposed framework formulates the kernel learning problem as a linear classification in the kernel space (so called $\mathcal{K}$-classifier). In this space, any task classifier with weight parameters directly corresponds to the task kernel weights.

For a given set of $T * K$ base kernels $\{\mathcal{K}_{tk}\}_{1\leq t\leq T}^{1\leq k\leq K}$ (K base kernels per task), we define a binary classification framework over a new instance space (so called $\mathcal{K}$-space) defined as follows:

$$\begin{aligned} z_{t,ii'} &= \{\mathcal{K}_1(\mathbf{x}_{ti}, \mathbf{x}_{ti'}), \mathcal{K}_2(\mathbf{x}_{ti}, \mathbf{x}_{ti'}), \ldots, \mathcal{K}_K(\mathbf{x}_{ti}, \mathbf{x}_{ti'})\} \\ l_{t,ii'} &= 2.\mathbf{1}\{y_{ti} = y_{ti'}\} - 1 \end{aligned} \tag{2.23}$$

Any hypothesis $h_t : \mathbb{R}^K \rightarrow \mathbb{R}$ for a task t induces a similarity function $\tilde{\mathcal{K}}_{h_t}(\mathbf{x}_{ti}, \mathbf{x}_{ti'})$ between instances $\mathbf{x}_{ti}$ and $\mathbf{x}_{ti'}$ in the original space:

$$\begin{aligned} \tilde{\mathcal{K}}_{h_t}(\mathbf{x}_{ti}, \mathbf{x}_{ti'}) &= h_t(z_{t,ii'}) \\ &= h_t(\mathcal{K}_1(\mathbf{x}_{ti}, \mathbf{x}_{ti'}), \ldots, \mathcal{K}_K(\mathbf{x}_{ti}, \mathbf{x}_{ti'})) \end{aligned}$$

Suppose we consider a linear function for our task hypothesis $h_t(z_{t,ii'}) = \boldsymbol{\beta}_t.z_{t,ii'}$ with the non-negative constraints $\boldsymbol{\beta}_t \geq 0$, then the resulting induced kernel $\tilde{\mathcal{K}}_{h_t}$ is also positive semi-definite. The key idea behind this two-stage approach is that if a $\mathcal{K}$-classifiers h_t is a good classifier in the $\mathcal{K}$-space, then the induced kernel $\tilde{\mathcal{K}}_{h_t}(\mathbf{x}_{ti}, \mathbf{x}_{ti'})$ will likely be positive when $\mathbf{x}_{ti}$ and $\mathbf{x}_{ti'}$ belong to the same class and negative otherwise. Thus the problem of learning a good combination of base kernels can be framed as a problem of learning a good $\mathcal{K}$-classifier.

With this framework, the optimization problem for learning $\boldsymbol{\beta}_t$ for each task t can be formulated as follows:

$$\begin{aligned} &\min_{\mathbf{B}\geq 0} \sum_{t=1}^{T} \ell(l_{t,ii'}, \langle \beta_t, z_{t,ii'} \rangle) + \frac{\mu}{2}\mathcal{R}(\mathbf{B}) \\ &\ell(l_{t,ii'}, \langle \beta_t, z_{t,ii'} \rangle) = \frac{1}{\binom{N_t}{2} + N_t} \sum_{1\leq i\leq i'\leq N_t} \left[1 - l_{t,ii'}\beta_t z_{t,ii'}\right]_+ \end{aligned} \tag{2.24}$$

where $[1-s]_+ = \max\{0, 1-s\}$ and $\mathcal{R}(\mathbf{B})$ is the regularization function on the kernel weights $\boldsymbol{B}$. Since we are interested in learning task relationships using the task kernel weights $\boldsymbol{\beta}_t$, we can directly extend the above formulation to incorporate the regularization on $\boldsymbol{\beta}_t$ based on *MK-MTRL*.

$$\begin{aligned} \min_{\boldsymbol{\Omega}} \min_{\mathbf{B} \geq 0} & \sum_{t=1}^{T} \ell(l_{t,ii'}, \langle \beta_t, z_{t,ii'} \rangle) + \frac{\mu}{2} tr(\mathbf{B}\boldsymbol{\Omega}^{-1}\mathbf{B}^\top) \\ & \boldsymbol{\Omega} \succeq 0, \\ & tr(\boldsymbol{\Omega}) \leq 1 \end{aligned} \tag{2.25}$$

Since the above objective function depends on every pair of observations, we consider an online learning procedure for faster computation that learns the kernel weights and the task relationship matrix sequentially. Note that with the above formulation, one can easily extend the existing approach to jointly learn both the feature and task relationship matrices using matrix normal penalty [119].

2.2.6 Algorithms

Algorithm 3 shows the pseudo-code for *MK-MTRL*. It outlines the update steps explained in Section 3. The algorithm alternates between learning the model parameters, kernel weights and task relationship matrix until it reaches the maximum number of iterations [7] or when there are minimal changes in the subsequent $\boldsymbol{B}$.

The two-stage, online learning of *MK-MTRL* is given in Algorithm 4. The online learning of $\boldsymbol{\beta}_t$ and $\boldsymbol{\Omega}$ is based on the recent work by Saha et al., 2011 [96]. We set the maximum number of rounds to $100,000$. Since we construct the examples in kernel space on the fly, there is no need keep the base kernel matrices in memory. This significantly reduces the computational burden required in computing $\boldsymbol{B}$.

We use *libSVM* to solve the T individual SVMs (equation 2.26). All the base kernels are normalized to unit trace. Note that equation 2.28 requires computing Singular Value Decomposition (SVD) on $(\boldsymbol{B}^\top \boldsymbol{B})$. One may use an efficient decomposition algorithm such as the randomized SVD to speed up the learning process [71].

2.2.7 Experiments

We evaluate the performance of our proposed model on several benchmark datasets. We compare our proposed model with five state-of-the-art baselines in multitask learning and in multitask multiple kernel learning. All reported results in this section are averaged over 10 random runs of the training data. Unless otherwise specified, all model parameters are chosen via 5-fold cross validation. The best model and models with statistically comparable results are shown in bold.

Compared Models

We compare the following models for our evaluation.

[7] *maxIter* is set to 50

Algorithm 3: Wrapper method for Multitask Multiple Kernel Relationship Learning (*MK-MTRL*)

Input : $Base\ kernels\ \{\mathcal{K}_{tk}\}_{1\leq t\leq T}^{1\leq k\leq K}$, $labels\ \{\boldsymbol{y}_t)\}_{t=1}^{T}$, $regularization\ \ parameter\ \mu > 0$

Output : $\boldsymbol{\alpha}, \boldsymbol{B}, \boldsymbol{\Omega}$

Initialize $\boldsymbol{\Omega} = \frac{1}{T}\mathbb{I}_{T\times T}$;

repeat

  repeat

    Set $\mathcal{K}_t \leftarrow \sum_{k=1}^{K} \beta_{tk}\mathcal{K}_{tk}, \forall t \in [T]$;

    Solve for $\boldsymbol{\alpha}_t, t \in [T]$

$$\max_{0\leq\boldsymbol{\alpha}_t\leq C, \boldsymbol{\alpha}_t^\top \boldsymbol{y}_t=0} \left\{\mathbf{1}^\top \alpha_t - \frac{1}{2}\alpha_t^\top \mathbf{Y}_t\mathcal{K}_t\mathbf{Y}_t\alpha_t\right\} \ (SVM) \qquad (2.26)$$

    Solve for $\boldsymbol{B} = \{\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_T\}$,

$$\min_{\mathbf{B}\geq 0} \frac{1}{2}\sum_{t=1}^{T}\sum_{k=1}^{K}\frac{\|\mathbf{w}_{tk}\|_{\mathcal{H}_k}^2}{\beta_{tk}} + \frac{\mu}{2}tr(\mathbf{B}\boldsymbol{\Omega}^{-1}\mathbf{B}^\top) \qquad (2.27)$$

    where $\|\mathbf{w}_{tk}\|_{\mathcal{H}_k}^2 = \beta_{tk}^2\alpha_t^\top\mathbf{Y}_t\mathcal{K}_{tk}\mathbf{Y}_t\alpha_t$

  until *converges*;

  Solve for $\boldsymbol{\Omega}$,

$$\min_{\boldsymbol{\Omega}\succeq\mathbf{0}, tr(\boldsymbol{\Omega})\leq 1} tr(\boldsymbol{\Omega}^{-1}(\boldsymbol{B}^\top\boldsymbol{B})) \qquad (2.28)$$

until *converges*;

- Single-Task Learning (STL) learns the tasks independently. STL uses either *SVM* (in case of binary classification tasks) or Kernel Ridge regression (in case of regression tasks) to learn the individual models.
- Multitask Feature Learning (MTFL [9]) learns a shared feature representation from all the tasks using regularization. It learns this shared feature representation along with the task model parameters alternatively[8].
- Multitask Relationship Learning (MTRL [120]) learns task relationship matrix under a regularization framework. This model can be viewed as a multitask generalization for single-task learning. It learns the task relationship matrix and the task parameters in an iterative fashion[9].
- Single-task Multiple Kernel Learning (IMKL) learns independent MKL for each task. This baseline does not use any shared information between the tasks. We use ℓ_p-MKL for each

[8]The source code for this baseline is available at http://ttic.uchicago.edu/~argyriou/code/mtl_feat/mtl_feat.tar

[9]The source code for this baseline is available at https://www.cse.ust.hk/~zhangyu/codes/MTRL.zip

Algorithm 4: Two-stage, online learning of (*MK-MTRL*)

Input : *Base kernels* $\{\mathcal{K}_{tk}\}_{1\le t\le T}^{1\le k\le K}$, *labels* $\{\boldsymbol{y}_t\}_{t=1}^{T}$, *regularization parameter* $\mu > 0$, Number of rounds R

Output : $\boldsymbol{\alpha}, \boldsymbol{B}, \boldsymbol{\Omega}$

Initialize $\boldsymbol{\beta}_t^{(1)} = \mathbf{0}, \boldsymbol{\Omega} = \frac{1}{T}\mathbb{I}_{T\times T}$;

for $r = 1 \dots R$ **do**

Construct $(z_{t,ii'}, l_{t,ii'})$ using $\mathcal{K}$ for any two examples $(\mathbf{x}_{ti}, y_{ti})$ and $(\mathbf{x}_{ti'}, y_{ti'})$ and for any task t, where

$$\begin{aligned} z_{t,ii'} &= \{\mathcal{K}_1(\mathbf{x}_{ti}, \mathbf{x}_{ti'}), \mathcal{K}_2(\mathbf{x}_{ti}, \mathbf{x}_{ti'}), \\ &\quad \dots, \mathcal{K}_K(\mathbf{x}_{ti}, \mathbf{x}_{ti'})\} \\ l_{t,ii'} &= 2.\mathbf{1}\{y_{ti} = y_{ti'}\} - 1 \end{aligned} \tag{2.29}$$

Predict $\hat{l}_{t,ii'} = \boldsymbol{\beta}_t^{(r)\top} z_{t,ii'}$

if $(l_{t,ii'} \neq \hat{l}_{t,ii'})$ **then**

for $t' = 1 \dots T$ **do**

$\beta_{t'}^{(r+1)} = \beta_{t'}^{(r)} + \frac{1}{\mu} l_{t,ii'} \boldsymbol{\Omega}_{t,t'} z_{t,ii'}$

Solve for $\boldsymbol{\Omega}$,

$$\min_{\boldsymbol{\Omega}\succeq\mathbf{0}, tr(\boldsymbol{\Omega})\le 1} tr(\boldsymbol{\Omega}^{-1}(\boldsymbol{B}^\top \boldsymbol{B})) \tag{2.30}$$

Set $\mathcal{K}_t \leftarrow \sum_{k=1}^{K} \beta_{tk}^{(R)} \mathcal{K}_{tk}, \forall t \in [T]$;

Solve for $\boldsymbol{\alpha}_t, t \in [T]$

$$\max_{0\le\boldsymbol{\alpha}_t\le C, \boldsymbol{\alpha}_t^\top \boldsymbol{y}_t = 0} \left\{ \mathbf{1}^\top \alpha_t - \frac{1}{2} \alpha_t^\top \mathbf{Y}_t \mathcal{K}_t \mathbf{Y}_t \alpha_t \right\} \ (SVM) \tag{2.31}$$

task. We tune the value of p from $[2, 3, 4, 6, 8.67]$ using 5-fold cross validation.

- Multitask Multiple Kernel Feature Learning (MK-MTFL [53]) learns a shared kernel for feature representation from all tasks. This is a multiple kernel generalization of multitask feature learning problem. Again, we tune the value of $\tilde{p}$ from $[2, 3, 4, 6, 8.67]$ using 5-fold cross validation[10].

Unless otherwise specified, the kernels for *STL*, *MTFL* and *MTRL* are chosen (via cross validation) from either a Gaussian RBF kernel with different bandwidth or a linear kernel for each dataset. The value for C is chosen from $[10^{-3}, \dots, 10^3]$. We tune the value of μ from $[10^{-7}, \dots, 10^3]$. We use *Newton's* method to learn the task kernel weight matrix $\boldsymbol{B}$. We compare our models on several applications: Asset Return Prediction, Landmine Detection and Object Recognition. Note that different applications require different types of base kernels. There is no common set of kernel functions that will work for all applications. We choose these base kernels

[10]The source code for this baseline is available at http://www.cse.iitb.ac.in/saketh/research/MTFL.tgz

Table 2.2: Mean Squared Error (MSE) for each company ($\times 1000$)

	OLS	Lasso	MRCE	FES	STL	IKL	IMKL	MK-MTFL	MK-MTRL
Walmart	0.98	0.42	0.41	0.40	0.44	0.43	0.45	0.44	0.44
Exxon	0.39	0.31	0.31	0.29	0.34	0.32	0.33	0.32	0.32
GM	1.68	0.71	0.71	0.62	0.82	0.62	0.60	0.61	**0.56**
Ford	2.15	0.77	0.77	0.69	0.91	0.56	0.53	0.55	**0.49**
GE	0.58	0.45	0.45	0.41	0.43	0.41	0.40	0.40	**0.39**
ConocoPhillips	0.98	0.79	0.79	0.79	0.84	0.81	0.83	0.80	0.80
Citigroup	0.65	0.66	0.62	0.59	0.64	0.66	0.62	0.62	0.60
IBM	0.62	0.49	0.49	0.51	0.48	0.47	0.45	0.45	**0.43**
AIG	1.93	1.88	1.88	1.74	1.91	1.94	1.88	1.89	1.83
AVG	1.11	0.72	0.71	0.67	0.76	0.69	0.68	0.68	**0.65**

based on the application and the type of data.

Asset Return Prediction

We begin our experiments with asset return prediction data used in [92] [11]. It consists of weekly log returns of 9 stocks from the year 2004. It is considered in linear multivariate regression with output covariance estimation techniques [92]. We consider first-order vector auto-regressive models of the form $\boldsymbol{x}_t = f(\boldsymbol{x}_{t-1})$ where $\boldsymbol{x}_t$ corresponds to the 9-dimensional vector of weekly log-returns from 9 companies as shown in table 2.2. The dataset is split evenly such that the first 26 weeks of the year is used as the training set and the next 26 weeks is used as the test set. Following [101], we use univariate Gaussian kernels with 13 varying bandwidth, generated from each feature, as base kernels. The total number of base kernels sums to 117.

Performance is measured by the average mean-squared prediction error over the test set for each task. The experimental setup for this dataset follows exactly [92]. We compare the results from our proposed and baseline model with the results from Ordinary Least Square (*OLS*), Lasso, Multivariate Regression with Covariate Estimation (*MRCE*) and Factor Estimation and Selection (*FES*) models reported in [92] (See [92] for more details about the models). In addition to the standard baselines, we include Input Kernel Learning (*IKL*), which learns a vector of kernel weights $\boldsymbol{\beta}$ shared by all tasks [105].

After running *MK-MTRL* on these 117 base kernels, the model sets most of them to 0 except for base kernels corresponding to bandwidths $(1e-4, 1)$. These bandwidth selections represent the long-term and short-term dependencies common in temporal data. We reran the model with the selected non-zero bandwidths and report the results for these selected base kernels. We can see that the proposed model *MK-MTRL* performs better than all the baselines.

[11] http://cran.r-project.org/web/packages/MRCE/index.html

Table 2.3: Average AUC scores for different samples of *landmine* dataset. The table reports the mean and standard errors over 10 random runs.

	30 samples	50 Samples	80 Samples
STL	0.6315 ± 0.032	0.6540 ± 0.026	0.6542 ± 0.027
MTFL	0.6387 ± 0.037	0.6968 ± 0.015	0.7051 ± 0.020
MTRL	0.6555 ± 0.034	0.6933 ± 0.023	0.7074 ± 0.024
IMKL	0.6857 ± 0.024	0.7138 ± 0.011	0.7278 ± 0.011
MK-MTFL	$\mathbf{0.6866 \pm 0.018}$	0.7145 ± 0.009	0.7305 ± 0.009
MK-MTRL	$\mathbf{0.6870 \pm 0.033}$	$\mathbf{0.7242 \pm 0.011}$	$\mathbf{0.7405 \pm 0.014}$

Landmine Detection

This dataset [12] consists of 19 tasks collected from different landmine fields. Each task is a binary classification problem: landmines $(+)$ or clutter $(-)$ and each example consists of 9 features extracted from radar images with four moment-based features, three correlation-based features, one energy ratio feature and a spatial variance feature. Landmine data is collected from two different terrains: tasks $1-10$ are from highly foliated regions and tasks $11-19$ are from desert regions, therefore tasks naturally form two clusters. Any hypothesis learned from a task should be able to utilize the information available from other tasks belonging to the same cluster.

We choose $\{30, 50, 80\}$ examples per task for this dataset. We use a polynomial kernel with power $\{1, 2, 3, 4, 5\}$ for generating our base kernels. Note that we intentionally kept the size of the training data small to drive the need for learning from other tasks, which diminishes as the training sets per task become large. Due to class-imbalance issue (with few $(+)$ examples compared to $(-)$ examples), we use average Area Under the ROC Curve (AUC) as the performance measure. This dataset has been previously used for jointly learning feature correlation and task correlation [119]. Hence, *landmine* dataset is an ideal dataset for evaluating all the models.

Table 2.3 reports the results from the experiment. We can see that *MK-MTRL* performs better in almost all cases. When the number of training examples is small, *MK-MTRL* has difficulty in learning the task relationship matrix $\mathbf{\Omega}$, but *MK-MTFL* performs equally well as it shares the feature representation among the tasks which is especially useful when the number of training is relatively low. As we get more and more training data, *MK-MTRL* performs significantly better than all the other baselines.

Robot Inverse Dynamics

We consider the problem of learning the inverse dynamics of a 7-DOF *SARCOS* anthropomorphic data [13]. The dataset consists of 28 dimensions, of which first 21 dimensions are considered as features and the last 7 dimensions are used as outputs. We add an additional feature to account for

[12] http://www.ee.duke.edu/~lcarin/LandmineData.zip
[13] http://www.gaussianprocess.org/gpml/data/

the bias. There are 7 regression tasks and use kernel ridge regression to learn the task parameters and kernel weights. The feature set includes seven joint positions, seven joint velocities and seven joint accelerations, which is used to predict seven joint torques for the seven degrees of freedom (DOF). We randomly sample 2000 examples, of which $\{15, 50, 100, 150, 200, 600\}$ are used for training sets and the rest of the examples are used for test set. This dataset has been previous shown to include positive correlation, negative correlation and task unrelatedness and will be a challenging problem for baselines that doesn't learn the task correlation.

Following [120], we use normalized Mean Squared error (nMSE), which is the mean squared error divided by the variance of the ground truth. We generate 31 base kernels from multivariate Gaussian kernels with 10 varying bandwidth (based on the range of the data) and feature-wise linear kernel on each of the 21 dimensions. We use linear kernel for single task learning. The results calculated for different training set size is reported in Figure 2.2. We can see that *MK-MTRL* performs better than all the baselines. Contrary to the results report in [53], *MK-MTFL* performs the worst. As the model sees more data, it struggles to learn the task relationship and even performs worse than the single task learning.

Table 2.4: Comparison for multiple kernel models using nMSE on SARCOS data

	STL	IMKL	MK-MTRL
1st DOF	0.0862 ± 0.0033	0.0838 ± 0.0032	$\mathbf{0.0717 \pm 0.0075}$
2nd DOF	0.0996 ± 0.0041	0.0945 ± 0.0045	$\mathbf{0.0686 \pm 0.0070}$
3rd DOF	0.0918 ± 0.0042	0.0871 ± 0.0040	$\mathbf{0.0649 \pm 0.0071}$
4th DOF	0.0581 ± 0.0021	0.0514 ± 0.0020	$\mathbf{0.0298 \pm 0.0037}$
5th DOF	0.1513 ± 0.0063	0.1405 ± 0.0057	$\mathbf{0.1070 \pm 0.0053}$
6th DOF	0.2911 ± 0.0094	0.2822 ± 0.0081	$\mathbf{0.1835 \pm 0.0125}$
7th DOF	0.0715 ± 0.0025	0.0628 ± 0.0024	$\mathbf{0.0457 \pm 0.0036}$
AVG	0.1214 ± 0.0015	0.1146 ± 0.0013	$\mathbf{0.0816 \pm 0.0028}$

Moreover, we report the individual nMSE for each DOF in Table 2.4. It shows that *MK-MTRL* consistently outperforms in all the tasks. Comparing the results to the one reported in [120], we can see that *MT-MTRL* (with 0.0816 AVG nMSE score) performs better than *MTFL* and *MTRL* (with 0.3149 and 0.0912 AVG nMSE scores respectively).

Exam Score Prediction

For completeness, we include the results for benchmark dataset in multitask regression [14]. The *school* dataset consists of examination scores of 15362 students from 139 schools in London. Each school is considered as a task and the feature set includes the year of the examination, four

[14] http://ttic.uchicago.edu/~argyriou/code/mtl_feat/school_splits.tar

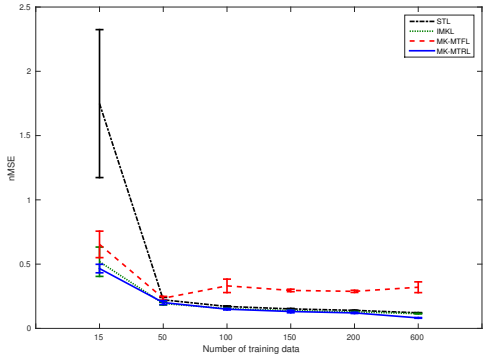

Figure 2.2: nMSE vs Number of training example for SARCOS data

school-specific and three student-specific attribute. We replace each categorical attribute with one binary variable for each possible attribute value, as in [9]. This results in 26 attributes with additional attribute to account for the bias term. We generate univariate Gaussian kernel with 13 varying bandwidths from each of the 26 attributes as our base kernels. Training and test set are obtained by dividing examples of each task into 60%-40%. We use explained variance as in [9], which is defined as one minus nMSE. We can see that *MK-MTRL* is better than both *IMKL* and *MK-MTFL*.

Table 2.5: Experiment on the usage of multiple kernels on *school* dataset *school*

	Explained Variance
STL	0.1883 ± 0.020
IMKL	0.1975 ± 0.017
MK-MTFL	0.2024 ± 0.016
MK-MTRL	$\mathbf{0.2134 \pm 0.016}$

Object Recognition

In this section, we evaluate our two proposed algorithms for *MK-MTRL* with computer vision datasets, *Caltech101*[15] and *Oxford* flowers [16] in terms of accuracy and training time. *Caltech101* dataset consists of 9144 images from 102 categories of objects such as faces, watches, animals, etc. The minimum, average and maximum number of images per category are 31,90 and 800 respectively. The *Caltech101* base kernels for each task are generated from feature descriptors such as geometric blur, PHOW gray/color, self-similarity, etc. For each of the 102 classes, we select 30 examples (for a total of 3060 examples per task) and then split these 30 examples into

[15] http://www.vision.ee.ethz.ch/~pgehler/projects/iccv09
[16] http://www.robots.ox.ac.uk/~vgg/data/flowers/17/datasplits.mat

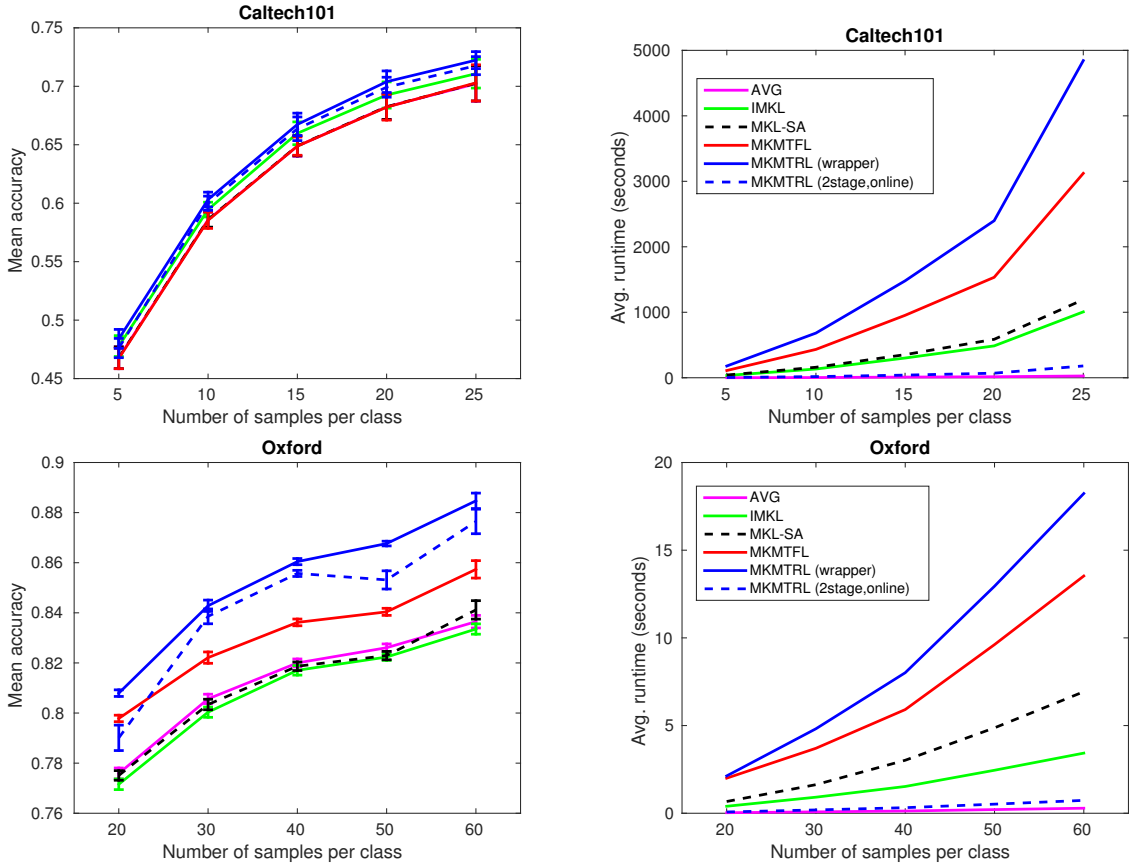

Figure 2.3: **Top**: Mean accuracy (left) and runtime (right) calculated for *Caltech101* dataset with varying training set sizes. **Bottom**: Mean accuracy (left) and runtime (right) calculated for *Oxford* dataset with varying training set sizes.

testing and training folds, which ensures matching training and testing distributions. *Oxford* flowers consists of 17 varies of flowers and the number of images per category is 80. The *Oxford* base kernels for each task are generated from a subset of feature values. Each one-vs-all binary classification problem is considered as a single task, which amount to 102 and 17 tasks with 38 and 7-base kernels per task, respectively. Following the previous work, we set the value of $C = 1000$ for *Caltech101* dataset.

In addition to the baselines used before, we compare our algorithms with Multiple Kernel Learning by Stochastic Approximation (*MKL-SA*) [19]. *MKL-SA* has a similar formulation to that of (*MK-MTFL*), except that it sets $\lambda_{tk} = \lambda_t, \forall k$ in equation 2.16. At each time step, it samples one task, according to the multinomial distribution $Multi(\lambda_1, \lambda_2, \ldots, \lambda_T)$, to update it's model parameter, making it suitable for multitask learning with large number of tasks.

The results for *Caltech101* and *Oxford* shown in Figure fig:obj. The left plots show how the mean accuracy varies with respect to different training set sizes. The right plots show the average training time taken by each model with varying training set sizes. From the plots, we can see that *Caltech101* outperforms all the other state-of-the art baselines. But the run-time of *MK-MTFL* and *MK-MTRL* grows steeply in the number of samples per class. Similar results are observed when

we increase the number of tasks or number of base kernels per task. This explains the need for efficient learning algorithm for multitask multiple kernel learning problems. We report *MK-MTRL* with two-stage, online procedure as one of the baselines. On both *Caltech101* and *Oxford*, the two-stage procedure yields comparable performance to that of *MK-MTRL*.

The run-time complexity of two-stage, online *MK-MTRL* learning is significantly better than almost all the baselines. Since *AVG* takes the average of the task-specific base kernels, it has the lowest computational time. It is interesting to see that two-stage, online *MK-MTRL* performs better than *MKL-SA* both in terms of accuracy and running time. We believe that since *MKL-SA* updates the kernel weights after learning a single model parameter, it takes more iterations to converge (in term of model parameters and the kernel weights).

2.3 Conclusions

In this chapter, we proposed a novel online multitask learning algorithm (*OSMTL*) that jointly learns the per-task hypothesis and the inter-task relationships. The key idea is based on smoothing the loss function of each task w.r.t. a probabilistic distribution over all tasks, and adaptively refining such distribution over time. In addition to closed-form updating rules, we show our method achieves the sub-linear regret bound. Effectiveness of our algorithm is empirically verified over several benchmark datasets.

Following this work, we proposed a novel multiple kernel multitask learning algorithm that uses inter-task relationships to efficiently learn the kernel weights. The key idea is based on the assumption that the related tasks will have similar weights for the task-specific base kernels. We proposed an iterative algorithm to jointly learn this task relationship matrix, kernel weights and the task model parameters. For large-scale datasets, we introduced a novel two-stage online learning algorithm to learn kernel weights efficiently. The effectiveness of our algorithm is empirically verified over several benchmark datasets. The results showed that both multiple kernel learning and task relationship learning for multitask problems significantly helps in boosting the performance of the model.

Chapter 3

Active Learning from Peers

Multitask learning leverages the relationship between the tasks to transfer relevant knowledge from information-rich tasks to information-poor ones. Most existing work in multitask learning focuses on how to take advantage of these task relationships, either by sharing data directly [31] or learning model parameters via cross-task regularization techniques [9, 114, 120]. This chapter focuses on a specific multitask setting where tasks are allowed to interact by requesting labels from other tasks for difficult cases.

Following the previous chapter, we consider an online multitask setting where, at each round t, the learner starts with *receiving* an example (along with a task identifier) and *predicts* the output label. Note that one may also consider learning multiple tasks simultaneously by receiving K examples for K tasks at each round t. Subsequently, the learner receives the true label and *updates* the model(s) as necessary. This sequence is repeated over the entire data, simulating a data stream. In this setting, the assumption is that the true label is readily available for the task learner, which is impractical in many applications.

Recent works in active learning for sequential problems have addressed this concern by allowing the learner to make a decision on whether to ask the oracle to provide the true label for the current example and incur a cost or to skip this example. Most approach in active learning for sequential problems use a measure such a confidence of the learner in the current example [2, 24, 26, 37, 82]. In online multitask learning, one can utilize the task relationship to further reduce the total number of labels requested from the oracle. This chapter presents a novel active learning for the sequential decision problems using *peers* or *related tasks*. The key idea is that when the learner is not confident on the current example, the learner is allowed to query its peers, which usually has a low cost, before requesting a true label from the oracle and incur a high cost. Our approach follows a perceptron-based update rule in which the model for a given task is updated only when the prediction for that task is in error. The goal of an online learner in this setting is to minimize errors attempting to reach the performance of the full hindsight learner and at the same time, reduce the total number of queries issued to the oracle.

There are many useful application areas for online multitask learning with selective sampling, including optimizing financial trading, email prioritization and filtering, personalized news, crowd source-based annotation, spam filtering and spoken dialog system, etc. Consider the latter, where several automated agents/bots servicing several clients. Each agent is specialized or trained to answer questions from customers on a specific subject such as automated payment,

troubleshooting, adding or cancelling services, etc. In such setting, when one of the automated agents cannot answer a customer's question, it may request the assistance of another automated agent that is an expert in the subject related to that question. For example, an automated agent for customer retention may request some help from an automated agent for new services to offer new deals for the customer. When both the agents could not answer the customer's question, the system may then direct the call to a live agent. This may reduce the number of service calls directed to live agents and the cost associated with such requests.

Similarly in spam filtering, where some spam is universal to all users (e.g. financial scams), some messages might be useful to certain affinity groups, but spam to most others (e.g. announcements of meditation classes or other special interest activities), and some may depend on evolving user interests. In spam filtering each user is a task, and shared interests and dis-interests formulate the inter-task relationship matrix. If we can learn the task relationship matrix as well as improving models from specific decisions from peers on difficult examples, we can perform mass customization of spam filtering, borrowing from spam/not-spam feedback from users with similar preferences. The primary contribution studied in this chapter is precisely active learning for multiple related tasks and its use in estimating per-task model parameters in an online setting.

3.1 Problem Setup

Suppose we are given K tasks where the k^{th} task is associated with N_k training examples. For brevity, we consider a binary classification problem for each task, but the methods generalize to multi-class settings and are also applicable to regression tasks. We denote by $[N]$ consecutive integers ranging from 1 to N. Let $\left\{(x_k^{(i)}, y_k^{(i)})\right\}_{i=1}^{N_k}$ be data for task k where $x_k^{(i)} \in \mathbb{R}^d$ is the i^{th} instance from the k^{th} task and $y_k^{(i)}$ is its corresponding true label. When the notation is clear from the context, we drop the index k and write $((x^{(i)}, k), y^{(i)})$.

Let $\{w_k^*\}_{k\in[K]}$ be any set of arbitrary vectors where $w_k^* \in \mathbb{R}^d$. The hinge losses on the example $\left((x^{(t)}, k), y^{(t)}\right)$ are given by $\ell_{kk}^{(t)*} = \left(1 - y^{(t)}\langle x^{(t)}, w_k^*\rangle\right)_+$ and $\ell_{km}^{(t)*} = \left(1 - y^{(t)}\langle x^{(t)}, w_m^*\rangle\right)_+$, respectively, where $(z)_+ = \max(0, z)$. Similarly, we define hinge losses $\ell_{kk}^{(t)}$ and $\ell_{km}^{(t)}$ for the linear predictors $\{w_k^{(t)}\}_{k\in[K]}$ learned at round t. Let $Z^{(t)}$ be a Bernoulli random variable to indicate whether the learner requested a true label for the example $x^{(t)}$. Let $M^{(t)}$ be a binary variable to indicate whether the learner made a mistake on the example $x^{(t)}$. We use the following expected hinge losses for our theoretical analysis: $\tilde{L}_{kk} = \mathbb{E}\left[\sum_t M^{(t)} Z^{(t)} \ell_{kk}^{(t)*}\right]$ and $\tilde{L}_{km} = \mathbb{E}\left[\sum_t M^{(t)} Z^{(t)} \ell_{km}^{(t)*}\right]$.

We start with our proposed active learning from peers algorithm based on selective sampling for online multitask problems and study the mistake bound for the algorithm in Section 3.2. We report our experimental results and analysis in Section 3.4. Additionally, we extend our problem setting to learning multiple task in parallel at the end of the chapter.

3.1.1 Related Work

While there is considerable literature in online multitask learning, many crucial aspects remain largely unexplored. Most existing work in online multitask learning focuses on how to take advantage of task relationships. To achieve this, Lugosi et. al [73] imposed a hard constraint on the K simultaneous actions taken by the learner in the expert setting, Agarwal et. al [3] used matrix regularization, and Dekel et. al [35] proposed a global loss function, as an absolute norm, to tie together the loss values of the individual tasks. In all these works, their proposed algorithms assume that the true labels are available for each instance.

Selective sampling-based learners in online setting, on the other hand, decides whether to ask the human oracle for labeling of difficult instances [2, 24, 26, 37, 82]. It can be easily extended to online multitask learning setting by applying selective sampling for each individual task separately. Saha et. al [96] formulated the learning of task relationship matrix as a Bregman-divergence minimization problem w.r.t. positive definite matrices and used this task-relationship matrix to naively select the instances for labelling from the human oracle.

Several recent works in online multitask learning recommended updating all the task learners on each round t [25, 79, 96]. When a task learner makes a mistake on an example, all the tasks' model parameters are updated to account for the new examples. This significantly increases the computational complexity at each round, especially when the number of tasks is large [112]. Our proposed method avoids this issue by updating only the learner of the current example and utilize the knowledge from peers only when the current learner requested them.

This work is motivated by the recent interests in active learning from multiple (strong or weak) teachers [37, 41, 42, 109, 115, 116]. Instead of single all-known oracle, these earlier works assume multiple oracles (or teachers) each with a different area of expertise. At round t, some of the teachers are experts in the current instance but the others may not be confident in their predicted labels. Such learning setting is very common in crowd-sourcing platform where multiple annotators are used to label an instance. Our learning setting is different from their approaches where, instead of learning from multiple oracles, we learn from our peers (or related tasks) without any associated high cost. Finally, our proposed method is closely related to learning with rejection option [13, 30] where the learner may choose not to predict label for an instance. To reject an instance, they use a measure of confidence to identify difficult instances. We use a similar approach to identify when to query peers and when to query the human oracle for true label.

3.2 Learning from Peers

We consider multitask perceptron for our online learning algorithm. On each round t, we receive an example $(x^{(t)}, k)$ from task k. We will consider a different online learning setting later in this chapter where we simultaneously receive K examples at each round, one for each task k. Each perceptron learner for the task k maintains a model represented by $w_k^{(t-1)}$ learned from examples received until round $t-1$. Task k predicts a label for the received example $x^{(t)}$ using $h_k(x^{(t)}) = \langle w_k^{(t-1)}, x^{(t)} \rangle$ [1]. As in the previous works [13, 26, 37], we use $|h_k(x^{(t)})|$ to measure

[1] We also use the notation $\hat{p}_{kk} = \langle w_k^{(t-1)}, x^{(t)} \rangle$ and $\hat{p}_{km} = \langle w_m^{(t-1)}, x^{(t)} \rangle$

1. Receive an example $x^{(t)}$ for the task k
2. If the task k is not confident in the prediction for this example, ask the *peers* or *related tasks* whether they can give a confident label to this example.
3. If the *peers* are not confident enough, ask the oracle for the true label $y^{(t)}$.

Figure 3.1: Proposed learning approach from peers.

the confidence of the k^{th} task learner on this example. When the confidence is higher, the learner doesn't require the need to request the true label $y^{(t)}$ from the oracle.

Built on this idea, [26] proposed a selective sampling algorithm using the margin $|h_k(x^{(t)})|$ to decide whether to query the oracle or not. Intuitively, if $|h_k(x^{(t)})|$ is small, then the k^{th} task learner is not confident in the prediction of $x^{(t)}$ and vice versa. They consider a Bernoulli random variable $P^{(t)}$ for the event $|h_k(x^{(t)})| \leq b_1$ with probability $\frac{b_1}{b_1+|h_k(x^{(t)})|}$ for some predefined constant $b_1 \geq 0$. If $P^{(t)} = 1$ (confidence is low), then the k^{th} learner requests the oracle for the true label. Similarly when $P^{(t)} = 0$ (confidence is high), the learner skips the request to the oracle. This considerably saves a lot of label requests from the oracle. When dealing with multiple tasks, one may use similar idea and apply selective sampling for each task individually [27]. Unfortunately, such approach doesn't take into account the inherent relationship between the tasks.

In this chapter, we consider a novel active learning (or selective sampling) for online multitask learning to address the concerns discussed above. Our proposed learning approach can be summarized in Figure 3.1. Unlike in the previous work [25, 79, 96], we update only the current task parameter w_k when we made a mistake at round t, instead of updating all the task model parameters $w_m, \forall m \in [K], m \neq k$. Our proposed method avoids this issue by updating only the learner of the current example and share the knowledge from peers only when the assistance is needed. In addition, the task relationship is taken into account, to measure whether the peers are confident in predicting this example. This approach provides a compromise between learning them independently and learning them by updating all the learners when a specific learner makes a mistake.

As in traditional selective sampling algorithm [26], we consider a Bernoulli random variable $P^{(t)}$ for the event $|h_k(x^{(t)})| \leq b_1$ with probability $\frac{b_1}{b_1+|h_k(x^{(t)})|}$. In addition, we consider a second Bernoulli random variable $Q^{(t)}$ for the event $|h_m(x^{(t)})| \leq b_2$ with probability $\frac{b_2}{b_2+\sum_{m\in[K],m\neq k} \tau_{km}^{(t-1)}|h_m(x^{(t)})|}$. The idea is that when the weighted sum of the confidence of the peers on the current example is high, then we use the predicted label $\tilde{y}^{(t)}$ from the peers for the perceptron update instead of requesting a true label $y^{(t)}$ from the oracle. In our experiment in Section 3.4, we consider the confidence of most related task instead of the weighted sum to reduce the computational complexity at each round. We set $Z^{(t)} = P^{(t)}Q^{(t)}$ and set $M^{(t)} = 1$ if we made a mistake at round t i.e., $(y^{(t)} \neq \hat{y}^{(t)})$ (only when the label is revealed/queried).

The pseudo-code is in Algorithm 5. Line 14 is executed when we request a label from the oracle or when peers are confident on the label for the current example. Note the two terms in $(M^{(t)}Z^{(t)}y^{(t)} + \tilde{Z}^{(t)}\tilde{y}^{(t)})$ are mutually exclusive (when $P^{(t)} = 1$). Line (15-16) computes the

Algorithm 5: Active Learning from Peers

Input : $b_1 > 0$, $b_2 > 0$ s.t., $b_2 \geq b_1$, $\lambda > 0$, Number of rounds T

Initialize $w_m^{(0)} = \mathbf{0}\ \forall m \in [K]$, $\boldsymbol{\tau}^{(0)}$.

for $t = 1 \dots T$ **do**

  Receive $(x^{(t)}, k)$

  Compute $\hat{p}_{kk}^{(t)} = \langle x^{(t)}, w_k^{(t-1)} \rangle$

  Predict $\hat{y}^{(t)} = sign(\hat{p}_{kk}^{(t)})$

  Draw a Bernoulli random variable $P^{(t)}$ with probability $\frac{b_1}{b_1 + |\hat{p}_{kk}^{(t)}|}$

  if $P^{(t)} = 1$ **then**

    Compute $\hat{p}_{km}^{(t)} = \langle x^{(t)}, w_m^{(t-1)} \rangle\ \forall m \neq k, m \in [K]$

    Compute $\tilde{p}^{(t)} = \sum_{m \neq k, m \in [K]} \tau_{km}^{(t-1)} \hat{p}_{km}^{(t)}$ and $\tilde{y}^{(t)} = sign(\tilde{p}^{(t)})$

    Draw a Bernoulli random variable $Q^{(t)}$ with probability $\frac{b_2}{b_2 + |\tilde{p}^{(t)}|}$

  end

  Set $Z^{(t)} = P^{(t)} Q^{(t)}$ & $\tilde{Z}^{(t)} = P^{(t)}(1 - Q^{(t)})$

  Query true label $y^{(t)}$ if $Z^{(t)} = 1$ and set $M^{(t)} = 1$ if $\hat{y}^{(t)} \neq y^{(t)}$

  Update $w_k^{(t)} = w_k^{(t-1)} + (M^{(t)} Z^{(t)} y^{(t)} + \tilde{Z}^{(t)} \tilde{y}^{(t)}) x^{(t)}$

  Update τ:

$$\tau_{km}^{(t)} = \frac{\tau_{km}^{(t-1)} e^{-\frac{Z^{(t)}}{\lambda} \ell_{km}^{(t)}}}{\sum_{\substack{m' \in [K] \\ m' \neq k}} \tau_{km'}^{(t-1)} e^{-\frac{Z^{(t)}}{\lambda} \ell_{km'}^{(t)}}} \qquad m \in [K], m \neq k \tag{3.1}$$

end

relationship between tasks τ_{km} based on the recent work by [79]. It maintains a distribution over peers w.r.t the current task. The value of τ is updated at each round using the cross-task error ℓ_{km}. In addition, we use the τ to get the confidence of the most-related task rather than the weighted sum of the confidence of the peers to get the predicted label from the peers (see Section 3.4 for more details). When we are learning with many tasks [112], it provides a faster computation without significantly compromising the performance of the learning algorithm. One may use different notion of task relationship based on the application at hand. Now, we give the bound on the expected number of mistakes.

Theorem 3. *let* $S_k = \left\{ \left((x^{(t)}, k), y^{(t)} \right) \right\}_{t=1}^{T}$ *be a sequence of* T *examples given to Algorithm 5 where* $x^{(t)} \in \mathbb{R}^d$, $y^{(t)} \in \{-1, +1\}$ *and* $X = \max_t \|x^{(t)}\|$. *Let* $P^{(t)}$ *be a Bernoulli random variable for the event* $|h_k(x^{(t)})| \leq b_1$ *with probability* $\frac{b_1}{b_1 + |h_k(x^{(t)})|}$ *and let* $Q^{(t)}$ *be a Bernoulli random variable for the event* $|h_m(x^{(t)})| \leq b_2$ *with probability* $\frac{b_2}{b_2 + \max_{\substack{m \in [K] \\ m \neq k}} |h_m(x^{(t)})|}$. *Let* $Z^{(t)} = P^{(t)} Q^{(t)}$ *and* $M^{(t)} = \mathbb{I}(y^{(t)} \neq \hat{y}^{(t)})$.

If the Algorithm 5 is run with $b_1 > 0$ *and* $b_2 > 0$ $(b_2 \geq b_1)$, *then* $\forall t \geq 1$ *and* $\gamma > 0$ *we have*

$$\mathbb{E}\Big[\sum_{t\in[T]} M^{(t)}\Big] \leq \frac{b_2}{\gamma}\Bigg[\frac{(2b_1+X^2)^2}{8b_1\gamma}\big(\|w_k^*\|^2 + \max_{m\in[K],m\neq k}\|w_m^*\|^2\big) + \big(1+\frac{X^2}{2b_1}\big)\Big(\tilde{L}_{kk} + \max_{m\in[K],m\neq k}\tilde{L}_{km}\Big)\Bigg]$$

Then, the expected number of label requests to the oracle by the algorithm is

$$\sum_t \frac{b_1}{b_1+|h_k(x^{(t)})|}\frac{b_2}{b_2+\max_{\substack{m\in[K]\\ m\neq k}}|h_m(x^{(t)})|}$$

The proof is given in Appendix A. It follows from Theorem 1 in [26] and Theorem 1 in [79] and setting $b_2 = b_1 + \frac{X^2}{2} + \frac{\|w_k^*\|}{2}$, where $b_1 = \frac{X^2}{2}\sqrt{1+\frac{4\gamma^2}{\|w_k^*\|X^2}\frac{\tilde{L}_{kk}}{\gamma}}$. Theorem 3 states that the quality of the bound depends on both $\tilde{L}_{kk}$ and the maximum of $\{\tilde{L}_{km}\}_{m\in[K],m\neq k}$. In other words, the worst-case regret will be lower if the k^{th} true hypothesis w_k^* can predict the labels for training examples in both the k^{th} task itself as well as those in all the other related tasks in high confidence. In addition, we consider a related problem setting in which all the K tasks receive an example simultaneously. We give the learning algorithm and mistake bound for this setting in Appendix B.

3.3 Learning with a Shared Annotator

In this section, we explore a related approach where multiple tasks are learned with a shared annotator. In this setting, we assume that all tasks will be performed in each round. At time t, the k^{th} task receives a training instance $x_k^{(t)}$, makes a prediction $\langle x_k^{(t)}, w_k^{(t)}\rangle$ and suffers a loss after $y^{(t)}$ is revealed. Unlike the problem setting in Algorithm 5, we are allowed to query the Oracle for at most κ examples out of the K examples received at round t where $\kappa \leq K$. Our algorithm follows a perceptron-based update rule, as in Algorithm 5, in which the model is updated only when a task makes a mistake. The key idea is that the algorithm picks κ examples from K tasks that desperately need human assistance.

Most recently, Cohen et. al. [27] proposed a selective sampling-based approach called SHAMPO for this problem setting. Each task k is chosen for label request from oracle with probability $\Pr(\mathcal{J}_i = k) \propto b_1/(b_1 + |\hat{p}_{kk}^{(t)}| - \min_{m=1}^{K}|\hat{p}_{mm}^{(t)}|)$, $\forall i \in [\kappa]$ and we choose at most κ tasks for label requests to oracle. We define $\mathcal{J} = \{\mathcal{J}_i : i \in [\kappa]\}$. Unlike in Algorithm 5, we perform a perceptron update when we make a mistake $M_k^{(t)} = \mathbb{I}(y_k^{(t)} \neq \hat{y}_k^{(t)})$ or when the example has less confidence $A_k^{(t)} = \mathbb{I}(0 < y_k^{(t)}\hat{p}_{kk}^{(t)} \leq \frac{\gamma}{2})$. In their proposed method, the examples from the other tasks $k \notin \mathcal{J}$ are not updated on this round. In addition, their methods doesn't take into account the relationship between the tasks. Our learning procedure from Algorithm 5 provides a natural way to extend their method to learn from peers and to utilize the relationship between the tasks efficiently.

The pseudo-code is in Algorithm 6. Lines $(9-12)$ is similar to their proposed learning framework. We add the lines $(15-18)$ to query the peers for labels for the tasks that are not

Algorithm 6: Learning Multiple Tasks in Parallel from Peers

Input : $b_1 > 0$, $b_2 > 0$ s.t., $b_2 \geq b_1$, $\lambda > 0$, $\gamma > 0$ Number of rounds T

Initialize $w_m^{(0)} = \mathbf{0}\ \forall m \in [K]$, $\boldsymbol{\tau}^{(0)}$.
for $t = 1 \dots T$ **do**
  Receive K examples: $\{x_k^{(t)} : k \in [K]\}$
  Compute $\hat{p}_{kk}^{(t)} = \langle x_k^{(t)}, w_k^{(t-1)} \rangle$, $k \in [K]$
  Predict K labels: $\hat{y}_k^{(t)} = sign(\hat{p}_{kk}^{(t)}), k \in [K]$
  Draw κ tasks for query with probability
   $\Pr(\mathcal{J}_i = k) \propto b_1/(b_1 + |\hat{p}_{kk}^{(t)}| - \min_{m=1}^{K} |\hat{p}_{mm}^{(t)}|), \forall i \in [\kappa]$
  for $k = 1 \dots [K]$ **do**
    if $k \in \mathcal{J}$ **then**
      Query true label $y_k^{(t)}$
      Set $M_k^{(t)} = 1$ if $\hat{y}_k^{(t)} \neq y_k^{(t)}$
      Set $A_k^{(t)} = 1$ if $0 < y_k^{(t)} \hat{p}_{kk}^{(t)} \leq \frac{\gamma}{2}$
      Update $w_k^{(t)} = w_k^{(t-1)} + (M_k^{(t)} + A_k^{(t)}) y_k^{(t)} x_k^{(t)}$.
      Update $\tau^{(t)}$ as in Equation 3.1.
    else
      Compute $\hat{p}_{km}^{(t)} = \langle x_k^{(t)}, w_m^{(t-1)} \rangle\ \forall m \neq k, m \in [K]$
      Compute $\tilde{p}_k^{(t)} = \sum_{m \neq k, m \in [K]} \tau_{km}^{(t-1)} \hat{p}_{km}^{(t)}$ and
       $\tilde{y}_k^{(t)} = sign(\tilde{p}_k^{(t)})$
      Draw a Bernoulli random variable $\tilde{Z}_k^{(t)}$ with probability
       $\frac{|\tilde{p}_k^{(t)}|}{b_2 + |\tilde{p}_k^{(t)}|}$
      Update $w_k^{(t)} = w_k^{(t-1)} + \tilde{Z}_k^{(t)} \tilde{y}_j^{(t)} x_k^{(t)}$
    end
  end
end

selected at this round $k \notin \mathcal{J}$ and the line 13 to incorporate the relationship between the tasks when the true labels are available. We give the (expected) mistake bound for the Algorithm 6 in Theorem 4.

Theorem 4. *$\forall k \in [K]$, let $S_k = \left\{ \left(x_k^{(t)}, y_k^{(t)} \right) \right\}_{t=1}^{T}$ be a sequence of T examples for the k^{th} task where $x_k^{(t)} \in \mathbb{R}^d$, $y_k^{(t)} \in \{-1, +1\}$ and $\|x_k^{(t)}\|_2 \leq R$, $\forall t \in [T]$. Let $M_k^{(t)} = \mathbb{I}(y_k^{(t)} \neq \hat{y}_k^{(t)})$ and $A_k^{(t)} = \mathbb{I}(0 < y_k^{(t)} \hat{p}_{kk}^{(t)} \leq \frac{\gamma}{2})$.*

If $\{S_k\}_{k \in [K]}$ is presented to Algorithm 6 with $b_1 > 0$ $(b_1 \geq \gamma)$ and $b_2 > 0$ $(b_2 \geq b_1)$, then $\forall t \geq 1$ and $\gamma > 0$ we have

$$\mathbb{E}\Big[\sum_{k\in[K]}\sum_{t\in[T]} M_k^{(t)}\Big] \leq \frac{b_2 K}{\gamma}\Big[\frac{(2b_1+X^2)^2}{8b_1\gamma}\big(\|w_k^*\|^2 + \max_{m\in[K],m\neq k}\|w_m^*\|^2\big) + \big(1+\frac{X^2}{2b_1}\big)\big(\tilde{L}_{kk} + \max_{m\in[K],m\neq k}\tilde{L}_{km}\big)\Big] + \Big(\frac{\gamma}{b_1}-1\Big)\mathbb{E}\Big[\sum_{k\in[K]}\sum_{t\in[T]} A_k^{(t)}\Big]$$

The proof is straightforward and follows directly from the proof of Theorem 3 and Theorem 1 in [27]. The first two terms in the bound are same as in Theorem 3. The last term in the bound accounts for the aggressiveness of the algorithm. The intuition is that when we set b_1 to the margin (i.e., $b_1 = \gamma$), the last term will become 0 and the bound reduces to the one given in Theorem 3. When $b_1 > \gamma$, the aggressive term in the bound reduces the expected number of the mistakes made by Algorithm 6 and increases the expected number of label requests to the peers and eventually to the oracle.

3.4 Experiments

We evaluate the performance of our algorithm in the online setting. All reported results in this section are averaged over 10 random runs on permutations of the training data. We set the value of $b_1 = 1$ for all the experiments and the value of b_2 is tuned from 20 different values. Unless otherwise specified, all model parameters are chosen via 5-fold cross-validation.

To evaluate the performance of our proposed approach, we compare our proposed methods to 2 standard baselines. The first baseline selects the examples to query randomly (Random) and the second baseline chooses the examples via selective sampling independently for each task (Independent) [26]. We compare these baselines against two versions of our proposed algorithm 5 with different confidence measures for predictions from peer tasks: PEERsum where the confidence $\tilde{p}^{(t)}$ at round t is computed by the weighted sum of the confidence of each task as shown originally in Algorithm 5 and PEERone where the confidence $\tilde{p}^{(t)}$ is set to the confidence of the most related task k ($\hat{p}_k^{(t)}$), sampled from the probability distribution $\tau_{km}^{(t)},\ m \in [K],\ m \neq k$. The intuition is that, for multitask learning with many tasks [112], PEERone provides a faster computation without significantly compromising the performance of the learning algorithm. The task weights τ are computed based on the relationship between the tasks. As mentioned earlier, the τ values can be easily replaced by other functions based on the application at hand [2].

In addition to PEERsum and PEERone, we evaluated a method that queries the peer with the highest confidence, instead of the most related task as in PEERone, to provide the label. Since this method uses only local information for the task with the highest confidence, it is not necessarily the best peer in hindsight, and the results are worse than or comparable (in some cases) to the Independent baseline. Hence, we do not report its results in our experiment.

Figure 3.2 shows the performance of the models during training. We measure the average rate of mistakes (cumulative measure), the number of label requests to the oracle and the number of peer query requests to evaluate the performance during the training time. From Figure 3.2

[2]Our algorithm and theorem can be easily generalized to other types of functions on τ

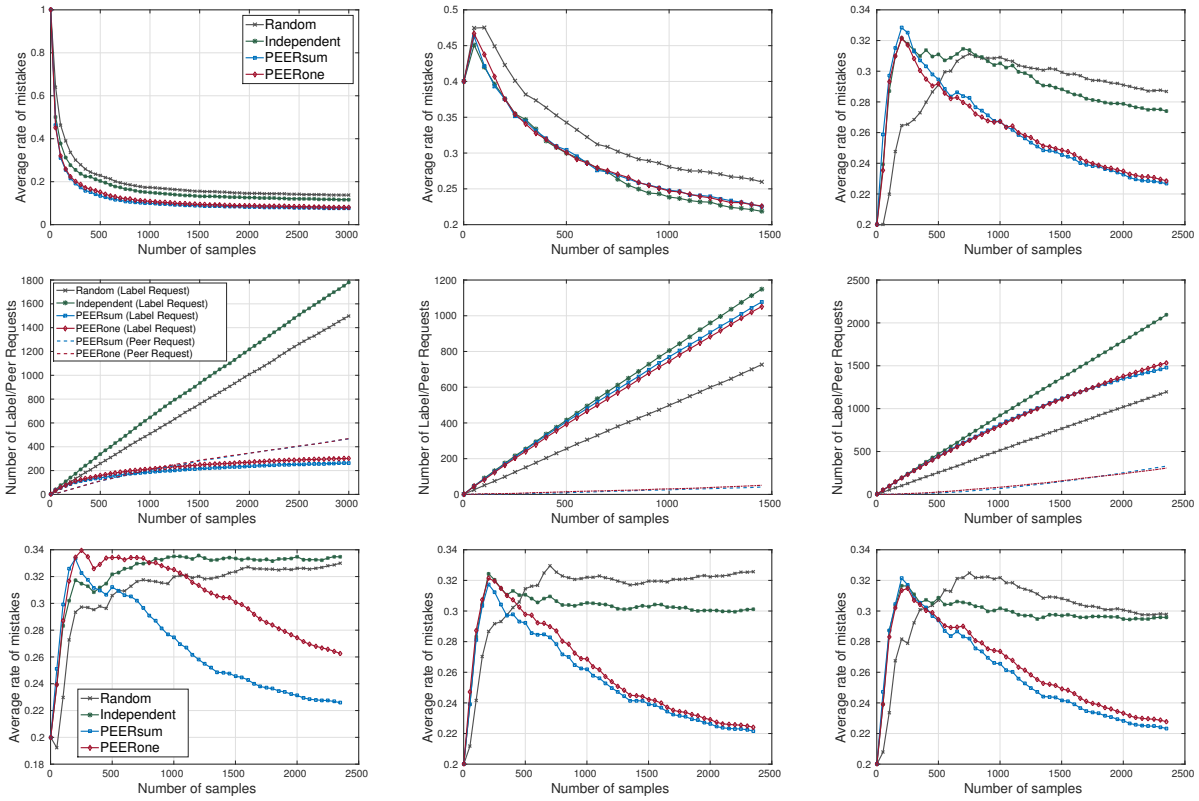

Figure 3.2: Average rate of mistakes vs. Number of examples calculated for compared models on the three datasets (top). Average number of label and peer requests on the three datasets (middle). Average rate of (training) mistakes vs. Number of examples with the query budget of $(10\%, 20\%, 30\%)$ of the total number of examples T on *sentiment* (bottom). These plots are generated during the training.

(top and middle), we can see that our proposed methods (PEERsum and PEERone) outperform both the baselines. Among the proposed methods, PEERsum outperforms PEERone as it uses the confidence from all the tasks (weighted by task relationship) to measure the final confidence. We notice that during the earlier part of the learning, all the methods issue more query to the oracle. After a few initial set of label requests, peer requests (dotted lines) steadily take over in our proposed methods. We can see three noticeable phases in our learning algorithm: initial label requests to the oracle, label requests to peers, and as task confidence grows, learning with less dependency on other tasks.

In order to efficiently evaluate the proposed methods, we restrict the total number of label requests issued to the oracle during training, that is we give all the methods the same query budget: $(10\%, 20\%, 30\%)$ of the total number of examples T on *sentiment* dataset. After the desired number of label requests to the oracle reached the said budget limit, the baseline methods predict label for the new examples based on the earlier assistance from the oracle. On the other hand, our proposed methods continue to reduce the average mistake rate by requesting labels from peers. This shows the power of learning from peers when human expert assistance is expensive,

Table 3.1: Average test accuracy on three datasets: means and standard errors over 10 random shuffles.

Models	**Landmine Detection**			**Spam Detection**			**Sentiment Analysis**		
	ACC	*#Queries*	*Time (s)*	*ACC*	*#Queries*	*Time (s)*	*ACC*	*#Queries*	*Time (s)*
Random	0.8905 (0.007)	1519.4 (31.9)	0.38	0.8117 (0.021)	753.4 (29.1)	8	0.7443 (0.028)	1221.8 (22.78)	35.6
Independent	0.9040 (0.016)	1802.8 (35.5)	0.29	0.8309 (0.022)	1186.6 (18.3)	7.9	0.7522 (0.015)	2137.6 (19.1)	35.6
PEERsum	**0.9403** (0.001)	265.6 (18.7)	0.38	**0.8497** (0.007)	1108.8 (32.1)	8	**0.8141** (0.001)	1494.4 (68.59)	36
PEERone	0.9377 (0.003)	303 (17)	1.01	0.8344 (0.018)	1084.2 (24.2)	8.3	0.8120 (0.01)	1554.6 (92.2)	36.3

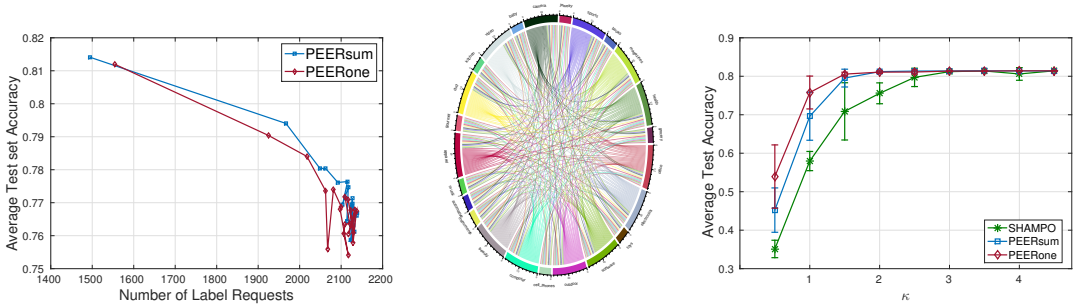

Figure 3.3: Average test set ACC calculated for different values of b_2 (left). A visualization of the peer query requests among the tasks in *sentiment* learned by PEERone (middle) and comparison of proposed methods against SHAMPO in parallel setting. We report the average test set accuracy (right).

scarce or unavailable.

Table 3.1 summarizes the performance of all the above algorithms on the test set for the three datasets. In addition to the average accuracy ACC scores, we report the average total number of queries or label requests to the oracle ($\#Queries$) and the CPU time taken (seconds) for learning from T examples (*Time*). From the table, it is evident that PEER* outperforms all the baselines in terms of both ACC and $\#Queries$. In case of *landmine* and *sentiment*, we get a significant improvement in the test set accuracy while reducing the total number of label requests to the oracle. As in the training set results before, PEERsum performs slightly better than PEERone. Our methods perform slightly better than Independent in *spam*, we can see from Figure 3.2 (middle) for *spam* dataset, the number of peer queries are lower compared to that of the other datasets.

The results justify our claim that relying on assistance from peers in addition to human intervention leads to improved performance. Moreover, our algorithm consumes less or comparable CPU time than the baselines which take into account inter-task relationships and peer requests. Note that PEERone takes a little more training time than PEERsum. This is due to our implementation that takes more time in (MATLAB's) inbuilt sampler to draw the most related task. One may improve the sampling procedure to get better run time. However, the time spent on selecting the most related tasks is small compared to the other operations when dealing with many tasks.

Figure 3.3 (left) shows the average test set accuracy computed for 20 different values of b_2 for

PEER* methods in *sentiment*. We set $b_1 = 1$. Each point in the plot corresponds to ACC (y-axis) and $\#Queries$ (x-axis) computed for a specific value of b_2. We find the algorithm performs well for $b_2 > b_1$ and the small values of b_2. When we increase the value of b_2 to ∞, our algorithm reduces to the baseline (Independent), as all request are directed to the oracle instead of the peers.

Figure 3.3 (middle) shows the snapshot of the total number of peer requests between the tasks in *sentiment* at the end of the training of PEERone. Each edge says that there was one peer query request from a task/domain to another related task/domain (based on the task relationship matrix τ). The edges with similar colors show the total number of peer requests from a task. It is evident from the figure that all the tasks are collaborative in terms of learning from each other.

Figure 3.3 (right) compares the PEER* implementation of Algorithm 2 against SHAMPO in terms of test set accuracy for *sentiment* dataset. The algorithm learns multiple tasks in parallel, where at most κ out of K label requests to the oracle are allowed at each round. While SHAMPO ignores the other tasks, our PEER* allows peer query to related tasks and thereby improves the overall performance. As we can see from the figure, when κ is set to small values, PEER* performs significantly better than SHAMPO.

3.5 Conclusions

In this chapter, we proposed a novel online multitask learning algorithm that learns to perform each task jointly with learning inter-task relationships. The primary intuition we leveraged here is that task performance can be improved both by querying external oracles and by querying peer tasks. The former incurs a cost or at least a query-budget bound, but the latter requires no human attention. Hence, our hypothesis was that with bounded queries to the human expert, additionally querying peers should improve task performance. Querying peers requires estimating the relation among tasks. The key idea is based on smoothing the loss function of each task w.r.t. a probabilistic distribution over all tasks, and adaptively refining such distribution over time. In addition to closed-form updating rules, we provided a theoretical bound on the expected number of mistakes. The effectiveness of our algorithm is empirically verified over three benchmark datasets where in all cases task accuracy improves both for PEERsum (sum of peer recommendations weighted by task similarity) and PEERone (peer recommendation from the most highly related task) over baselines such as assuming task independence.

Chapter 4

Lifelong Learning of Multiple Tasks

Lifelong learning, inspired by established human learning principles, works by accumulating and retaining the knowledge from the past and leverages this knowledge to acquire new skills and solve new problems efficiently. It poses considerable challenges in terms of effectiveness and overall computational tractability for real-time performance. Unlike traditional multitask learning, where the tasks are presented simultaneously and an entire training set is available to the learner ([21]), in lifelong learning the tasks arrive sequentially ([106]).

When it comes to lifelong learning, it is natural to ask, Is it possible to use an existing multitask algorithm to solve a lifelong learning problem? We propose a novel framework based on self-paced learning (SP) which utilizes a curriculum defined dynamically by the learner ("self-paced") instead of a fixed curriculum set *a-priori* by a teacher. It allows us to use a certain class of existing multitask algorithms for solving lifelong learning setting.

The proposed approach starts with an easier set of tasks, and gradually introduces more difficult ones to build the shared knowledge base. Our proposed method provides a natural way to specify the trade-off between choosing the easier tasks to update the shared knowledge and learning new tasks using the knowledge acquired from previously learned tasks. Our proposed algorithm based on self-paced learning for multiple tasks addresses these three key challenges: 1) it embeds task selection into the model learning; 2) it gradually learns the shared knowledge at the system's own pace; 3) it is generalizable to a wider group of existing multitask algorithms such that they can be easily adapted to lifelong learning setting.

In addition, we propose a co-clustering approach to multitask and lifelong learning. In several applications, we encounter clusters and overlapping groups among the tasks at hand. We provide a flexible way to cluster both the features and the tasks using the shared feature representation and task relationship matrix. Learning task relationships has been shown beneficial in (positive and negative) transfer of knowledge from information-rich tasks to information-poor tasks [120], whereas the shared feature representation has been shown to perform well when each task has a limited number of training instances (observations) compared to the total number across all tasks [9].

Existing research in multitask learning considers either the first approach and learns a task relationship matrix in addition to the task parameters, or relies on the latter approach and learns a shared latent feature representation from the task parameters. To the best of our knowledge, there is no prior work that utilizes both principles jointly for multitask learning. We propose a

new approach that learns a shared feature representation along with the task relationship matrix jointly to combine the advantages of both principles into a general multitask learning framework. Our proposed approach is closely related to Output Kernel Learning (*OKL*) from the previous chapter where we learn the kernel between the components of the output vector for problems such as multi-output learning, multitask learning, etc. We extend the proposed methods to the lifelong learning setting where the shared feature representation learned from the previous tasks is used to solve unseen tasks.

Later in this chapter, we consider a continuous lifelong learning setting in which both the tasks and the examples of the tasks arrive in an online fashion, without any predetermined order. We propose a novel method called *Online Output Kernel Learning Algorithm* (OOKLA) for lifelong learning setting by jointly re-estimating the inter-task relationships (*output* kernel) and the per-task model parameters at each round. To avoid the memory explosion, a robust budget-limited version of the proposed algorithm is introduced, which efficiently utilizes the relationships between the tasks to bound the total number of representative examples in the support set.

Related Work

The concept of lifelong learning was first proposed by Thrun and Mitchell [1995] in autonomous robot learning. There has been a very little progress in lifelong learning until very recently. Never-Ending Language Learner (NELL) continuously learns to extract facts from the web and updates its confidence in the previous beliefs [75]. Curriculum learning [17] and self-paced learning [65] defines either a predefined or adaptive curriculum for the learner to acquire knowledge in a progression of easier-to-harder tasks. Efficient Lifelong Learning Algorithm (*ELLA*), on the other hand, learns shared basis from tasks that arrive sequentially and new tasks are represented as a sparse linear combination of the columns of this shared basis [95]. This work was further extended to policy gradient search for lifelong learning setting [7]. The recent survey of lifelong learning approaches and their challenges can be found in [98]. This chapter provides an efficient way to learn from multiple related tasks in the lifelong learning setting.

Most lifelong learning approaches use a single model for all the tasks or reuse the models from the previous tasks to build a model for the new task ([5, 12, 51, 84, 85, 89, 93, 95]). These approaches either increase the computation time on iterations where we encounter a novel task or reduce the prediction power of the model learned from the previous tasks due to catastrophic forgetting. To the best of our knowledge, relationships among the tasks have not been successfully exploited in the lifelong learning setting due to the difficulty in learning a positive semi-definite task relationship matrix in large-scale applications.

In related works, Cavallanti et al. [25] assume that relationships between the tasks are available *a priori*. However often such task-relation prior knowledge is either unavailable or infeasible to obtain for many applications especially when the number of tasks K is large ([112]) and/or when the manual annotation of task relationships is expensive ([62]). [96] formulated the learning of task relationship matrix as a Bregman-divergence minimization problem w.r.t. positive definite matrices. The model suffers from high computational complexity as semi-definite programming is required when updating the task relationship matrix at each online round. We show that with a different formulation, we can obtain a similar but much cheaper updating rule for learning the inter-task weights. [79] proposed an efficient method for learning the task relationship matrix

using the cross-task performance measure, but their approach learns only the positive correlation between the tasks. Our proposed approach learns positive and negative correlations between the tasks for robust transfer of knowledge from the previously learned tasks.

Recent work in output kernel learning estimate the task covariance matrix in RKHS space, inferred it directly from the data ([40, 54, 100]). The task covariance matrix is called the output kernel defined on the tasks, similar to the scalar kernel on the inputs. Most recently, [54] showed that for a class of regularization functions, we can efficiently learn this output kernel. Unfortunately most of the proposed methods for learning output kernels require access to the entire data for the learning algorithm, a luxury unavailable in online learning and especially in the lifelong learning setting.

4.1 Self-Paced Multitask Learning with Shared Knowledge

Self-paced learning, inspired by established human education principles, defines a new machine learning paradigm based on a curriculum defined dynamically by the learner ("self-paced") instead of a fixed curriculum set *a-priori* by a teacher [17]. It is an iterative approach that alternatively learns the model parameters and selects easier instances at first, progressing to harder ones [65]. However, naive extension of self-paced learning to the multitask setting may result in intractable increases in the number of learning parameters and therefore inefficient use of shared knowledge among the tasks. Existing work in this area is not scalable and/or lacks sufficient generality to apply to several multitask learning challenges [69].

Not all tasks are equal. Some tasks are easy to learn and some tasks are complex, facilitated by previously learned tasks to solve it efficiently. For example, classification task of whether an image has a bird or not can be learned by solving easier component tasks first such as *Is there a wing?*, *Is there a beak?*, *Does it have feathers?*, etc. The knowledge learned from these previously learned easier tasks can be used to solve the complex tasks effectively and such shared knowledge plays an important role in transfer of information between these tasks. This phenomenon is more evident in many real-world data such as object detection, weather prediction, landmine detection, etc.

We introduce a new learning framework for multiple tasks that addresses the aforementioned issues. It starts with easier set of tasks, and gradually introduces more difficult ones to build the shared knowledge base. Our proposed method provides a natural way to specify the trade-off between choosing the easier tasks to update the shared knowledge and learning new tasks using the knowledge acquired from previously learned tasks. Our proposed framework based on self-paced learning for multiple tasks addresses these three key challenges: 1) it embeds task selection into the model learning; 2) it gradually learns the shared knowledge at the system's own pace; 3) it is generalizable to a wider group of multitask problems.

We first briefly introduce the self-paced learning framework. Next, we describe our proposed approach for self-paced multitask learning with efficient learning of latent task weights. We give a probabilistic interpretation of these task weights, based on their training errors. We apply our learning framework to a few popular multitask problems such as Multitask Feature Learning, Multitask Learning with Alternating Structure Optimization (ASO), Mean regularized Multitask Learning and show that self-paced multitask learning significantly improves the learning

performance of the original problem. In addition, we evaluate our method against several algorithms for sequential learning of multiple tasks.

4.1.1 Background

Given a set of N training instances along with their labels $(x_i, y_i)_{i\in[N]}$, the general form of the objective function for single task learning is given by:

$$\mathcal{E}_\lambda\{\hat{\mathbf{w}}\} = \arg\min_{\mathbf{w}} \sum_{i\in[N]} \ell(y_i, f(x_i, \mathbf{w})) + \rho_\gamma(\mathbf{w}) \tag{4.1}$$

where $\rho_\gamma(\mathbf{w})$ is the regularization term on the model parameters and typically it is set to $\rho_\gamma(\mathbf{w}) = \gamma||\mathbf{w}||_2^2$ (*ridge* or L2 penalty) or $\gamma||\mathbf{w}||_1$ (*lasso* or L1 penalty). γ is the regularization parameter and $[N]$ is the index set $\{1, 2, \ldots N\}$

Self-paced learning (*SPL*) provides a strategy for simultaneously selecting the easier instances and re-estimating the model parameters $\mathbf{w}$ at each iteration [65]. We assume a linear predictor function $f(x_i, \mathbf{w})$ with unknown parameter $\mathbf{w}$. Self-paced learning solves the following objective function:

$$\mathcal{E}_\lambda\{\hat{\mathbf{w}}, \hat{\tau}\} = \arg\min_{\mathbf{w}, \tau\in\Omega} \sum_{i\in[N]} \tau_i \ell(y_i, f(x_i, \mathbf{w})) + \rho_\gamma(\mathbf{w}) + \lambda r(\tau) \tag{4.2}$$

where $r(\tau)$ is the regularization term, Ω is the domain space of τ, $\rho_\gamma(\mathbf{w})$ is the regularization term on model parameters $\mathbf{w}$ as defined earlier, and λ is the regularization parameter that identifies the difficulty of the instances. There are two unknowns in equation 4.2: model parameter vector $\mathbf{w}$ and the selection parameter τ (restricted to the domain Ω).

A common choice of the constraint space $\mathcal{C} = \{\rho_\gamma(\mathbf{w}), r(\tau), \Omega\}$ in *SPL* is $\{\gamma||\mathbf{w}||_2^2, -||\tau||_1\}, \{0, 1\}^N\}$. See [55] for more examples on the constraint space. With this setting, equation 4.2 is a bi-convex optimization problem over $\mathbf{w}$ and τ, which can be efficiently solved by *alternating minimization*. Given a fixed τ, the solution for $\mathbf{w}$ can be obtained using any off-the-shelf solver and for a fixed $\mathbf{w}$, solution for τ can be given as follows:

$$\hat{\tau}_i = \begin{cases} 1 & \text{if } \ell(y_i, f(x_i, \mathbf{w})) < \lambda \\ 0 & otherwise \end{cases} \quad \forall i \in [N] \tag{4.3}$$

There exists an intuitive explanation for this alternative search strategy: 1) when updating τ with a fixed w, a sample whose loss is smaller than a certain threshold λ is taken as an "easy" sample because it is a sample with "*less error*", and will be selected in training ($\tau_i^* = 1$) or otherwise unselected ($\tau_i^* = 0$); 2) when updating w with a fixed τ, the classifier is trained only on the selected "easy" samples. When λ is small, only "easy" samples with small losses will be considered.

4.1.2 Learning with Shared Knowledge

Suppose we are given T tasks where the t-th task is associated with N_t training examples. Denote by $\left\{(x_i^t, y_i^t)\right\}_{i=1}^{N_t}$ and $\mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) = \frac{1}{N_t}\sum_{i\in[N_t]} \ell(y_i^t, f(x_i^t, \mathbf{w}_t))$ the training set and loss for

task t, respectively. In this section, we consider a more general formulation for multitask learning, which is given by [15, 20, 44]:

$$\mathcal{E}\{\hat{\mathbf{W}}, \hat{\mathbf{\Theta}}\} = \arg\min_{\mathbf{W}, \mathbf{\Theta} \in \mathbf{\Gamma}} \sum_{t \in [T]} \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) + P_\gamma(\mathbf{W}, \mathbf{\Theta}) \tag{4.4}$$

where $P_\gamma(\mathbf{W}, \mathbf{\Theta})$ is the regularization term on task parameters $\mathbf{W}$, $\mathbf{\Theta}$ is the knowledge shared among the tasks which depends on the problem under consideration. We assume that $P_\gamma(\mathbf{W}, \mathbf{\Theta})$ can be written as $\sum_{t \in [T]} P_\gamma(\mathbf{w}_t, \mathbf{\Theta})$, such that, for a given $\mathbf{\Theta}$, the above objective function decomposes into T independent optimization problems. $P_\gamma(\mathbf{w}_t, \mathbf{\Theta})$ gives a scoring function on how easier the task is, compared to that of the learned knowledge $\mathbf{\Theta}$. Several multitask learning problems fall under this general characterization. For example, Multitask Feature Learning (*MTFL*), Regularized Multitask Learning (*MMTL*), Multitask learning with manifold regularization (*MTML*), Multitask learning via Alternating Structure Optimization (*MTASO*), Sparse coding for multitask learning (*SC-MTL*), etc [4, 8, 43, 44, 74]. With this formulation, one can easily extend the SPL framework to multitask setting, by considering instance weights for each task.

$$\mathcal{E}_\lambda\{\hat{\mathbf{W}}, \hat{\mathbf{\Theta}}, \hat{\tau}\} = \arg\min_{\substack{\mathbf{W}, \mathbf{\Theta} \in \mathbf{\Gamma} \\ \tau \in \Omega}} \sum_{t \in [T]} \frac{1}{N_t} \sum_{i \in [N_t]} \tau_{ti} \ell(y_i^t, f(x_i^t, \mathbf{w}_t)) + P_\gamma(\mathbf{W}, \mathbf{\Theta}) + \lambda r(\tau) \tag{4.5}$$

But there are two key issues with this naive extension of *SPL*: 1) The above formulation fails to effectively utilize the knowledge shared among the tasks; 2) The number of unknown parameters τ grows with the total number of instances $N = \sum_t N_t$ from all the tasks. This is a serious problem especially when the number of tasks T is large [112] and/or when manual annotation of task instances is expensive [62].

To address these issues, we consider task-level weights, instead of instance-level weights. Our motivation behind this approach is based on the human educational process. When students learn a new concept, they (or their teachers) choose a new task that is relevant to their recently-acquired knowledge, rather that more distant tasks or concepts or other haphazard selections. Inspired by this interpretation, we propose the following objective function for Self-Paced Multitask Learning (*sp*MTL):

$$\mathcal{E}_\lambda\{\hat{\mathbf{W}}, \hat{\mathbf{\Theta}}, \hat{\tau}\} = \arg\min_{\substack{\mathbf{W}, \mathbf{\Theta} \in \mathbf{\Gamma} \\ \tau \in \Omega}} \sum_{t \in [T]} \tau_t \big[\mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) + P_\gamma(\mathbf{w}_t, \mathbf{\Theta})\big] + \lambda r(\tau) \tag{4.6}$$

Note that the number of parameters τ_t depends on T instead of N and the τ_t depends on both the training error of the task and the task regularization term for the shared knowledge $\mathbf{\Theta}$.

The pseudo-code is in Algorithm 7. The learning algorithm defines a task as "easy" task if it has low training error $\frac{1}{N_t} \sum_{i \in [N_t]} \ell(y_i, f(x_i, \mathbf{w}_t))$ and similar to the shared knowledge representation $P_\gamma(\mathbf{w}_t, \mathbf{\Theta})$. These tasks will be selected in building the shared knowledge $\mathbf{\Theta}$. Following Equation

Algorithm 7: Self-Paced Multitask Learning: A General Framework

Input : $\mathcal{D} = \{(\mathbf{X}_t, \mathbf{y}_t)\}_{t=1}^{T}, \mathbf{\Theta}^{(0)}, c > 1$
Output : $\mathbf{W}, \mathbf{\Theta}$
$k \leftarrow 1, \lambda \leftarrow \lambda_0$
repeat
  Solve for $\mathbf{w}_t^{(k)} \leftarrow \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w})) + P_\gamma(\mathbf{w}, \mathbf{\Theta}^{(k-1)})\ \forall t$;
  Solve for $\tau^{(k)}$ using equation (4.7) or equation (4.8) ;
  Solve for $\mathbf{\Theta}^{(k)}$:
  $\mathbf{\Theta}^{(k)} \leftarrow \arg\min_{\mathbf{\Theta}} \sum_{t\in[T]} \tau_t^{(k)} P_\gamma(\mathbf{w}_t^{(k)}, \mathbf{\Theta})$;
  $\lambda \leftarrow c\lambda$;
  $k \leftarrow k + 1$;
until $\|\tau^{(k)} - \tau^{(k-1)}\|_2^2 \leq \epsilon$;

4.3, by setting $\mathcal{C} = \{\gamma||\mathbf{w}||_2^2, -||\tau||_1, [0,1]^N\}$, we can define τ_t as [1]:

$$\hat{\tau}_t = \begin{cases} 1 & \text{if } \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t^{(k)})) + P_\gamma(\mathbf{w}_t^{(k)}, \mathbf{\Theta}^{(k-1)}) < \lambda \\ \delta & otherwise \end{cases} \quad \forall t \in [T] \tag{4.7}$$

For multitask setting, it is desirable to consider an alternative constraint space that gives probabilistic interpretation for τ. By setting $\mathcal{C} = \{\gamma||\mathbf{w}||_2^2, -\mathbf{H}(\tau), \Delta^{N-1}\}$, we get

$$\hat{\tau}_t \propto \exp(-[\mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) + P_\gamma(\mathbf{w}_t, \mathbf{\Theta})]/\lambda), \tag{4.8}$$

where $\mathbf{H}(\tau) = -\sum_{t\in[T]} \tau_t \log \tau_t$ denotes the entropy of the probability distribution τ over the tasks. The key idea is that the algorithm, at each iteration, maintains a probability distribution over the tasks to identify the simpler tasks based on the shared knowledge. Similar approach has been used in learning relationship between multiple tasks in an online setting [79]. Using this representation, we can use τ to sample, at each iteration, the "easy" tasks and thus makes the learning problem scalable using stochastic approximation when the number of tasks is large. It is worth noting that our framework can easily handle outlier tasks by a simple modification to Algorithm 7. Since outlier tasks are different from the main tasks and are usually difficult to learn, we can take advantage of this simple observation for early stopping, before the algorithm visits all the tasks [91].

Our algorithm can be easily generalized to other types of updating rules by replacing exp in (4.8) with other functions. In latter cases, however, τ may no longer have probabilistic interpretations. Algorithm 7 shows the basic steps in learning the task weights and the shared knowledge. The algorithm uses an additional parameter $'c'$ that controls the learning pace of the self-paced procedure. Typically, $'c'$ is set to some value greater than 1 (in our experiments, we set it to 1.1) such that, at each iteration, the threshold λ is relaxed to included more tasks. The input to the algorithm also takes $\mathbf{\Theta}^{(0)}$, initial knowledge about the domain and can be initialized based on some external sources.

[1] For correctness of the algorithm, we set $\tau_t = \delta$ for the hard tasks, instead of $\tau_t = 0$ with $\delta = 0.01$.

Handling Outlier Tasks

Our framework can easily handle outlier tasks by a simple modification to Algorithm 7. Since outlier tasks are different from the main tasks and are usually difficult to learn, we can take advantage of this simple observation for early stopping, before the algorithm visits all the tasks (i.e., $\mathcal{A} = [T]$). Here, we introduce two early stopping criteria to achieve our goal. First, we consider the maximum value that the λ can take: $\lambda_{\max}$. Second, we learn the joint $\mathbf{\Theta}$ until our active set contains consider number of tasks in the pool: $|\mathcal{A}| < T$.

4.1.3 Motivating Examples

We give three examples to motivate our self-paced learning procedure. We briefly discuss how our algorithm alters the learning pace of the original problem. Note that the existing implementation of these problems can be easily "self-paced", by simply adding a few lines of code to get a better performance of the original problem. We refer the readers to [4, 8, 43] for additional background.

Example 1: Self-Paced Mean Regularized Multitask Learning (*sp*MMTL)

Mean Regularized Multitask learning assumes that all task parameters are close to some fixed parameter $\mathbf{w}_0$ in the parameter space. *sp*MMTL learns τ to select the easy tasks based on the distance of each task parameter $\mathbf{w}_t$ from $\mathbf{w}_0$.

$$\mathcal{E}_{MMTL,\lambda} = \underset{\substack{\{\mathbf{w}_1,\mathbf{w}_2,\dots\mathbf{w}_T\}\\ \mathbf{w}_0,\tau\in\Omega}}{\arg\min} \sum_{t\in[T]} \tau_t \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) + \gamma||\mathbf{w}_t - \mathbf{w}_0||_2^2 + \lambda||\tau||_1 \tag{4.9}$$

In the above objective function, we can get the closed-form solution for $\mathbf{w}_0$ as $\mathbf{w}_0 = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_t$ which is the mean of the task parameters.

Example 2: Self-paced Multitask Feature Learning (*sp*MTFL) Multitask feature learning learns a common feature representation $\mathbf{D}$ shared across multiple related tasks. In addition to learning the task parameters and the shared feature representation, *sp*MTFL learns τ to select the easy tasks first, defined by the learning parameter λ. The algorithm starts with these easy tasks to learn the shared feature representation which is used for solving progressively harder tasks.

$$\mathcal{E}_{MTFL,\lambda} = \underset{\substack{\{\mathbf{w}_1,\mathbf{w}_2,\dots\mathbf{w}_T\}\\ \mathbf{D}\in\mathbf{S}_{++}^{d}\\ \tau\in\Omega}}{\arg\min} \sum_{t\in[T]} \tau_t \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) + \gamma \sum_{t\in[T]} \tau_t \langle \mathbf{w}_t, \mathbf{D}^{-1}\mathbf{w}_t\rangle + \lambda r(\tau) \tag{4.10}$$

The value of τ_t determines the importance of a task in learning this shared feature representation, i.e., tasks with high probability contributes more towards learning $\mathbf{D}$ than the tasks with low probability.

Example 3: Self-paced Multitask learning with Alternating Structure Optimization (*sp*MTASO)

Alternating Structure Optimization learns a shared low-dimensional predictive structure $\mathbf{U}$ on a hypothesis space from multiple-related tasks. This low-dimensional structure along with the

low-dimensional model parameters $\mathbf{v}_t$ are learned gradually from easy tasks guided by τ.

$$\mathcal{E}_{MTASO,\lambda} = \underset{\substack{\{\mathbf{w}_1,\mathbf{w}_2,\ldots\mathbf{w}_T\}\\ \mathbf{U}\mathbf{U}^\top = I_{h\times h}\\ \tau\in\Omega}}{\arg\min} \sum_{t\in[T]} \tau_t \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) + \gamma \sum_{t\in[T]} \tau_t ||\mathbf{w}_t - \mathbf{U}^\top \mathbf{v}_t||_2^2 + \lambda r(\tau) \tag{4.11}$$

Example 4: Self-paced Multitask learning with Manifold regularization (*sp*MTML) Manifold-regularized multitask learning assumes that all the task parameters lie on a manifold $\mathcal{M}$. It alternatively learns the task parameters and the manifold. Our proposed learning algorithm 7 chooses the tasks that are good representatives for constructing a reliable manifold structure. The objective function for *sp*MTML is given as follows:

$$\mathcal{E}_{MTML,\lambda} = \underset{\substack{\{\mathbf{w}_1,\mathbf{w}_2,\ldots\mathbf{w}_T\}\\ \mathbf{W}^{\mathcal{M}}\\ \tau\in\Omega}}{\arg\min} \sum_{t\in[T]} \tau_t \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{w}_t)) + \gamma \sum_{t\in[T]} \tau_t ||\mathbf{w}_t - \mathbf{w}_t^{\mathcal{M}}||_2^2 + \lambda r(\tau) \tag{4.12}$$

where $\mathbf{w}_t^{\mathcal{M}} = g(h(\mathbf{w}_t))$ is the projection distance of $\mathbf{w}_t$ from the manifold $\mathcal{M}$.

4.1.4 Experiments

All reported results in this section are averaged over 10 random runs of the training data. Unless otherwise specified, all model parameters are chosen via 3-fold cross validation. For all the experiments, we update the τ values using the equation 4.8. We evaluate our self-paced multitask learning algorithm on the four well-known multitask problems (MMTL, MTFL, MTASO), briefly discussed in the previous section. We also compare our results with Independent multitask learning (ITL) where each task is learned independently and Single-task learning (STL) where we learn a single model by pooling together data from all the tasks.

Synthetic Experiment

Synthetic data *(syn1)* consists of 30 tasks that belong to 3 groups of tasks with 15 training examples per task. We generate the task parameters as in [56]. Each example consists of 20 features. We randomly select a subset of tasks and increase their variance to $(\sigma = 25)$, and variances for the rest of the tasks are set to be low $(\sigma = 5)$ in order to simulate the difference between easy and hard tasks. With this setting, we expect that our self-paced learning algorithm should be able to learn the shared knowledge from the easier tasks and use this knowledge to improve the performance of the harder tasks.

Synthetic data *(syn2)* consists of 30 tasks with 15 training examples per task as before. We randomly generate a 30-dimensional vector $(s_1, s_2, s_3, \ldots, s_{30})$ such that the parameter for each task t is given as $\mathbf{w}_t = (s_1, s_2, \ldots s_t, 0, 0, \ldots, 0)$ and each example consists of 30 features. The dataset is constructed in such a way that learning the task t is easier than learning the task $t+1$ and so on.

The result for *syn1* and *syn2* are shown in Table 4.1. We report the RMSE (mean and std) of our methods. All of our self-paced methods perform better than their baseline methods on average in both the synthetic datasets. Figure 4.1 (bottom-left) shows the τ learned using self-paced

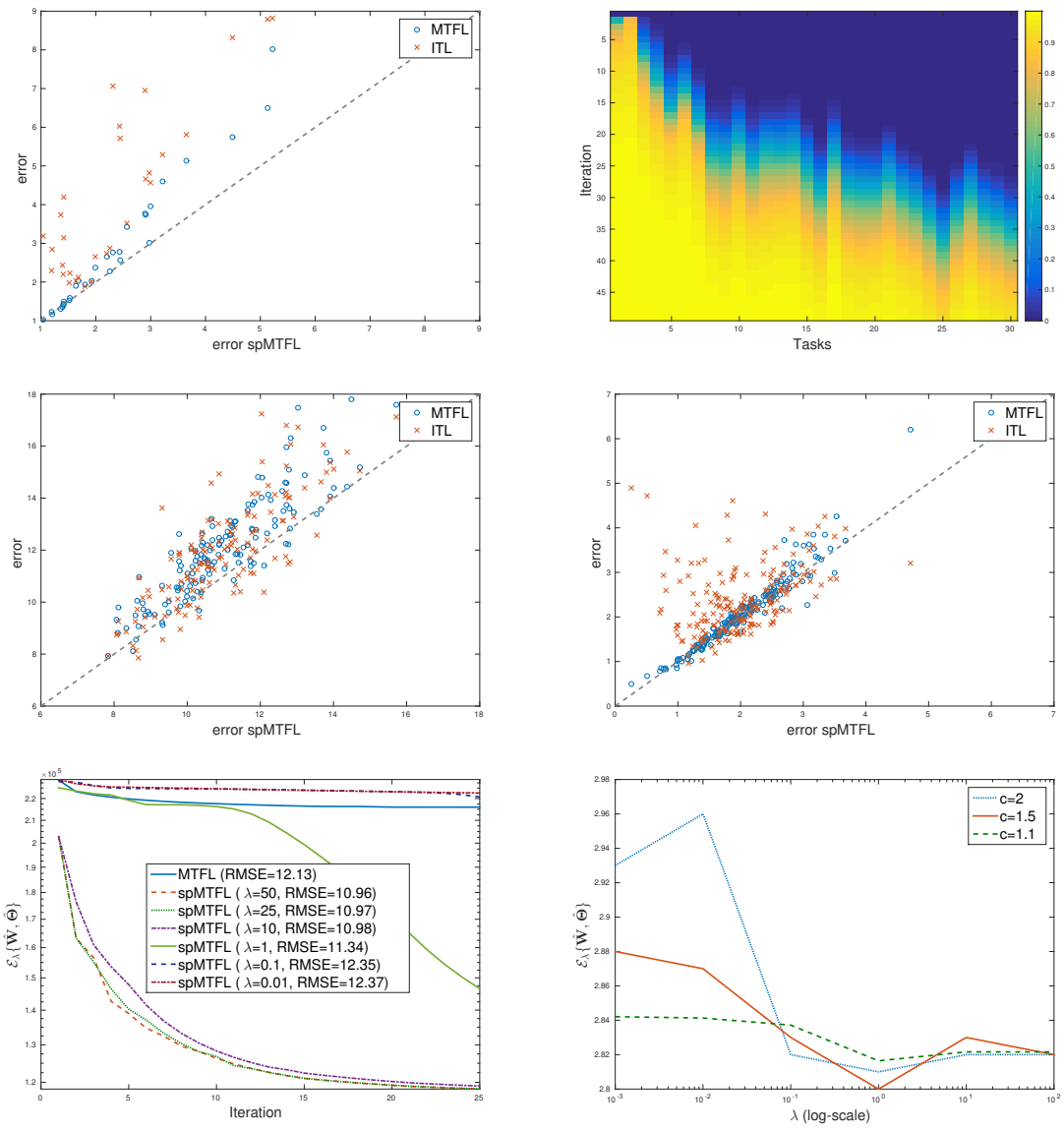

Figure 4.1: Error of MTFL and ITL vs. Error of *sp*MTFL calculated for *syn2* dataset (Top-left). Values of $\hat{\tau}$ from *sp*MTFL at each iteration calculated for *syn2* dataset (Top-right). Error of MTFL and ITL vs. Error of *sp*MTFL calculated for *school* and *cs* datasets (Middle). Convergence of the algorithm with varying threshold λ (Bottom-left) calculated from *sp*MTFL for *school* dataset. Convergence of the algorithm with different learning pace $'c'$ (Bottom-right) calculated from *sp*MTFL for *cs* dataset. The experiment shows $'c' = 1.1$ for learning pace yields a stable performance.

task selection (*sp*MTFL) at each iteration. We can see that the tasks are selected based on their difficulty and the number of features used in each task. Figure 4.1 (top-left) shows the task-specific test errors for *syn2* dataset (*sp*MTFL vs. their corresponding baseline methods MTFL and ITL). Each red point in the plot compares the RMSE of ITL with *sp*MTFL and each blue point compares

Models	*syn1*	*syn2*	*school*	*cs*	*sentiment*	*landmine*
STL	1.60 (0.02)	4.16 (0.09)	12.13 (0.08)	2.45 (0.13)	58.49 (0.40)	74.11 (0.50)
ITL	1.13 (0.07)	3.25 (0.10)	12.00 (0.04)	1.99 (0.14)	68.39 (0.34)	74.39 (1.11)
MMTL	1.12 (0.07)	3.24 (0.10)	12.10 (0.08)	1.99 (0.18)	68.54 (0.27)	75.50 (1.86)
spMMTL	**1.03 (0.05)**	3.24 (0.10)	**10.34 (0.06)**	1.89 (0.10)	68.54 (0.26)	75.73 (1.29)
MTFL	0.81 (0.06)	2.82 (0.13)	12.06 (0.08)	1.91 (0.18)	68.91 (0.31)	75.67 (1.03)
spMTFL	**0.73 (0.05)**	**2.34 (0.12)**	**10.99 (0.08)**	1.87 (0.15)	**75.60 (0.17)**	**76.92 (1.06)**
MTASO	0.56 (0.03)	2.66 (0.16)	11.14 (0.10)	1.38 (0.19)	72.03 (0.18)	72.58 (1.46)
spMTASO	0.52 (0.03)	**2.54 (0.14)**	11.14 (0.11)	**1.12 (0.17)**	72.36 (0.19)	**75.73 (1.46)**

Table 4.1: Average performance on six datasets: means and standard errors over 10 random runs. We use RMSE as our performance measure for *syn1*, *syn2*, *school*, and *cs* and Area under the curve (AUC) for *sentiment* and *landmine*. Self-paced methods with the best performance against their corresponding MTL baselines (paired t-tests at 95% significance level) are shown in boldface.

the RMSE of MTFL vs. *sp*MTFL. Points above the line $y = x$ show that the self-paced methods does better than ITL or their MTL baseline methods. From the (MTFL vs. *sp*MTFL) plot, we can see that our self-paced learning method *sp*MTFL achieves significant improvement on harder tasks (blue points in top-right) compared to the easier tasks (blue points in bottom-left). Based on our learning procedure, these harder tasks must have been learned at the later part of the learning and thus efficiently utilize the knowledge learned from the easier tasks to improve their performances. Similar behaviour can be observed in the other two plots. Note that some of the points fall slightly below the $y = x$ line, but since the decrease in performance of these tasks are small, it has very little impact on the overall score. We believe this can be avoided if we tune different regularization parameter λ_t for each task. However, this will increase the number of parameters to tune in addition to the task weight parameters τ.

Evaluation on Real Data

We use the following benchmark real datasets for our experiments on self-paced multitask learning.

London School data (*school*) consists of examination scores of $15,362$ students from 139 schools in London. Each school is considered as a task and the feature set includes year of the examination, four school-specific and three student-specific features. We replace each categorical feature with one binary variable for each possible feature value, as suggested in [9]. This results in 26 features with additional feature to account for the bias term. We use the ten $20\% - 80\%$ train-test splits that came with the dataset for our experiments.

Computer Survey data (*cs*) was collected from the ratings of 190 students on each of the 20 different personal computers. Each student here is considered as a single task and the rating ranges from $0 - 10$. There are 20 observations in each task. Each computer is represented by

13 different features such as RAM, cache-size, CPU speed, etc. We add an additional feature to account for the bias term. Train-test splits are obtained by selecting $75\% - 25\%$, thus giving 15 examples for training and 5 examples for test set.

Sentiment Detection data (*sentiment*) contains reviews from 14 domains. The reviews are represented by a bag of unigram/bigram *TF-IDF* features from a dictionary of size $28,775$. Each review is associated with a rating from $\{1, 2, 4, 5\}$. We select $1,000$ reviews for each domain and create two tasks (500 reviews per task), based on whether the rating is 5 or not and whether the rating is 1 or not, in order to represent the different levels of sentiment. This gives us 28 binary classification tasks. We use 120 reviews per task for training and the rest of the reviews for test set.

Landmine Detection data (*landmine*) consists of 19 tasks collected from different landmine fields. Each task is a binary classification problem: landmines $(+)$ or clutter $(-)$ and each example consists of 9 features extracted from radar images. Landmine data is collected from two different terrains: tasks 1-10 are from highly foliated regions and tasks 11-19 are from desert regions, therefore tasks naturally form two clusters. We use 80 examples from each task for training and the rest as the test data. We repeat the experiments on 10 (stratified) splits to measure the performance reliably. Since the dataset is highly skewed, we use AUC score to compare our results.

Table 4.1 summarizes the performance of our methods on the four real datasets. We can see that our proposed self-paced learning algorithm does well on almost all datasets. As in our synthetic experiments, we observe that *sp*MTFL performs significantly better than MTFL, which is a state-of-the-art method for multitask problems. It is interesting to see that when the self-paced learning procedure doesn't help the original algorithm, it doesn't perform worse than the baseline results. In such cases, our self-paced learning algorithm gives equal probability to all the tasks $(\tau_t = \frac{1}{T}, \forall t \in [T])$ within the first few iterations. Thus the proposed self-paced methods reduce to their original methods and the performance of the self-paced methods are on par with their baselines.

We also notice that if a dataset doesn't adhere to the assumptions of a model, such as task parameters lie on a manifold or low-dimensional space, then our self-paced methods result in little improvement, as it can be seen in *cs* (and also in *sentiment* for *sp*MTASO). It is worth mentioning that our proposed self-paced multitask learning algorithm does exceptionally better in *school*, which is a benchmark dataset for multitask experiments in the existing literature [4, 63]. Our proposed methods achieve as much as 14% improvement over their baselines on some experiments. Figures (top-middle) and (top-right) show the task-specific errors for *school* and *cs* dataset. We can see similar pattern as in *syn2*. The easier tasks learned at an earlier stage help the harder tasks at the later stages as it is evident from these plots.

Comparing *sp*MTFL with Sequential Learning Algorithms

In this section, we briefly review two learning methods and compare them to our proposed learning algorithm. Both these methods learn from multiple tasks sequentially in a specific order to either improve the learning performance or to speedup the algorithm. Pentina et al. (2015) propose a curriculum learning method (*CL*) for multiple tasks to find the best order of tasks to be learned based on training error. The tasks are solved in a sequential manner based on this order by

transferring information from the previously learned tasks to the next ones through shared task parameters. They show that this sequential learning of tasks in a meaningful order can be superior than solving the tasks simultaneously. The objective function of *CL* for learning the best task order and the task parameters is given as follows:

$$\mathcal{E}_{CL} = \underset{\substack{\{\mathbf{w}_1,\mathbf{w}_2,\ldots\mathbf{w}_T\} \\ \pi\in\Psi_T}}{\arg\min} \sum_{t\in[T]} \mathcal{L}(\mathbf{y}_{\pi(t)}, f(\mathbf{X}_{\pi(t)}, \mathbf{w}_{\pi(t)})) + \gamma \sum_{t\in[T]} ||\mathbf{w}_{\pi(t)} - \mathbf{w}_{\pi(t-1)}||_2^2 \tag{4.13}$$

where Ψ_T is the symmetric group of all permutations over $[T]$. Since, minimizing with respect to all possible permutations $\pi \in \Psi_T$ is an expensive combinatorial problem, they suggest a greedy, incremental procedure for approximating the task order. Their method shares with ours the motivation of learning from easier tasks first, and then gradually add more difficult tasks, based on training errors. But unlike our proposed method, which utilizes shared knowledge from all previous tasks, their method does not allow sharing between different levels of task relatedness. In addition, the Euclidean distance based regularization in their objective function forces the parameter of newly learned task to be similar to its immediate predecessor. This more myopic approach can be a restrictive assumption for many applications.

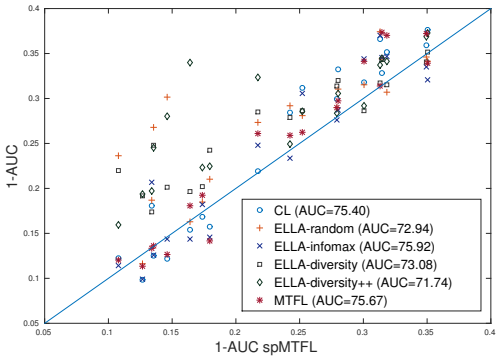

Figure 4.2: Average performance on *landmine* for sequential learning algorithms and *sp*MTFL: means and standard errors over 10 random runs. We use $(1 - AUC)$ score as our performance measure for comparison. Mean AUC score is shown in the bracket.

Perhaps the most relevant work to ours in the context of lifelong learning is from [95], which learns the shared basis $\mathbf{L}$ from tasks that arrives sequentially. They propose an efficient online multitask learning algorithm (*ELLA*) that allows the transfer of knowledge from previously learned tasks to the new tasks using this shared basis. The task parameters are represented as a sparse linear combination of the columns of the shared basis $\mathbf{w}_t = \mathbf{L}\mathbf{s}_t$. The motivation for *ELLA* and our method are significantly different. Whereas *ELLA* tries to achieve nearly identical to the performance of batch MTL with increased speedup in learning, our proposed method focuses on improving the learning performance over that of the original algorithm, with minimal changes to said original algorithm. Unlike our proposed method, *ELLA* cannot be easily generalized to existing multitask problems. It only uses efficient update equations specific to their proposed objective function.

ELLA optimizes the following objective function:

$$\mathcal{E}_{ELLA} = \underset{\mathbf{L},\{\mathbf{s}_1,\mathbf{s}_2,\ldots\mathbf{s}_T\}}{\arg\min} \frac{1}{T} \sum_{t\in[T]} \mathcal{L}(\mathbf{y}_t, f(\mathbf{X}_t, \mathbf{L}\mathbf{s}_t)) + \mu\|\mathbf{s}_t\|_1 + \lambda\|\mathbf{L}\|_F^2 \tag{4.14}$$

Finally, we compare our self-paced multitask learning algorithm against these sequential multitask learning algorithms (curriculum learning for multiple tasks [86] and efficient lifelong learning [94, 95] [2]. We choose *sp*MTFL for comparison based on its overall performance in the previous experiments. We use landmine dataset for evaluation. We use different variant of *ELLA* for fair comparison against our proposed approach. The original *ELLA* algorithm assumes that the tasks arrive randomly and the lifelong learner has no control over their order (*ELLA-random*).

Ruvolo and Eaton (2013) show that if the learner can choose the next task actively, it can improve the learning performance using as few tasks as possible. They proposed two active task selection procedures for choosing the next best task: 1) Information Maximization (*ELLA-infomax*) chooses the next task to maximize the expected information gain about the basis $\mathbf{L}$; 2) Diversity (*ELLA-diversity*) chooses the next task as the one that the current basis $\mathbf{L}$ is doing the worst performance. Both these approaches select the tasks that are significantly different from the previously learned tasks (*active task selection*), rather than a progression of tasks that build upon each other. Our proposed method selects the task based on the training error and its relevance to the shared knowledge learned from the previous tasks (*self-paced task selection*).

Figure 4.2 shows the task-specific test performance results for this experiment on *landmine* dataset. We compare our results from *sp*MTFL against *CL* and variants of *ELLA*. We use $(1 - AUC)$ score for our comparison. As in Figure 4.1, points above the line $y = x$ show that the *sp*MTFL does better than the other sequential learning methods. We can see that *sp*MTFL outperforms all the baselines on average (76.92). Compared to *sp*MTFL, *CL* performs better on easier tasks but worse on harder tasks. On the other hand, the performance of the variants of *ELLA* on harder tasks are comparable to that of our self-paced method, but worse on some easier tasks.

It is interesting to see that the lifelong learning algorithm with active task selection does worse than even ITL for some task selection strategy. It is because ELLA learns the tasks in an online fashion and does not update the previously learned tasks whenever $\mathbf{L}$ is updated. This is less of an issue if the tasks have considerable amount of data, but since we use limited data for our experiment, it hurts the performance of the lifelong learning algorithm significantly.

4.2 Co-Clustering for Multitask and Lifelong Learning

Early work on latent shared representation includes [117], which proposes a model based on Independent Component Analysis (ICA) for learning multiple related tasks. The task parameters are assumed to be generated from independent sources. [9] consider sparse representations common across many learning tasks. Similar in spirit to PCA for unsupervised tasks, their approach learns a low dimensional representation of the observations [38]. More recently, [63] assume that relationships among tasks are sparse to enforce that each observed task is obtained from only a few of the latent features, and from there learn the overlapping group structure among

[2] http://www.seas.upenn.edu/~eeaton/software/ELLAv1.0.zip

the tasks. [31] propose a K-means-like procedure that simultaneously clustering different tasks and learning a small pool of $m \ll T$ shared models. Specifically, each task is free to choose a model from the pool that better classifies its own data, and each model is learned from pooling together all the training data that belongs to the same cluster. [14] propose a similar approach that clusters the T tasks into K task-clusters with hard assignments.

These methods compute the factorization of the task weight matrix to learn the shared feature representation and the task structure. This matrix factorization induces the simultaneous clustering of both the tasks and the features in the K-dimensional latent subspace [70]. One of the major disadvantages of this assumption is that it restricts the model to define both the tasks and the features to have same number of clusters. For example, in the case of sentiment analysis, where each task belongs to a certain domain or a product category such as books, automobiles, etc., and each feature is simply a word from the vocabulary of the product reviews. Clearly, assuming both the features and the tasks have same number of clusters is an unjustified assumption, as the number of feature clusters are typically more than the number of task clusters, but the latter increase more than the former, as new products are introduced. Such a restrictive assumption may (and often does) hurt the performance of the model.

Unlike in the previous work, our proposed approach provides a flexible way to cluster both the tasks and the features. We introduce an additional degree of freedom that allows the number of task clusters to differ from the number of features clusters [39, 111]. In addition, our proposed models learn both the task relationship matrix and the feature relationship matrix along with the co-clustering of both the tasks and the features [47, 99]. Our proposed approach is closely related to Output Kernel Learning (*OKL*) where we learn the kernel between the components of the output vector for problems such as multi-output learning, multitask learning, etc [40, 100]. The key disadvantage of *OKL* is that it requires the computation of kernel matrix between every pair of instances from all the tasks. This results in scalability constraint especially when the number of tasks/features is large [112]. Our proposed models achieve the similar effect by learning a shared feature representation common across the tasks.

A key challenge in factoring with the extra degree of freedom is optimizing the resulting objective function. Previous work on co-clustering for multitask learning requires strong assumptions on the task parameters. [121] or not scalable to large-scale applications [113]. We propose an efficient algorithm that scales well to large-scale multitask learning and utilizes the structure of the objective function to learn the factorized task parameters. We formulate the learning of latent variables in terms of a *generalized Sylvester* equation which can be efficiently solved using the conjugate gradient descent algorithm. We start from the mathematical background and then motivate our approach in Section 4.2.1. Then we introduce our proposed models and their learning procedures in Section 4.2.2. Section 4.3.5 reports the empirical analysis of our proposed models and shows that learning both the task clusters and the feature clusters along with the task parameters gives significant improvements compared to the state-of-the-art baselines in multitask learning.

4.2.1 Preliminaries

Suppose we have T tasks and $\mathcal{D}_t = \{\mathbf{X}_t, Y_t\} = \{(x_{ti}, y_{ti}) : i = 1, 2, ..., N_t\}$ is the training set for each task $t = \{1, 2, \ldots, T\}$. Let W_t represent the weight vector for a task indexed by t. These

task weight vectors are stacked as columns of a matrix $\mathbf{W}$, which is of size $P \times T$, with P being the feature dimension. Traditional multitask learning imposes additional assumptions on $\mathbf{W}$ such as low-rank, ℓ_1 norm, $\ell_{2,1}$ norm, etc to leverage the shared characteristics among the tasks. In this paper, we consider a similar assumption based on the factorization of the task weight matrix $\mathbf{W}$.

In factored models, we decompose the weight matrix $\mathbf{W}$ as $F\mathbf{G}^\top$, where $\mathbf{F}$ can be interpreted as a feature cluster matrix of size $P \times K$ with K feature clusters and, similarly, $\mathbf{G}$ as a task cluster matrix of size $T \times K$ with K task clusters. If we consider squared error losses for all the tasks, then the objective function for learning $\mathbf{F}$ and $\mathbf{G}$ can be given as follows:

$$\underset{\substack{\mathbf{F}\in\mathcal{R}^{P\times K}\\ \mathbf{G}\in\mathcal{R}^{T\times K}\\ \mathbf{F}\in\Gamma_{\mathcal{F}},\mathbf{G}\in\Gamma_{\mathcal{G}}}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t\mathbf{F}G_t^\top\|_2^2 + \mathcal{P}_{\lambda_1}(\mathbf{F}) + \mathcal{P}_{\lambda_2}(\mathbf{G}) \tag{4.15}$$

In the above objective function, the latent feature representation is captured by the matrix $\mathbf{F}$ and the grouping structure on the tasks is determined by the matrix $\mathbf{G}$. The predictor W_t for task t can then be computed from $\mathbf{F}G_t^\top$, where G_t is t^{th} row of matrix $\mathbf{G}$. In the above objective function, $\mathcal{P}_{\lambda_1}(\mathbf{F})$ is a regularization term that penalizes the unknown matrix $\mathbf{F}$ with regularization parameter λ_1. Similarly, $\mathcal{P}_{\lambda_2}(\mathbf{G})$ is a regularization term that penalizes the unknown matrix $\mathbf{G}$ with regularization parameter λ_2. $\Gamma_{\mathcal{F}}$ and $\Gamma_{\mathcal{G}}$ are their corresponding constraint spaces. Without these additional constraints on $\mathbf{F}$ and $\mathbf{G}$, the objective function reduces to solving each task independently, since any task weight matrix from $\mathbf{F}$ and $\mathbf{G}$ can also be attained by $\mathbf{W}$.

Several assumptions can be enforced on these unknown factors $\mathbf{F}$ and $\mathbf{G}$. Below we discuss some of the previous models that make some well-known assumptions on $\mathbf{F}$ and $\mathbf{G}$ and can be written in terms of the above objective function.

(1) **Factored Multitask Learning** (*FMTL*) [6] considers a squared frobenius norm on both $\mathbf{F}$ and $\mathbf{G}$.

$$\underset{\substack{\mathbf{F}\in\mathcal{R}^{P\times K}\\ \mathbf{G}\in\mathcal{R}^{T\times K}}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t\mathbf{F}G_t^\top\|_2^2 + \lambda_1\|\mathbf{F}\|_F^2 + \lambda_2\|\mathbf{G}\|_F^2 \tag{4.16}$$

It can be shown that the above problem can equivalently written as the multitask learning with trace norm constraint on the task weight matrix $\mathbf{W}$.

(2) **Multitask Feature Learning** (*MTFL*) [9] assumes that the matrix $\mathbf{G}$ learns sparse representations common across many tasks. Similar in spirit to PCA for unsupervised tasks, *MTFL* learns a low dimensional representation of the observations $\mathbf{X}_t$ for each task, using $\mathbf{F}$ such that $\mathbf{F}\mathbf{F}^\top = \mathbf{I}_p$.

$$\underset{\substack{\mathbf{F}\in\mathcal{R}^{P\times K},\mathbf{G}\in\mathcal{R}^{T\times K}\\ \mathbf{F}\mathbf{F}^\top=\mathbf{I}_p}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t\mathbf{F}G_t^\top\|_2^2 + \lambda\|\mathbf{G}\|_{2,1}^2 \tag{4.17}$$

where K is usually set to P. It considers an $\ell_{2,1}$ norm on $\mathbf{G}$ to force all the tasks to have a similar sparsity pattern such that the tasks select the same latent features (columns of $\mathbf{F}$). It is worth noting that the Equation 4.17 can be equivalently written as follows:

$$\underset{\substack{\mathbf{W}\in\mathcal{R}^{P\times T},\\ \mathbf{\Sigma}\succeq 0}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t W_t\|_2^2 + \lambda tr(\mathbf{W}^\top\mathbf{\Sigma}^{-1}\mathbf{W}) \tag{4.18}$$

which then can be rewritten as multitask learning with a trace norm constraint on the task weight matrix $\mathbf{W}$ as before.

(3) **Group Overlap MTL** (*GO-MTL*) [63] assumes that the matrix $\mathbf{G}$ is sparse to enforce that each observed task is obtained from only a few of the latent features, indexed by the non-zero pattern of the corresponding rows of the matrix $\mathbf{G}$.

$$\underset{\substack{\mathbf{F}\in\mathcal{R}^{P\times K}\\ \mathbf{G}\in\mathcal{R}^{T\times K}}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t\mathbf{F}G_t^\top\|_2^2 + \lambda_1\|\mathbf{F}\|_F^2 + \lambda_2\|\mathbf{G}\|_1^1 \tag{4.19}$$

The above objective function can be compared to dictionary learning where each column of $\mathbf{F}$ is considered as a dictionary atom and each row of $\mathbf{G}$ is considered as their corresponding sparse codes [74].

(4) **Multitask Learning by Clustering** (*CMTL*) [14] assumes that the T tasks can be clustered into K task-clusters with hard assignment. For example, if the kth element of G_t is one, and all other elements of G_t are zero, we say that task t is associated with cluster k.

$$\underset{\substack{\mathbf{F}\in\mathcal{R}^{P\times K},\mathbf{G}\in\mathcal{R}^{T\times K}\\ G_t\in\{0,1\}^K\\ \|G_t\|_2=1,\forall t\in[T]}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t\mathbf{F}G_t^\top\|_2^2 + \lambda_1\|\mathbf{F}\|_F^2 \tag{4.20}$$

The constraints $G_t \in \{0,1\}^K, \|G_t\|_2 = 1$ ensure that $\mathbf{G}$ is a proper clustering matrix. Since the above problem is computationally expensive as it involves solving a combinatorial problem, the constraint on $\mathbf{G}$ is relaxed as $G_t \in [0,1]^K$.

These four methods require the number of task clusters to be same as the number of features clusters, which as mentioned earlier, is a restrictive assumption that may and often does hurt performance. In addition, these methods do not leverage the inherent relationship between the features (via $\mathbf{F}$) and the relationship between the tasks (via $\mathbf{G}$). Note that these objective functions are bi-convex problems where the optimization is convex in $\mathbf{F}$ when fixing $\mathbf{G}$ and vice versa. We cannot achieve globally optimal solution but one can show that algorithm reaches the locally optimal solution in a fixed number of iterations.

4.2.2 Proposed Methods: *BiFactor* MTL and *TriFactor* MTL

BiFactor MTL

Existing models do not take into consideration both the relationship between the tasks and the relationship between the features. Here we consider a more general formulation that in addition to estimating the parameters $\mathbf{F}$ and $\mathbf{G}$, we learn their task relationship matrix $\mathbf{\Omega}$ and the feature relationship matrix $\mathbf{\Sigma}$. We call this framework *BiFactor* multitask learning, following the factorization of the task parameters $\mathbf{W}$ into two low-rank matrices $\mathbf{F}$ and $\mathbf{G}$.

$$\underset{\substack{\mathbf{F}\in\mathcal{R}^{P\times K},\mathbf{G}\in\mathcal{R}^{T\times K}\\ \mathbf{\Sigma}\succeq 0,\mathbf{\Omega}\succeq 0}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t\mathbf{F}G_t\|_2^2 + \lambda_1 tr(\mathbf{F}^\top\mathbf{\Sigma}^{-1}\mathbf{F}) + \lambda_2 tr(\mathbf{G}^\top\mathbf{\Omega}^{-1}\mathbf{G}) \tag{4.21}$$

In the above objective function, we consider $\mathcal{P}_{\lambda_1}(\mathbf{F})$ and $\mathcal{P}_{\lambda_2}(\mathbf{G})$ to learn task relationship and feature relationship matrices $\mathbf{\Sigma}$ and $\mathbf{\Omega}$. The motivation for these regularization terms is based

on [9, 120] where they considered separately either the task relationship matrix $\mathbf{\Omega}$ or the feature relationship matrix $\mathbf{\Sigma}$. Note that the value of K is typically set to value less than $\min(P, T)$.

It is easy to see that by setting the value of $\mathbf{G}$ to I_T [3] , our objective function reduces to multitask feature learning ($MTFL$) discussed in the previous section. Similarly, by setting the value of $\mathbf{F}$ to I_P [4] , our objective function reduces to multitask relationship learning ($MTRL$) [120]. If we set $\mathbf{\Omega} = I_T$ and $\mathbf{\Sigma} = I_P$, we obtain the factored multitask learning setting ($FMTL$) defined in Equation 4.16. Hence the prior art can be cast as special cases of our more general formulation by imposing certain limiting restrictions.

Optimization for *BiFactor* MTL

We propose an efficient learning algorithm for solving the above objective function*BiFactor* MTL. Consider an alternating minimization algorithm, where we learn the shared representation $\mathbf{F}$ while fixing the task structure $\mathbf{G}$ and we learn the task structure $\mathbf{G}$ while fixing the shared representation $\mathbf{F}$. We repeat these steps until we converge to the locally optimal solution.

Optimizing w.r.t $\mathbf{F}$ gives an equation called generalized *Sylvester equation* of the form $AQB^\top + CQD^\top = E$ for the unknown Q. We will show in the next section on how to solve these linear equation efficiently. From the objective function, we have:

$$\sum_t (\mathbf{X}_t^\top \mathbf{X}_t)\mathbf{F}(G_t^\top G_t) + \lambda_1 \mathbf{\Sigma}^{-1} F = \sum_t \mathbf{X}_t^\top Y_t G_t \tag{4.22}$$

Optimizing w.r.t $\mathbf{G}$ for squared error loss results in the similar linear equation:

$$(\mathbf{F}^\top \mathbf{X}_t^\top \mathbf{X}_t \mathbf{F})G_t + \lambda_1 \mathbf{\Omega}^{-1}\mathbf{G} = \mathbf{F}^\top \mathbf{X}_t^\top Y_t \tag{4.23}$$

Optimizing w.r.t $\mathbf{\Omega}$ *and* $\mathbf{\Sigma}$: The optimization of the above function w.r.t $\mathbf{\Omega}$ and $\mathbf{\Sigma}$ while fixing the other unknowns can be learned easily with the following closed-form solutions [120]:

$$\mathbf{\Omega} = \frac{(\mathbf{G}\mathbf{G}^\top)^{\frac{1}{2}}}{tr((\mathbf{G}\mathbf{G}^\top)^{\frac{1}{2}})}$$

$$\mathbf{\Sigma} = \frac{(\mathbf{F}\mathbf{F}^\top)^{\frac{1}{2}}}{tr((\mathbf{F}\mathbf{F}^\top)^{\frac{1}{2}})}$$

TriFactor MTL

As mentioned earlier, one of the restrictions in *BiFactor* MTL and factored models is that both the number of feature clusters and task clusters should be set to K. This poses a serious model restriction, by assuming both the latent task and feature representation live in a same subspace. Such assumption can significantly hinder the flexibility of the model search space and we address this problem with a modification to our previous framework.

Following the previous work in matrix tri-factorization, we introduce an additional factor $\mathbf{S}$ such that we write $\mathbf{W}$ as $\mathbf{FSG}^\top$ where $\mathbf{F}$ is a feature cluster matrix of size $P \times K_1$ with K_1

[3] identity matrix of size $T \times T$ (assuming that the rank K is set to T)
[4] identity matrix of size $P \times P$ (assuming that the rank K is set to P)

feature clusters and $\mathbf{G}$ is a task cluster matrix of size $T \times K_2$ with K_2 task clusters and $\mathbf{S}$ is the matrix that maps feature clusters to task clusters. With this representation, latent features lie in a K_1 dimensional subspace and the latent tasks lie in a K_2 dimensional subspace.

$$\underset{\substack{\mathbf{F}\in\mathcal{R}^{P\times K_1},\\ \mathbf{G}\in\mathcal{R}^{T\times K_2},\\ \mathbf{S}\in\mathcal{R}^{K_1\times K_2}\\ \mathbf{\Sigma}\succeq 0,\mathbf{\Omega}\succeq 0}}{\arg\min} \sum_{t\in[T]} \|Y_t - \mathbf{X}_t\mathbf{F}\mathbf{S}G_t^\top\|_2^2 + \lambda_1 tr(\mathbf{F}^\top\mathbf{\Sigma}^{-1}\mathbf{F}) + \lambda_2 tr(\mathbf{G}^\top\mathbf{\Omega}^{-1}\mathbf{G}) \tag{4.24}$$

The cluster mapping matrix $\mathbf{S}$ introduces an additional degree of freedom in the factored models and addresses the realistic assumptions encountered in many applications. Note that we do not consider any regularization on $\mathbf{S}$ in this paper, but one may impose additional constraint on $\mathbf{S}$ such as ℓ_1 (sparse penalty), ℓ_2^2 (ridge penalty), non-negative constraints, etc, to further improve performance.

Optimization for *TriFactor* MTL

We introduce an efficient learning algorithm for solving *TriFactor* MTL, similar to the optimization procedure for *BiFactor* MTL. As before, we consider an alternating minimization algorithm, where we learn the shared representation $\mathbf{F}$ while fixing the $\mathbf{G}$ and $\mathbf{S}$, we learn the task structure $\mathbf{G}$ while fixing the $\mathbf{F}$ and $\mathbf{S}$ and we learn the cluster mapping matrix $\mathbf{S}$, by fixing $\mathbf{F}$ and $\mathbf{G}$. We repeat these steps until we converge to a locally optimal solution.

Optimizing w.r.t $\mathbf{F}$ gives a generalized *Sylvester equation* as before.

$$\sum_t (\mathbf{X}_t^\top\mathbf{X}_t)\mathbf{F}(\mathbf{S}G_t^\top G_t\mathbf{S}^\top) + \lambda_1\mathbf{\Sigma}^{-1}\mathbf{F} = \sum_t \mathbf{X}_t^\top Y_t G_t\mathbf{S}^\top \tag{4.25}$$

Optimizing w.r.t $\mathbf{G}$ gives the following linear equation:

$$(\mathbf{S}^\top\mathbf{F}^\top\mathbf{X}_t^\top\mathbf{X}_t\mathbf{F}\mathbf{S})G_t + \lambda_1\mathbf{\Omega}^{-1}\mathbf{G} = \mathbf{S}^\top\mathbf{F}^\top\mathbf{X}_t^\top Y_t \tag{4.26}$$

for all $t \in [T]$.

Optimizing w.r.t $\mathbf{S}$: Solving for $\mathbf{S}$ results in the following equation:

$$\sum_t (\mathbf{F}^\top\mathbf{X}_t^\top\mathbf{X}_t\mathbf{F})\mathbf{S}(G_t^\top G_t) = \sum_t \mathbf{F}^\top\mathbf{X}_t^\top Y_t G_t \tag{4.27}$$

Optimizing w.r.t $\mathbf{\Omega}$ *and* $\mathbf{\Sigma}$: The optimization of the above function w.r.t $\mathbf{\Omega}$ and $\mathbf{\Sigma}$ while fixing the other unknowns can be learned as in *BiFactor*MTL. Note that one may consider ℓ_1 regularization on $\mathbf{\Omega}$ and $\mathbf{\Sigma}$ to learn the sparse relationship between the tasks and the features [119].

Solving the Generalized Sylvester Equations

We give some details on how to solve the generalized *Sylvester equations* (4.22,4.23,4.25,4.26,4.27) encountered in *BiFactor* and *TriFactor* MTL optimization steps. The generalized *Sylvester equation* of the form $AQB^\top + CQD^\top = E$ has a unique solution Q under certain regularity conditions which can be exactly obtained by an extended version of the classical Bartels-Stewart method

whose complexity is $\mathcal{O}((p+q)^3)$ for $p \times q$-matrix variable Q, compared to the naive matrix inversion which requires $\mathcal{O}(p^3q^3)$.

Alternatively one can solve the linear equation using the properties of the Kronecker product: $(B^\top \otimes A)vec(Q) + (D^\top \otimes C)vec(Q) = vec(E)$ where $\otimes$ is the Kronecker product and $vec(.)$ vectorizes Q in a column oriented way. Below, we show the alternative form for *TriFactor* MTL equations:

$$\sum_t ((\mathbf{S}G_t^\top G_t\mathbf{S}^\top) \otimes (\mathbf{X}_t^\top \mathbf{X}_t))vec(\mathbf{F}) + \lambda_1(\mathbf{I}_{K_1} \otimes \mathbf{\Sigma}^{-1})vec(\mathbf{F}) = vec(\sum_t \mathbf{X}_t^\top Y_t G_t \mathbf{S}^\top) \tag{4.28}$$

$$diag(\mathbf{S}^\top\mathbf{F}^\top\mathbf{X}_t^\top\mathbf{X}_t\mathbf{F}\mathbf{S})_{t=1}^T vec(\mathbf{G}) + (\mathbf{I}_{K_2} \otimes \mathbf{\Omega}^{-1})vec(\mathbf{G}) = vec([\mathbf{S}^\top\mathbf{F}^\top\mathbf{X}_t^\top Y_t]_{t=1}^T) \tag{4.29}$$

$$\sum_t ((G_t^\top G_t) \otimes (\mathbf{F}^\top\mathbf{X}_t^\top\mathbf{X}_t\mathbf{F}))vec(S) = vec(\sum_t \mathbf{F}^\top\mathbf{X}_t^\top Y_t) \tag{4.30}$$

We can do the same for *BiFactor* MTL, enabling us to use conjugate gradient descent (CG) to learn our unknown factors whose complexity depends on the condition number of the matrix $(B^\top \otimes A) + (D^\top \otimes C)$. To optimize $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{S}$, we iteratively run conjugate gradient descent for each factor while fixing the other unknowns until a convergence condition (tolerance $\leq 10^{-6}$) is met. In addition, CG can exploit the solution from the previous iteration, low rank structure in the equation and the fact that the matrix vector products can be computed relatively efficiently. From our experiments. We find that our algorithm converges fast, i.e. in a few iterations.

4.2.3 Experiments

In this section, we report on experiments on both synthetic datasets and three real world datasets to evaluate the effectiveness of our proposed MTL methods. We compare both our models with several state-of-the-art baselines discussed in Section 4.2.1. We include the results for Shared Multitask learning (*SHAMO*) [31], which uses a K-means like procedure that simultaneously clusters different tasks using a small pool of $m \ll T$ shared model. Following [14], we use gradient-projection algorithm to optimize the dual of the objective function (Equation 4.20). In addition, we compare our results with Single-task learning (*STL*), which learns a single model by pooling together the data from all the tasks and Independent task learning (*ITL*) which learns each task independently.

1. **Single-task learning** (*STL*) learns a single model by pooling together the data from all the tasks.
2. **Independent task learning** (*ITL*) learns each task independently. In this case, there will effectively be no transfer of information between the tasks.
3. **Shared Multitask learning** (*SHAMO*) [31], uses a K-means like procedure that simultaneously clusters different tasks using a small pool of $m \ll T$ shared model.
4. **Multitask Feature learning** (*MTFL*) learns the feature relationship matrix $\mathbf{\Sigma}$ along with the task parameters $\mathbf{W}$ alternatively[5]., as discussed in Section 4.2.1.

[5]The source code for this baseline is available at http://ttic.uchicago.edu/~argyriou/code/mtl_feat/mtl_feat.tar

5. **Multitask learning by Clustering** (*CMTL*) clusters T tasks into K task-clusters. We use gradient-projection algorithm as suggested in [14] to optimize the dual of the objective function (Equation 4.20). We show the results for *CMTL* as a baseline for sentiment analysis since their algorithm is applicable only to classification tasks.
6. **Group Overlap Multitask Learning** (*GO-MTL*) considers a dictionary learning for task parameters, as discussed in Section 4.2.1. .

The parameters for the proposed formulations and several state-of-the-art baselines are chosen from 3-fold cross validation. Following [63], we fix the value of λ_1 to 0.1 in order to reduce the search space. The value for λ_2 is chosen from the search grid $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$. The value for K, K_1 and K_2 are chosen from $\{2, 3, 5, 7, 9, 10, 15\}$. We evaluate the models using Root Mean Squared Error (*RMSE*) for the regression tasks and using F-measure for the classification tasks. For our experiments, we consider the squared error loss for each task. We repeat all our experiments 5 times to compensate for statistical variability. The best model and the statistically competitive models (by paired *t-test* with $\alpha = 0.05$) are shown in boldface.

Synthetic Data

We evaluate our models on five synthetic datasets based on the assumptions considered in both the baselines and the proposed methods. We generate 100 examples from $\mathbf{X}_t \sim \mathcal{N}(0, \mathbf{I}_P)$ with $P = 20$ for each task t. All the datasets consist of 30 tasks with 25 training examples per task. Each task is constructed using $Y_t = \mathbf{X}_t W_t + \mathcal{N}(0, 1)$. The task parameters for each synthetic dataset is generated as follows:

1. ***syn1*** dataset consists of 3 groups of tasks with 10 tasks in each group without any overlap. We generate $K = 15$ latent features from $\mathbf{F} \sim \mathcal{N}(0, 1)$ and each W_t is constructed from linearly combining 5 latent features from $\mathbf{F}$.
2. ***syn2*** dataset is generated with 3 overlapping groups of tasks. As before, we generate $K = 15$ latent features from $\mathbf{F} \sim \mathcal{N}(0, 1)$ but tasks in group 1 are constructed from features $1 - 7$, tasks in group 2 are constructed from features $4 - 12$ and the tasks in group 3 are constructed from features $9 - 15$.
3. ***syn3*** dataset simulates the *BiFactor* MTL. We randomly generate task covariance matrix $\mathbf{\Omega}$ and feature covariance matrix $\mathbf{\Sigma}$. We sample $\mathbf{F} \sim \mathcal{N}(0, \mathbf{\Sigma})$ and $\mathbf{G} \sim \mathcal{N}(0, \mathbf{\Omega})$ and compute $\mathbf{W} = \mathbf{F}\mathbf{G}^\top$.
4. ***syn4*** dataset simulates the *TriFactor* MTL. We randomly generate task covariance matrix $\mathbf{\Omega}$ and feature covariance matrix $\mathbf{\Sigma}$. We sample $\mathbf{F} \sim \mathcal{N}(0, \mathbf{\Sigma})$, $\mathbf{G} \sim \mathcal{N}(0, \mathbf{\Omega})$ and $\mathbf{S} \sim \mathcal{U}(0, 1)$. We compute the task weight matrix by $\mathbf{W} = \mathbf{F}\mathbf{S}\mathbf{G}^\top$.
5. ***syn5*** dataset simulates the experiment with task weight matrix drawn from a matrix normal distribution [119]. We randomly generate task covariance matrix $\mathbf{\Omega}$ and feature covariance matrix $\mathbf{\Sigma}$. We sample $vec(\mathbf{W}) \sim \mathcal{N}(0, \mathbf{\Sigma} \otimes \mathbf{\Omega})$.

We compare the proposed methods *BiFactor* MTL and *TriFactor* MTL against the baselines. We can see in Table 4.2 that *BiFactor* and *TriFactor* MTL outperforms all the baselines in all the synthetic datasets. *STL* performs the worst since it combines the data from all the tasks. We can

Table 4.2: Performance results (RMSE) on synthetic datasets. The table reports the mean and standard errors over 5 random runs. The best model and the statistically competitive models (by paired *t-test* with $\alpha = 0.05$) are shown in boldface.

Model	**syn1**	**syn2**	**syn3**	**syn4**	**syn5**
STL	4.79 (0.04)	5.71 (0.05)	5.5 (0.04)	4.02 (0.02)	5.72 (0.06)
ITL	1.98 (0.08)	2.10 (0.09)	2.01 (0.06)	1.95 (0.06)	2.14 (0.07)
SHAMO	3.63 (0.22)	4.37 (0.27)	3.56 (0.23)	2.76 (0.13)	4.27 (0.31)
MTFL	1.91 (0.08)	1.95 (0.07)	1.64 (0.06)	1.47 (0.05)	1.91 (0.07)
GO-MTL	1.84 (0.10)	1.90 (0.08)	1.72 (0.04)	1.63 (0.06)	1.84 (0.06)
BiFactorMTL	1.85 (0.08)	**1.85 (0.06)**	1.68 (0.08)	1.37 (0.08)	1.74 (0.07)
TriFactorMTL	**1.78 (0.08)**	**1.83 (0.07)**	**1.46 (0.05)**	**1.31 (0.02)**	**1.68 (0.10)**

see that the *SHAMO* performs better than *STL* but worse than *ITL* which shows that learning these tasks separately is beneficial than combining them to learn a fewer models.

As mentioned earlier, since *MTFL* is similar to *FMTL* in Equation 4.16, we can see how the results of *BiFactor* MTL improve when it learns both the task relationship matrix and the feature relationship matrix. Note that the ***syn1*** and ***syn2*** datasets are based on assumptions in *GO-MTL*, hence, it performs better than the other baselines. *BiFactor* MTL and *TriFactor* MTL models are equally competent with *GO-MTL* which shows that our proposed methods can easily adapt to these assumptions. Synthetic datasets ***syn3***, ***syn4*** and ***syn5*** are generated with both the task covariance matrix and the feature covariance matrix. Since both *BiFactor* MTL and *TriFactor* MTL learns task and feature relationship matrix along with the task weight parameters, they performs significantly better than other baselines.

Exam Score Prediction

We evaluate the proposed methods on examination score prediction data, a benchmark dataset in multitask regression reported in several previous articles [9, 63, 120] [6]. The *school* dataset consists of examination scores of $15,362$ students from 139 schools in London. Each school is considered as a task and we need to predict exam scores for students from these 139 schools. The feature set includes the year of the examination, four school-specific and three student-specific attributes. We replace each categorical attribute with one binary variable for each possible attribute value, as suggested in [9]. This results in 26 attributes with an additional attribute to account for the bias term.

Clearly, the dataset has the school and student specific feature clusters that can help in learning

[6] http://ttic.uchicago.edu/~argyriou/code/mtl_feat/school_splits.tar

Table 4.3: Performance results (RMSE) on school datasets. The table reports the mean and standard errors over 5 random runs.

Models	*STL*	*ITL*	*SHAMO*	*MTFL*	*GO-MTL*	*BiFactor*	*TriFactor*
20%	12.19 (0.03)	12.00 (0.04)	11.91 (0.05)	11.25 (0.05)	11.15 (0.05)	10.68 (0.08)	**10.54 (0.09)**
30%	12.09 (0.07)	12.01 (0.05)	10.92 (0.05)	10.85 (0.02)	10.53 (0.10)	10.38 (0.11)	**10.22 (0.08)**
40%	12.00 (0.10)	11.88 (0.06)	11.82 (0.06)	10.61 (0.06)	10.31 (0.14)	10.20 (0.13)	**10.12 (0.10)**

the shared feature representation better than the other factored baselines. In addition, there must be several task clusters in the data to account for the differences among the schools. The training and test sets are obtained by dividing examples of each task into many small datasets, by varying the size of the training data with 20%, 30% and 40%, in order to evaluate the proposed methods on many tasks with limited numbers of examples.

Table 4.3 shows the experimental results for *school* data. All the factorized MTL methods outperform *STL* and *ITL*. We can see that both *TriFactor* MTL and *BiFactor* MTL outperform other baselines significantly. It is interesting to see that *TriFactor* MTL performs considerably well even when the tasks have limited numbers of examples. When there is more training data, the result the advantage of *TriFactor* MTL over the strongest baseline *GO-MTL* is reduced.

Sentiment Analysis

We follow the experimental setup in [14, 31] and evaluate our algorithm on product reviews from amazon[7]. The dataset contains product reviews from 14 domains such as books, dvd, etc. We consider each domain as a binary classification task. The reviews are stemmed and stopwords are removed from the review text. We represent each review as a bag of $5,000$ unigrams/bigrams with *TF-IDF* scores. We choose $1,000$ reviews from each domain and each review is associated with a rating from $\{1, 2, 4, 5\}$. The reviews with rating 3 is not included in this experiment as such sentiments were ambiguous and therefore cannot be reliably predicted. h

We ran several experiments on this dataset to test the importance of learning shared feature representation and co-clustering of tasks and features. In Experiment **I**, we construct 14 classification tasks with reviews labeled positive $(+)$ when rating < 3 and labeled negative $(-)$ when rating < 3. We use 240 training examples for each task and the remaining for test set. Since all the tasks are essentially same, *ITL* perform better than all the other models (with an F-measure of 0.749) by combining data from all the other tasks. The results for our proposed methods *BiFactor* MTL (0.722) and *TriFactor* MTL (0.733) are comparable to that of *ITL*. See supplementary material for the results of Experiment **I**.

For Experiment **II**, we split each domain into *two* equal sets, from which we create two prediction tasks based on the two different thresholds: whether the rating for the reviews is 5 or not and whether the rating for the reviews is 1 or not. Obviously, combining all the tasks together

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

Table 4.4: Performance results (F-measure) for various experiments on sentiment detection. The table reports the mean and standard errors over 5 random runs.

Data	**II**	**III**	**IV**	**V**	**VI**	**VII**
Tasks	28	56	84	42	86	126
Thresholds (Splits)	2 (2)	2 (4)	2 (6)	3 (3)	3 (6)	3 (9)
Train Size	120	60	40	80	40	26
STL	0.429 (0.002)	0.432 (0.001)	0.429 (0.002)	0.400 (0.002)	0.399 (0.003)	0.397 (0.001)
ITL	0.433 (0.001)	0.440 (0.002)	0.431 (0.001)	0.499 (0.001)	0.486 (0.002)	0.479 (0.001)
SHAMO	0.423 (0.002)	0.437 (0.006)	0.429 (0.002)	0.498 (0.006)	0.460 (0.002)	0.496 (0.013)
CMTL	0.557 (0.016)	0.436 (0.007)	0.429 (0.004)	0.508 (0.002)	0.486 (0.002)	0.476 (0.002)
MTFL	0.482 (0.004)	0.473 (0.002)	0.432 (0.007)	0.522 (0.002)	0.487 (0.003)	0.481 (0.002)
GO-MTL	0.582 (0.012)	0.526 (0.013)	0.516 (0.007)	0.587 (0.004)	0.540 (0.005)	0.539 (0.008)
BiFactor	0.611 (0.018)	0.561 (0.013)	**0.598 (0.002)**	**0.643 (0.013)**	0.578 (0.020)	0.574 (0.052)
TriFactor	**0.627 (0.008)**	**0.588 (0.006)**	**0.603 (0.012)**	**0.655 (0.013)**	**0.606 (0.020)**	**0.632 (0.029)**

will not help in this setting. Experiments **III** and **IV** are similar to Experiment **II**, except that each task is further divided into 2 or 3 sub-tasks.

Experiment **V** splits each domain into *three* equal sets to construct three prediction tasks based on three different thresholds: whether the rating for the reviews is 5 or not, whether the rating for the reviews is < 3 or not and whether the rating for the reviews is 1 or not. This setting captures the reviews with different levels of sentiments. As before, we build the dataset for Experiments **VI** and **VII** by further dividing the three prediction tasks from Experiment **V** into 2 or 3 sub-tasks.

The results from our experiments are reported in Table 4.4. The first four rows in the table show the number of tasks in each experiment, number of thresholds considered for the ratings, number of splits constructed from each domain and the total number of training examples in each task. The general trend is that factorized models performs significantly better than the other baselines. Since *MTFL*, *BiFactor*MTL and *TriFactor*MTL learn feature relationship matrix $\mathbf{\Sigma}$ in addition to the task parameter, they achieve better results than *CMTL*, which considers only the task clusters.

We notice that as we increase the number of tasks, the gap between the performances of *TriFactor*MTL and *BiFactor*MTL (and *GO-MTL*) widens, since the assumption that the the

Table 4.5: Performance results (F-measure) on 20*Newsgroups* dataset. The table reports the mean and standard errors over 5 random runs.

Models	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**
GO-MTL	0.42 (0.09)	0.57 (0.06)	0.42 (0.04)	0.47 (0.06)	0.40 (0.03)
BiFactorMTL	0.42 (0.09)	0.60 (0.05)	0.41 (0.04)	0.49 (0.03)	0.36 (0.01)
TriFactorMTL	**0.49 (0.03)**	**0.63 (0.02)**	**0.54 (0.02)**	**0.54 (0.02)**	**0.51 (0.02)**

number of feature and task clusters K should be same is clearly violated. On the other hand, *TriFactor*MTL learns with a different number of feature and task clusters (K_1, K_2) and, hence achieves a better performance than all the other methods considered in these experiments.

Learning to Transfer to New Tasks

Finally, we evaluate our proposed models on 20*Newsgroups* dataset for transfer learning [8]. The dataset contains postings from 20 Usenet newsgroups. As before, the postings are stemmed and the stopwords are removed from the text. We represent each posting as a bag of 500 unigrams/bigrams with *TF-IDF* scores. We construct 10 tasks from the postings of the newsgroups. We randomly select a pair of newsgroup classes to build each one-vs-one classification task. We follow the hold-out experiment suggested by [87] for the transfer learning setup. For each of the 10 tasks (target task), we learn $\mathbf{F}$ ($\mathbf{F}$ and $\mathbf{S}$ in case of *TriFactor*MTL) from the remaining 9 tasks (source tasks).

With $\mathbf{F}$ ($\mathbf{F}$ and $\mathbf{S}$) known from the source tasks, we select 10% of the data from the target task to learn G_{target}. This experiment shows how well the learned latent feature representation from the source tasks in a K-dimensional subspace (K_1-dimensional subspace for *TriFactor*MTL) adapt to the new task. We evaluate our results on the remaining data from the target task. We select *GO-MTL* as our baseline to compare our results. Since *CMTL* doesn't explicitly learn $\mathbf{F}$, we did not include it in this experiment.

Table 4.5 shows the results for this experiment. We report the first 5 tasks here. See supplementary material for the performance results of all the 10 tasks. We see that both *GO-MTL* and *BiFactor*MTL perform almost the same, since both of them learn the latent feature representation in a K-dimensional space. As is evident from the table, *TriFactor*MTL outperforms both *GO-MTL* and *BiFactor*MTL, which shows that learning both the factors $\mathbf{F}$ and $\mathbf{S}$ improves information transfer from the source tasks to the target task.

4.3 Lifelong Multitask Learning with Output Kernels

Lifelong learning poses considerable challenges in terms of effectiveness (minimizing prediction errors for all tasks) and overall computational tractability for real-time performance. A lifelong

[8] http://qwone.com/~jason/20Newsgroups/

learning agent must provide an efficient way to learn new tasks faster by utilizing the knowledge learned from the previous tasks and also not forgetting or significantly degrading performance on the old tasks. The goal of a lifelong learner is to minimize errors as compared to the full ideal hindsight learner, which has access to all the training data and no bounds on memory or computation.

This section addresses lifelong multitask learning by jointly re-estimating the inter-task relations from the data and the per-task model parameters at each round. Following the online setting, particularly from [25, 96], at each round t, the learner *receives* an example from a task, along with the task identifier and *predicts* the output label for the example. Subsequently, the learner receives the true label and *updates* the model(s) as necessary. This process is repeated as we receive additional data from the same or different tasks. Our approach follows an error-driven update rule in which the model for a given task is updated only when the prediction for that task is in error.

In this work, we define the task relationship matrix as *output* kernels in *Reproducing Kernel Hilbert Space* (RKHS) on multitask examples. We propose a novel algorithm called *Online Output Kernel Learning Algorithm* (OOKLA) for lifelong learning setting. For a successful lifelong learning with kernels, we need to address two key challenges: (1) learn the relationships between the tasks (output kernel) efficiently from the data stream and (2) bound the size of the knowledge to avoid memory explosion. The key challenge in learning with a large number of tasks is to adaptively learn the model parameters and the task relationships, which potentially change over time. Without manageability-efficient updates at each round, learning the task relationship matrix automatically may impose a severe computational burden. In other words, we need to make predictions and update the models in an efficient real time manner.

We propose simple and quite intuitive update rules for learning the task relationship matrix. When we receive a new example, the algorithm updates the output kernel when the learner made a mistake by computing the similarity between the new example and the set of representative examples (stored in the memory) that belongs to a specific task. If the two examples have similar (different) labels and high similarity, then the relationship between the tasks is increased (decreased) to reflect the positive (negative) correlation and vice versa.

To avoid the memory explosion associated with the lifelong learning setting, we propose a robust budget-limited version of the proposed algorithm that efficiently utilizes the relationship between the tasks to bound the total number of representative examples in the support set. In addition, we propose a two-stage budgeted scheme for efficiently tackling the task-specific budget constraints in lifelong learning.

It is worth noting that the problem of lifelong multitask learning is closely related to online multitask learning. Although the objectives of both online multitask learning and lifelong learning are similar, one key difference is that the online multitask learning, unlike in the lifelong learning, may require that the number of tasks be specified beforehand. In recent years, online multitask learning has attracted extensive research attention [1, 25, 35, 73, 79, 96]. We evaluate our proposed methods with several state-of-the-art online learning algorithms for multiple tasks.

There are many useful application areas for lifelong learning, including optimizing financial trading as market conditions evolve, email prioritization with new tasks or preferences emerging, personalized news, and spam filtering, with evolving nature of spam. Consider the latter, where some spam is universal to all users (e.g. financial scams), some messages might be useful to

certain affinity groups, but spam to most others (e.g. announcements of meditation classes or other special interest activities), and some may depend on evolving user interests. In spam filtering each user is a "task," and shared interests and dis-interests formulate the inter-task relationship matrix. If we can learn the matrix as well as improving models from specific spam/not-spam decisions, we can perform mass customization of spam filtering, borrowing from spam/not-spam feedback from users with similar preferences. Our primary contribution is precisely the joint learning of inter-task relationships and its use in estimating per-task model parameters in a lifelong learning setting.

4.3.1 Problem Setup

Let $((x_t, i_t), y_t)$ be the example received by the learner from the task i_t (at the time step t) where we assume that the $\boldsymbol{x}_t \in \mathcal{X}$ and y_t is its corresponding true label. The task i_t can be a new task or one seen by the learner in the previous iterations. We denote by $[N]$ the consecutive integers ranging from 1 to N. Throughout this section, we do not assume that the number of tasks is known to the learner ahead of time, an important constraint in lifelong learning problems. Let K be the number of tasks seen so far until the current iteration t.

For brevity, we consider a binary classification problem for each task $y_t \in \{-1, +1\}$, but the methods generalize to multi-class cases and are also applicable to regression tasks. We assume that the learner made a mistake if $y_t \neq \hat{y}_t$ where $\hat{y}_t$ is the predicted label. Our approach follows a mistake-driven update rule in which the model for a given task is updated only on rounds where the learner predictions differ from the true label.

Let $\mathcal{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ (kernel on *input* space) and $\Omega : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ (*output* kernel) be symmetric, positive semi-definite (p.s.d) multitask kernel functions and denote $\mathcal{H}$ as their corresponding RKHS of functions with the norm $\| \cdot \|_{\mathcal{H}}$ on multitask examples ([40, 54, 102]). Using the above notation, we can define a kernel representation of an example based on a set of representative examples collected on the previous iterations (*prototypes*). Formally, given an example $\boldsymbol{x} \in \mathcal{X}$, its kernel representation can be written using this set:

$$\boldsymbol{x} \longmapsto \{\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_s) : s \in \mathcal{S}\}$$

$\mathcal{S}$ is the set of stored examples for which the learner made a mistake in the past. The set $\mathcal{S}$ is called the *support set*. The online classification function is then defined as the weighted sum of the kernel combination of the examples in the support set. To account for the examples from the different tasks, we consider both the kernel on the input space $\mathcal{K}$ and the output kernel Ω in our classification function.

$$\boldsymbol{f}(\cdot) = \sum_{(\boldsymbol{x}_s, i_s) \in \mathcal{S}} \alpha_s \Omega_{i_s} . \phi(\boldsymbol{x}_s) \tag{4.31}$$

We set $\alpha_s = y_s$. The predicted label for a new example is computed from the linear combination of the labels of the examples from the support set $\mathcal{S}$ weighted by their input similarity $\mathcal{K}$ and the task similarity Ω to the new example. Using the *kernel trick*, one can write:

$$\boldsymbol{f}((x_t, i_t)) = \langle \phi(\boldsymbol{x}_t) \sum_{(\boldsymbol{x}_s, i_s) \in \mathcal{S}} \alpha_s \Omega_{i_s i_t} \phi(\boldsymbol{x}_s) \rangle = \sum_{(\boldsymbol{x}_s, i_s) \in \mathcal{S}} \alpha_s \Omega_{i_s i_t} \mathcal{K}(\boldsymbol{x}_s, \boldsymbol{x}_t)$$

Algorithm 8: Online Output Kernel Learning Algorithm

Initialize: $\mathcal{S} = \emptyset$, $\boldsymbol{\Omega}$
for $t = 1, 2, \cdots$ **do**
  Receive new example $(\boldsymbol{x}_t, i_t)$
  Let $\boldsymbol{f}_{i_t} = \sum_{(\boldsymbol{x}_s, i_s) \in \mathcal{S}} y_s \Omega_{i_s i_t} \phi(\boldsymbol{x}_s)$
  Predict $\hat{y}_t = sign(\boldsymbol{f}_{i_t}(\boldsymbol{x}_t))$
  Receive label y_t
  if $y_t \neq \hat{y}_t$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup (\boldsymbol{x}_t, i_t)$
    Update $\Omega_{i_t k} \forall k \in [K]$

$$\Omega_{i_t k} \leftarrow \Omega_{i_t k} + \frac{1}{\lambda} \sum_{\substack{(\boldsymbol{x}_s, i_s) \in \mathcal{S} \\ i_s = k}} y_t \mathcal{K}(\boldsymbol{x}_t, \boldsymbol{x}_s) y_s$$

(or)

$$\Omega_{i_t k} \leftarrow \Omega_{i_t k} \exp\left\{ \frac{1}{\lambda} \sum_{\substack{(\boldsymbol{x}_s, i_s) \in \mathcal{S} \\ i_s = k}} y_t \mathcal{K}(\boldsymbol{x}_t, \boldsymbol{x}_s) y_s \right\}$$

  end
end

4.3.2 Online Output Kernel Learning

Note that, in the above representation, we need to learn both the support set $\mathcal{S}$ and the output kernel Ω from the data. As explained in the previous section, for a successful lifelong learning with kernels, we need to address two key challenges: (1) learn the relationships between the tasks (output kernel) efficiently from the data arriving in an online fashion and (2) bound the size of the *support* set $\mathcal{S}$ to avoid memory explosion. We address these two challenges in the following sections.

Our objective function for the lifelong learning problem is given as follows:

$$\min_{\Omega \in \boldsymbol{S}_+, \boldsymbol{f} \in \mathcal{H}} \sum_t \ell(y_t, \boldsymbol{f}((x_t, i_t))) + \frac{1}{2} \|\boldsymbol{f}\|_{\mathcal{H}}^2 + \lambda \mathcal{R}(\Omega) \tag{4.32}$$

where $\ell(\cdot)$ is some loss function such as hinge loss or logistic loss, $\mathcal{R}(\cdot)$ is the regularization on the task relationship matrix Ω and λ is the regularization parameter. Note that f in the above equation depends on Ω. In order to reduce the time taken for each time-step, we require an efficient update to the task relationship matrix Ω. Following the work of [54] in the batch setting, we consider a subset of regularization functions $\mathcal{R}$ for which we can efficiently learn the task covariance matrix. Consider the dual function of the above equation, at time-step t (see [14, 54]):

$$\max_{\Omega \in \boldsymbol{S}_+} \frac{1}{2} \sum_k \sum_{\substack{(\boldsymbol{x}_s, i_s) \in \mathcal{S} \\ i_s = k}} \alpha_t \mathcal{K}(\boldsymbol{x}_t, \boldsymbol{x}_s) \alpha_s \Omega_{i_t k} - \lambda \mathcal{R}(\Omega, \Omega_{(t-1)}) \tag{4.33}$$

When we consider the entry-wise l_p norm between Ω and $\Omega_{(t-1)}$ from the previous iteration as our regularization i.e., $\mathcal{R}(\Omega, \Omega_{(t-1)}) = \sum_k |\Omega_{i_t k} - (\Omega_{(t-1)})_{i_t k}|^p$ for $p \geq 1$ [9], we get the following update function:

$$\Omega_{i_t k} \leftarrow \Omega_{i_t k} + \frac{1}{\lambda} \sum_{\substack{(\boldsymbol{x}_s, i_s) \in \mathcal{S} \\ i_s = k}} y_t \mathcal{K}(\boldsymbol{x}_t, \boldsymbol{x}_s) y_s \tag{4.34}$$

Similarly, if we consider the generalized KL-divergence between Ω and $\Omega_{(t-1)}$ i.e., $\mathcal{R}(\Omega, \Omega_{(t-1)}) = \sum_k \Omega_{i_t k} \log \frac{\Omega_{i_t k}}{(\Omega_{(t-1)})_{i_t k}} - \Omega_{i_t k} + (\Omega_{(t-1)})_{i_t k}$, we get the following update function:

$$\Omega_{i_t k} \leftarrow \Omega_{i_t k} \exp \left\{ \frac{1}{\lambda} \sum_{\substack{(\boldsymbol{x}_s, i_s) \in \mathcal{S} \\ i_s = k}} y_t \mathcal{K}(\boldsymbol{x}_t, \boldsymbol{x}_s) y_s \right\} \tag{4.35}$$

Unlike in the previous work, we update only the row (and the corresponding column) of the task relationship matrix Ω specific to the task i_t, which significantly reduces the time taken per example.

We can see that the update equations are simple and quite intuitive. For a given new example $(\boldsymbol{x}_t, i_t)$ at round t, the algorithm updates $\Omega_{i_t k}$ (for some $k \in [K]$) by computing the similarity between the new example and the examples in the support set $\mathcal{S}$ that belongs to the task k. If the two examples have similar (different) labels and high similarity $\mathcal{K}(\boldsymbol{x}_t, \boldsymbol{x}_s)$, then the $\Omega_{i_t k}$ is increased to reflect the positive (negative) correlation and vice versa. A value close to 0 implies no significant relationship between the tasks. The update to the $\Omega_{i_t k}$ is normalized by the regularization parameter λ for scaling.

It is worth noting that our update equations do not violate the *p.s.d* constraints on Ω in Equation 4.33. If Ω from the previous iteration is a *p.s.d* matrix and the update is a *p.s.d* matrix (as it is computed using the Gram matrix of the example from the previous iteration), the sum and Hadamard product of two *p.s.d* matrices satisfy the *p.s.d* constraint (using the Schur Product Theorem).

Algorithm 8 outlines the key steps in our proposed method. We write $f((x_t, i_t))$ as $f_{i_t}(x_t)$ for notational convenience. At each time-step t, the learner *receives* an example $\boldsymbol{x}_t \in \mathcal{X}$ and *predicts* the output label y_t using $\hat{y}_t = sign(\boldsymbol{f}_{i_t}(\boldsymbol{x}_t))$. We update both the support set $\mathcal{S}$ and the output kernel $\Omega_{i_t \cdot}$ when the learner makes a mistake.

4.3.3 Learning with a Budget

In Algorithm 8, we can see that both the classification function $\boldsymbol{f}$ and the update equations for Ω use the support set $\mathcal{S}$. When the target function changes over time, the support set $\mathcal{S}$ grows unbounded. This leads to serious computational and runtime issues especially in the lifelong learning setting. The most common solution to this problem is to impose a bound on the number of examples in the support set $\mathcal{S}$. There are several budget maintenance strategies proposed recently ([23, 36, 83]). Unfortunately these schemes cannot be directly used in our setting due to

[9] We set $p = 2$ doe our experiments.

Algorithm 9: Budgeted OOKLA

Initialize: $\mathcal{S} = \emptyset$, $\boldsymbol{\Omega}$
for $t = 1, 2, \cdots$ **do**
  Receive new example $(\boldsymbol{x}_t, i_t)$
  Let $\boldsymbol{f}_{i_t} = \sum_{(\boldsymbol{x}_s, i_s) \in \mathcal{S}} y_s \Omega_{i_s i_t} \phi(\boldsymbol{x}_s)$
  Predict $\hat{y}_t = sign(\boldsymbol{f}_{i_t}(\boldsymbol{x}_t))$
  Receive label y_t
  if $y_t \neq \hat{y}_t$ **then**
    if $|\mathcal{S}| < B$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup (\boldsymbol{x}_t, i_t)$
    else
      Find an example to remove

$$\arg\max_{(\boldsymbol{x}_r, i_r) \in \mathcal{S}} \sum_{k \in [K]} \Omega_{i_r k} \big[y_r \big(\boldsymbol{f}_k - y_r \Omega_{i_r k} \phi(\boldsymbol{x}_r) \big) \phi(\boldsymbol{x}_r) \big] \tag{4.36}$$

      $\mathcal{S} \leftarrow \mathcal{S} \cup (\boldsymbol{x}_t, i_t) \backslash (\boldsymbol{x}_r, i_r)$
    end
    Update $\Omega_{i_t k} \forall k \in [K]$ as in Algorithm 8.
  end
end

the output kernels in our learning formulation. [22] proposed multitask variants of these schemes but they are impractical for the lifelong learning setting. We follow a simple support set removal schemes based on Crammer et al. 2004. In single-task setting, when the number of examples in the support set $\mathcal{S}$ exceeds the limit (say B), a simple removal scheme chooses an example $\boldsymbol{x}_r$ with the highest confidence from $\mathcal{S}$. The confidence of an example $\boldsymbol{x}_r$ is measured using $y_r \boldsymbol{f}(\boldsymbol{x}_r)$ after removing $\boldsymbol{x}_r$ from the support set $\mathcal{S}$.

$$\arg\max_{\boldsymbol{x}_r \in \mathcal{S}} \big[y_r \big(\boldsymbol{f} - y_r \phi(\boldsymbol{x}_r) \big) \phi(\boldsymbol{x}_r) \big] \tag{4.37}$$

We extend the above approach to the multitask and lifelong learning settings. Since the support set $\mathcal{S}$ is shared by all the tasks, we choose an example $\boldsymbol{x}_r$ with high confidence to remove from each task function $\boldsymbol{f}_k$, weighted by the relationship among the tasks. The objective function to choose the example is shown in Equation 4.36. We show in the experiment section that this simple approach is efficient and performs significantly better than the state-of-the-art budget maintenance strategies. Algorithm 9 shows pseudocode of the proposed budgeted learning algorithm.

4.3.4 Two-Stage Budgeted Learning

In lifelong learning setting, the number of tasks is typically large. The support set $\mathcal{S}$ may have hundreds or thousands of examples from all the tasks. Each task does not use all the examples from the support set $\mathcal{S}$. For example, in movie recommendations task, recommendation for each

user (task) can be characterized by just a few movies (subset of examples) in the support set $\mathcal{S}$. Motivated by this observation, we propose a two-stage budgeted learning algorithm for the lifelong learning setting.

Algorithm 10: Two-Stage Budgeted Learning

Initialize: $\mathcal{S} = \emptyset$, $\mathcal{T}_k = \emptyset, k \in [K]$, $\mathbf{\Omega}$
for $t = 1, 2, \cdots$ **do**
  Receive new example $(\boldsymbol{x}_t, i_t)$
  Let $\boldsymbol{f}_{i_t} = \sum_{(\boldsymbol{x}_s, i_s) \in \mathcal{T}_{i_t}} y_s \Omega_{i_s i_t} \phi(\boldsymbol{x}_s)$
  Predict $\hat{y}_t = sign(\boldsymbol{f}_{i_t}(\boldsymbol{x}_t))$
  Receive label y_t
  if $y_t \neq \hat{y}_t$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{t\} \backslash \{r\}$
    /* r=∅ if $|\mathcal{S}| < B$, else choose r using Equation 4.36 */
    for $k \in [K]$ **do**
      $\mathcal{T}_k \leftarrow \mathcal{T}_k \cup \{l\} \backslash \{r\}$
      /* remove r from $\mathcal{T}_k$; choose l from $\mathcal{S} - \mathcal{T}_k$ */
    end
    Update $\Omega_{i_t k} \forall k \in [K]$ as in Algorithm 8.
  end
end

Algorithm 10 shows pseudocode of the proposed two-stage budgeted learning algorithm. In addition to the support set $\mathcal{S}$, we maintain task-specific support set $\mathcal{T}_k$. We choose the budget for each task (say L) where $L <<< B$. Similar to the removal strategies for $\mathcal{S}$, we remove an example from $\mathcal{T}_k$ when $|\mathcal{T}_k| > L$ and replace with an example from the set $\mathcal{S} - \mathcal{T}_k$. The proposed two-stage approach provides better runtime complexity compared to the budgeted algorithm in Algorithm 9. Since only a subset of tasks may hold an example from $\mathcal{S}$, the removal step in Equation 4.36 requires only a subset of tasks for choosing an example. This improves the runtime per iteration significantly when the number of tasks is large. One may consider a different budget size for each task L_k based on the complexity of the task.

In addition, the proposed two-stage budgeted learning algorithm provides an alternative approach to using state-of-the-art budget maintenance strategies. For example, it is easier to use the Projectron algorithm ([83]) on $\mathcal{T}_k$, rather than on $\mathcal{S}$.

4.3.5 Experiments

In this section, we evaluate the performance of our algorithms. All reported results are averaged over 10 random runs on permutations of the training data. Unless otherwise specified, all model parameters are chosen via 5-fold cross validation.

Benchmark Datasets

We use three benchmark datasets, commonly used for evaluating online multitask learning. Details are given below:

Newsgroups Dataset[10] consists of 20 tasks generated from two subject groups: *comp* and *talk.politics*. We paired two newsgroups, one from each subject (e.g.,comp.graphics vs talk.politics.guns), for each task. In order to account for positive/negative correlation between the tasks, we randomly choose one of the newsgroups as positive $(+)$ or negative $(-)$ class. Each post in a newsgroup is represented by a vocabulary of approximately $60K$ unique features.

Spam Dataset[11] We use the dataset obtained from ECML PAKDD 2006 Discovery challenge for the spam detection task. We used the task B challenge dataset, which consists of labeled training data from the inboxes of 15 users. We consider each user as a single task and the goal is to build a personalized spam filter for each user. Each task is a binary classification problem: spam $(+)$ or non-spam $(-)$ and each example consists of approximately $150K$ features representing term frequency of the word occurrences. Some spam is universal to all users (e.g. financial scams), but some messages might be useful to certain affinity groups and spam to most others. Such adaptive behavior of each user's interests and dis-interests can be modeled efficiently by utilizing the data from other users to learn per-user model parameters.

Sentiment Dataset[12] We also evaluated our algorithm on product reviews from amazon. The dataset contains product reviews from 25 domains. We consider each domain as a binary classification task. Reviews with rating > 3 were labeled positive $(+)$, those with rating < 3 were labeled negative $(-)$, reviews with rating = 3 are discarded as the sentiments were ambiguous and hard to predict. Similar to the previous datasets, each example consists of approximately $350K$ features representing term frequency of the word occurrences.

We choose 2000 examples (100 posts per task) for *20 Newsgroups*, 1500 emails for *spam* (100 emails per user inbox) and 2500 reviews for *sentiment* (100 reviews per domain) as training set for our experiments. Note that we intentionally kept the size of the training data small to simulate the lifelong learning setting and drive the need for learning from previous tasks, which diminishes as the training sets per task become large. Since these datasets have a class-imbalance issue (with few $(+)$ examples as compared to $(-)$ examples), we use average Area Under the ROC Curve (AUC) as the performance measure on the test set.

Results

To evaluate the performance of our proposed algorithm (*OOKLA*), we use the three datasets (*Newsgroups*, *Spam* and *Sentiment*) for evaluation and compare our proposed methods to 5 baselines. We implemented Perceptron and Passive-Aggressive algorithm (*PA*) [33] for online multitask learning. Both Perceptron and *PA* learn independent model for each task. These two baselines do not exploit the task-relationship or the data from other tasks during model update. Next, we implemented two online multitask learning related to our approach: *FOML* – initializes Ω with fixed weights [25], Online Multitask Relationship Learning (*OMTRL*) [96] – learns a task

[10] http://qwone.com/~jason/20Newsgroups/
[11] http://ecmlpkdd2006.org/challenge.html
[12] http://www.cs.jhu.edu/~mdredze/datasets/sentiment

Table 4.6: Average performance on three datasets: means and standard (test set) scores over 10 random shuffles.

Models	**Newsgroups**			**Spam**			**Sentiment**		
	AUC	*nSV*	*Time (s)*	*AUC*	*nSV*	*Time (s)*	*AUC*	*nSV*	*Time (s)*
Perceptron	0.7974 (0.01)	1421.6	8.36	0.8621 (0.01)	1110.0	9.23	0.6363 (0.01)	1018.8	255.3
PA	0.7633 (0.02)	1926.6	9.89	0.8394 (0.01)	1482.1	11.5	0.5321 (0.01)	2500.0	357.7
FOML	0.7830 (0.01)	1592.0	18.46	**0.8803** (0.01)	1228.2	16.0	0.6484 (0.01)	1030.0	266.0
OMTRL	0.7694 (0.01)	1560.4	72.36	0.8583 (0.01)	1266.3	45.4	0.5651 (0.01)	1057.6	328.2
OSMTL	0.7969 (0.02)	1426.2	18.18	0.8611 (0.01)	1115.1	17.3	0.6425 (0.01)	1021.8	264.3
OOKLA-*sum*	**0.8512** (0.02)	884.9	15.58	**0.8825** (0.01)	833.4	17.3	0.6261 (0.01)	925.0	306.6
OOKLA-*exp*	0.8382 (0.02)	886.9	15.50	**0.8819** (0.01)	1062.5	19.2	**0.6731** (0.01)	882.2	302.8

covariance matrix along with task parameters. Since *OMTRL* requires expensive calls to SVD routines, we update the task-relationship matrix every 10 iterations. In addition, we compare our proposed methods against the performance of Online Smooth Multitask Learning (*OSMTL*) which learns a probabilistic distribution over all tasks, and adaptively refines the distribution over time [79]. We implement two versions of our proposed algorithm with different update rules for the task-relationship matrix: *OOKLA-sum* (Equation 4.34 OOKLA with sum update) *OOKLA-exp* (Equation 4.35 OOKLA with exponential update) as shown in Algorithm 8.

Table 4.6 summarizes the performance of all the above algorithms on the three datasets. In addition to the AUC scores, we report the average total number of support vectors (*nSV*) and the CPU time taken for learning from one instance (*Time*).

From the table, it is evident that both OOKLA-*sum* and OOKLA-*exp* outperform all the baselines in terms of both AUC and *nSV*. This is expected for the two default baselines (*Perceptron* and *PA*). The update rule for *FOML* is similar to ours but using fixed weights. The results justify our claim that learning the task-relationship matrix adaptively leads to improved performance. As expected, both *OOKLA* and *OSMTL* consume less or comparable CPU time than the subset of baselines which take into account learning inter-task relationships. Unlike in the *OMTRL* algorithm that recomputes the task covariance matrix every (10) iteration using expensive SVD routines, the task-relationship matrix in our proposed methods (and *OSMTL*) are updated independently for each task. We implement the *OSMTL* with exponential update for our experiments as it has shown to perform better than the other baselines. One of the major drawbacks of *OSMTL* is that it learn only the positive correlations between the tasks. The performance of *OSMTL* worsens when the tasks are negatively correlated. As we can see from the table, our proposed methods outperform *OSMTL* significantly in the *Newsgroup* dataset.

Table 4.7 compares the proposed methods with different budget schemes and budget sizes in terms of test set AUC scores and the runtime. We use *OOKLA-sum* for this experiment. We set the value of B to $\{50, 100, 150\}$ for all the datasets. We compare our proposed budgeted

Table 4.7: Evaluation of proposed methods with different budgets: means and standard AUC (test set) scores over 10 random shuffles.

Models	**Newsgroups**			**Spam**			**Sentiment**		
Budget *(B)*	*50*	*100*	*150*	*50*	*100*	*150*	*50*	*100*	*150*
RBP	0.6272 (0.07) 6.5s	0.6954 (0.03) 6.4s	0.7373 (0.01) 7.4s	0.6426 (0.06) 14.5s	0.7197 (0.03) 18.5s	0.7536 (0.04) 16.3s	0.5679 (0.02) 1772.4s	0.5790 (0.02) 1666.9s	0.5889 (0.02) 1860.2s
Forgetron	0.6353 (0.07) 9.1s	0.6860 (0.05) 8.6s	0.7240 (0.01) 9.9s	0.6546 (0.06) 20.5s	0.7403 (0.04) 21.4s	0.7339 (0.04) 21.9s	0.5579 (0.02) 2219.1s	0.5847 (0.02) 2097.0s	0.6195 (0.02) 2272.6s
Projectron	0.6399 (0.03) 16.7s	0.6984 (0.02) 16.6s	0.6986 (0.02) 17.6s	0.6352 (0.03) 24.7s	0.6041 (0.05) 26.1s	0.6402 (0.05) 27.9s	0.5215 (0.01) 2712.0s	0.5187 (0.01) 2622.5s	0.5221 (0.02) 2804.5s
Budget -OKL	0.6810 (0.02) 6.3s	0.7106 (0.03) 10.2s	0.7499 (0.03) 13.4s	0.6472 (0.03) 18.8s	0.6471 (0.03) 22.6s	0.6552 (0.04) 24.7s	0.5683 (0.02) 1708.8s	0.5732 (0.01) 1674.4s	0.6153 (0.01) 1935.2s
Budget -OOKLA	**0.7576** (0.04) 9.9s	**0.7562** (0.04) 11.7s	**0.7749** (0.03) 14.1s	**0.7485** (0.08) 20.1s	**0.7969** (0.04) 24.3s	**0.8472** (0.02) 26.9s	**0.6117** (0.06) 2250.5s	**0.6901** (0.01) 2073.0s	**0.7000** (0.04) 2334.3.3s

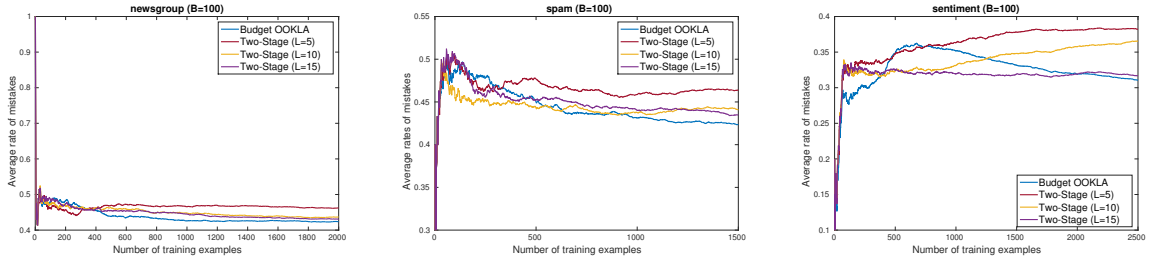

Figure 4.3: Number of training examples vs Average rate of mistakes calculated for different values of L.

learning algorithm (Algorithm 9) with the following state-of-the-art algorithms for online budgeted learning: (1) Random Budgeted Perceptron (RBP) [23]- randomly chooses an example to remove (2) (Self-tuned) Forgetron [36]- forgets the oldest example (3) Projectron++ [83]- projects the new example on the others (4) Budgeted Online Kernel learning (Budget-OKL)- removes an example using equation 4.37 (5) Budgeted Online Output Kernel Learning (Budget-OOKLA)- removes an example using equation 4.36. Following [22], we implement the online multitask version of RBP, Forgetron and Projectron for our experiments.

Table 4.7 shows both the test set AUC scores (first line) and time taken for learning from one instance (including the removal step). It is evident from the table, our proposed budgeted learning algorithm for online multitask learning significantly outperforms the other state-of-the-art budget schemes on most settings. Our proposed algorithm uses the relationship between the tasks efficiently to choose the next example for removal.

Finally, we evaluate the performance of the proposed two-stage budgeted scheme compared to the Algorithm 9. To study the effect of different budget sizes L, we compute the cumulative mistake rate $\frac{\sum_t \mathbb{I}\{y_t \neq \hat{y}_t\}}{t}$ measured at each time-step t. Figures 4.3 show average rate of mistakes against the number of training examples seen so far on *newsgroup*, *spam* and *sentiment* datasets for different values of L. We fix the value of $B = 100$. As we can see from the figures, even for small values of L, the average mistake rate of the two-stage budgeted learning algorithm is comparable to the model which uses all the examples from the support set $\mathcal{S}$. We observe similar trend in the test set AUC scores. On average, we achieved over 16% improvement in running time compared to the budget maintenance scheme in Algorithm 9. We believe that the time consumption and the performance improvement will be even better for applications with larger numbers of tasks.

4.4 Conclusions

In this chapter, we proposed novel algorithms for lifelong learning problem. We proposed a self-paced learning framework for multiple tasks that jointly learns the latent task weights and shared knowledge from all the tasks. The proposed method iteratively updates the shared knowledge based on these task weights and thus improves the learning performance. By allowing the τ to take the probabilistic interpretation, we can easily see which tasks are easier to learn at any iteration, and prefer those for task selection.

We proposed a novel framework for multitask and lifelong learning that factors the task parameters into a shared feature representation and a task structure to learn from multiple related tasks. We formulated two approaches, motivated from recent work in multitask latent feature learning. The first (*BiFactor* MTL), decomposes the task parameters $\mathbf{W}$ into two low-rank matrices: latent feature representation $\mathbf{F}$ and task structure $\mathbf{G}$. As this approach is restrictive on the number of clusters in the latent feature and task space, we proposed a method called *TriFactor* MTL, which introduces an additional degree of freedom to permit different clusterings. We developed a highly scalable and efficient learning algorithm using conjugate gradient descent and generalized Sylvester equations. We extended the proposed approaches to the lifelong learning setting where the latent representations $\mathbf{F}$ and $\mathbf{S}$ (in case of *TriFactor* MTL) to transfer to the novel tasks. Extensive empirical analysis on both synthetic and real datasets show that *Trifactor* multitask learning outperforms the other state-of-the-art multitask and lifelong baselines, thereby demonstrating the effectiveness of the proposed approach.

We proposed a lifelong learning algorithm using output kernels. The proposed method efficiently learns both the model and the inter-task relationships at each iteration. Our update rules for learning the task relationship matrix, at each iteration, were motivated by the recent work in output kernel learning. In order to handle the memory explosion from an unbounded support set in the lifelong learning setting, we proposed a new budget maintenance scheme that utilizes the task relationship matrix to remove the least-useful (high confidence) example from the support set. In addition, we proposed a two-stage budget learning scheme based on the intuition that each task only requires a subset of the representative examples in the support set for efficient learning. It provides a competitive and efficient approach to handle large number of tasks in many real-life applications.

The effectiveness of our algorithms are empirically verified over several benchmark datasets,

outperforming several competitive baselines both in the unconstrained case and the budget-limited case, where selective forgetting was required. Future research includes other forms of budget limits (e.g. time or reprocessing capacity), more complex inter-task relationships (e.g. hierarchical), and additional factors such as task reliability (e.g. prefer transferring from a well-trained related task, vs from a relatively data-sparse one, whose model is less reliable).

Chapter 5

Conclusions

Inspired by established human learning principles, we proposed online and adaptive algorithms for learning from multiple related tasks and leveraging the acquired knowledge to learn new concepts and solve new problems efficiently. We leveraged the inherent relationship between the tasks and their shared representation to improve the performance of multitask and lifelong learning problems. Our work in multitask and lifelong learning scales to large-scale applications where both the tasks and the examples of the tasks arrive in an online fashion.

Our contributions in this thesis will be useful in several applications such as spam detection, sentiment analysis, stock market prediction, object detection, autonomous robot learning, etc. The effectiveness of the proposed algorithms is empirically verified over several benchmark datasets. Future research includes other forms of budget limits (e.g. time or reprocessing capacity), more complex inter-task relationships (e.g. hierarchical), structural constraints in the environment and additional factors such as task reliability (e.g. prefer transferring from a well-trained related task, vs from a relatively data-sparse one, whose model is less reliable).

Appendix A

Online Multitask Learning Proofs

A.1 Second-order OSMTL

In case of squared error loss, the second order online algorithm for the proposed OSMTL is given below.

Algorithm 11: Second-order Smoothed MTL (Supplementary)

Input: Number of Rounds T, Regularization Parameters $\gamma, \lambda > 0$
Initialize $\mathbf{S}^0 = \emptyset, \mathbf{v}_j^0 = \mathbf{0}_p\ \eta$ s.t. $\eta_j = \mathbf{e}_j, j = 1 \dots K$
for round $t = 1, 2, \dots$ T

1. *Receive* $\{x_1^t, x_2^t, \cdots, x_K^t\}$
2. *Predict* $\hat{y}_j^t = sign(\mathbf{w}_j^t x_j^t)$, for $j = 1 \dots K$
   where $\mathbf{w}_j^t = (\gamma I + \mathbf{S}_j^{t-1}(\mathbf{S}_j^{t-1})^\top)^{-1}\mathbf{v}_j^{t-1}$
3. *Receive* $y_1^t, y_2^t \dots y_K^t$
4. *Update* each task $j = 1 \dots K$ if $y_j^t \neq \hat{y}_j^t$:
   (a) $\mathbf{v}_j{}^t = \mathbf{v}_j{}^{t-1} + \sum_k \eta_{jk}^t y_k^t x_k^t$
   (b) *Set* $\mathbf{S}_j^t = [\mathbf{S}_j^{t-1}, [\eta_{jk}^t x_k^t]_{k=1}^K]$
   (c) *for* $k = 1 \dots K$, update η_{jk}^t as in OSMTL

A.2 Proof of Theorem 1

Proof. Define $\Delta_k^{(t)} \stackrel{def}{=} \|w_k^{(t)} - w_k^*\|^2 - \|w_k^{(t+1)} - w_k^*\|^2$.

We can first upper bound $\sum_{t\in T} \Delta_k^{(t)}$ via $\sum_{t\in[T]} \Delta_k^{(t)} = \sum_{t\in[T]} \|w_k^{(t)} - w_k^*\|^2 - \|w_k^{(t+1)} - w_k^*\|^2 = \|w_k^{(0)} - w_k^*\|^2 - \|w_k^{(T+1)} - w_k^*\|^2 \leq \|w_k^*\|^2$.

We further notice any non-zero $\Delta_k^{(t)}$ can be lower-bounded via

$$\begin{aligned}
\Delta_k^{(t)} =& \|w_k^{(t)} - w_k^*\|^2 - \|w_k^{(t)} - C\sum_{j\in[K]^+}\eta_{kj}^{(t)}\ell_{kj}^{(t)\prime} - w_k^*\|^2 && \text{(A.1)}\\
=& 2Cw_k^{(t)}\sum_{j\in[K]^+}\eta_{kj}^{(t)}\ell_{kj}^{(t)\prime} - 2Cw_k^*\sum_{j\in[K]^+}\eta_{kj}^{(t)}\ell_{kj}^{(t)\prime} - C^2\|\sum_{j\in[K]^+}\eta_{kj}^{(t)}\ell_{kj}^{(t)\prime}\|_2^2 && \text{(A.2)}\\
\geq& 2C\sum_{j\in[K]^+}\eta_{kj}^{(t)}\big(\ell_{kj}^{(t)} - 1\big) - 2C\sum_{j\in[K]^+}\eta_{kj}^{(t)}\big(\ell_{kj}^{(t)*} - 1\big) - C^2\|\sum_{j\in[K]^+}\eta_{kj}^{(t)}y_j^{(t)}x_j^{(t)}\|_2^2 && \text{(A.3)}\\
=& 2C\sum_{j\in[K]^+}\eta_{kj}^{(t)}\ell_{kj}^{(t)} - 2C\sum_{j\in[K]^+}\eta_{kj}^{(t)}\ell_{kj}^{(t)*} - C^2\|\sum_{j\in[K]^+}\eta_{kj}^{(t)}y_j^{(t)}x_j^{(t)}\|_2^2 && \text{(A.4)}\\
\geq& 2C\eta_{kk}^{(t)}\ell_{kk}^{(t)} - 2C\sum_{j\in[K]^+}\eta_{kj}^{(t)}\ell_{kj}^{(t)*} - C^2\Big(\sum_{j\in[K]^+}\eta_{kj}^{(t)}\|x_j^{(t)}\|_2\Big)^2 && \text{(A.5)}\\
\geq& 2C\eta_{kk}^{(t)}\big(\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}\big) - 2C\sum_{j\in[K]^+,j\neq k}\eta_{kj}^{(t)}\ell_{kj}^{(t)*} - C^2R^2 && \text{(A.6)}\\
\geq& 2C\alpha\big(\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}\big) - 2C(1-\alpha)\ell_{kk}^{(t)*} - 2C\sum_{j\in[K]^+,j\neq k}\eta_{kj}^{(t)}\ell_{kj}^{(t)*} - C^2R^2 && \text{(A.7)}\\
=& 2C\alpha\big(\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}\big) - 2C(1-\alpha)\Big(\ell_{kk}^{(t)*} + \sum_{j\in[K]^+,j\neq k}\eta_{kj}^{(t)}\ell_{kj}^{(t)*}\Big) - C^2R^2 && \text{(A.8)}\\
\geq& 2C\alpha\big(\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}\big) - 2C(1-\alpha)\Big(\ell_{kk}^{(t)*} + \max_{j\in[K]^+,j\neq k}\ell_{kj}^{(t)*}\Big) - C^2R^2 && \text{(A.9)}
\end{aligned}$$

Combining the aforementioned upper and lower bound over $\sum_{t\in[T]}\Delta_k^{(t)}$, we have

$$\sum_{t\in[T]}(\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}) \leq \frac{1}{2C\alpha}\|w_k^*\|^2 + \frac{(1-\alpha)T}{\alpha}\Big(\ell_{kk}^{(t)*} + \max_{j\in[K]^+,j\neq k}\ell_{kj}^{(t)*}\Big) + \frac{CR^2T}{2\alpha} \qquad \text{(A.10)}$$

□

A.3 Proof of Corollary 2

Proof. By setting $\alpha = \frac{\sqrt{T}}{1+\sqrt{T}}$ and $C = \frac{1+\sqrt{T}}{T}$, we have

$$\begin{aligned}
\sum_{t\in[T]}(\ell_{kk}^{(t)} - \ell_{kk}^{(t)*}) &\leq \frac{\sqrt{T}}{2}\|w_k^*\|^2 + \sqrt{T}\Big(\ell_{kk}^{(t)*} + \max_{j\in[K]^+,j\neq k}\ell_{kj}^{(t)*}\Big) + \frac{(1+\sqrt{T})^2}{2\sqrt{T}}R^2 && \text{(A.11)}\\
&\leq \frac{\sqrt{T}}{2}\|w_k^*\|^2 + \sqrt{T}\Big(\ell_{kk}^{(t)*} + \max_{j\in[K]^+,j\neq k}\ell_{kj}^{(t)*}\Big) + 2\sqrt{T}R^2 && \text{(A.12)}\\
&= \sqrt{T}\left(\frac{1}{2}\|w_k^*\|^2 + \ell_{kk}^{(t)*} + \max_{j\in[K]^+,j\neq k}\ell_{kj}^{(t)*} + 2R^2\right) && \text{(A.13)}
\end{aligned}$$

Asymptotically, the average regret of our algorithm w.r.t the best predictor w^* in hindsight goes to 0. Since our algorithm depends on C and α, our algorithm needs to know the value of T. We can get rid of the dependence of our regret bound on T using the *doubling trick*. $\square$

Appendix B

Active Learning from Peer Proofs

B.1 Proof of Theorem 3

Proof. We prove our theorem 3 for Algorithm 5 using the following lemma [26].

Lemma 5. *For a given example at round t, $\big((x^{(t)}, k), y^{(t)}\big)$, let α be some constant and γ be the margin. Let $\{w_k^*\}_{k\in[K]}$ be any arbitrary vectors where $w_k^* \in \mathbb{R}^d$ and its hinge loss on the examples $\big(x^{(t)}, y^{(t)}\big)$ is given by $\ell_{kk}^{(t)*} = \big(\gamma - y^{(t)}\langle x^{(t)}, w_k^*\rangle\big)_+$. We have the following inequality:*

$$\alpha\gamma + |\hat{p}_k^{(t)}| \leq \alpha\ell_{kk}^{(t)*} \leq \alpha\ell_{kk}^{(t)*} + \frac{1}{2}\|\alpha w_k^* - w_k^{(t-1)}\|^2 - \frac{1}{2}\|\alpha w_k^* - w_k^{(t)}\|^2 + \frac{1}{2}\|w_k^{(t-1)} - w_k^{(t)}\|^2 + \frac{1}{2}\|w_k^{(t-1)} - w_k^{(t)}\|$$

Proof.

$$\begin{aligned}
\gamma - \ell_{kk}^{(t)*} &= \gamma - \big(\gamma - y^{(t)}\langle x^{(t)}, w_k^*\rangle\big)_+ \\
&\leq y^{(t)}\langle x^{(t)}, w_k^*\rangle \\
&= y^{(t)}\langle x^{(t)}, (w_k^* - w_k^{(t-1)} + w_k^{(t-1)})\rangle \\
&= y^{(t)}\langle x^{(t)}, w_k^{(t-1)}\rangle + \frac{1}{2}\|w_k^* - w_k^{(t-1)}\|^2 - \frac{1}{2}\|w_k^* - w_k^{(t)}\|^2 + \frac{1}{2}\|w_k^{(t-1)} - w_k^{(t)}\|^2
\end{aligned}$$

The above inequality holds for any $\gamma > 0$ and any arbitrary vector w_k^*, we replace γ by $\alpha\gamma$ and w_k^* by αw_k^* where α is some constant to be optimized. Since $y^{(t)}\langle x^{(t)}, w_k^{(t-1)}\rangle \leq 0$ when we make a mistake at round t, we get our inequality by using the notation $\hat{p}_k^{(t)} = \langle x^{(t)}, w_k^{(t-1)}\rangle$.

$$\alpha\gamma + |\hat{p}_k^{(t)}| \leq \alpha\ell_{kk}^{(t)*} + \frac{1}{2}\|\alpha w_k^* - w_k^{(t-1)}\|^2 - \frac{1}{2}\|\alpha w_k^* - w_k^{(t)}\|^2 + \frac{1}{2}\|w_k^{(t-1)} - w_k^{(t)}\|^2$$

Note that, for a task m ($m \neq k$), $y^{(t)}\langle x^{(t)}, w_m^{(t-1)}\rangle \leq 0$ is not necessarily true and $\|w_m^{(t-1)} - w_m^{(t)}\|^2 = 0$ since $w_m^{(t)} = w_m^{(t-1)}$ at round t. □

To prove theorem 3, we bound the following two terms: $b_2\big(\alpha\gamma+|\hat{p}_k^{(t)}|\big)+\sum_{\substack{m\in[K]\\ m\neq k}}\tau_{km}^{(t)}|\hat{p}_m^{(t)}|\big(\alpha\gamma+|\hat{p}_k^{(t)}|\big)$ where $\hat{p}_m^{(t)}=\langle x^{(t)},w_m^{(t-1)}\rangle$. Summing over t, we use $w_k^{(t)}=w_k^{(t-1)}$ when there is no mistake $(M^{(t)}=0)$ and $\|w_k^{(t-1)}-w_k^{(t)}\|^2\leq X^2$ otherwise. We use $\sum_{t=1}^{T}[\frac{1}{2}\|\alpha w_k^*-w_k^{(t-1)}\|^2-\frac{1}{2}\|\alpha w_k^*-w_k^{(t)}\|^2]=\frac{\alpha^2}{2}\|w_k^*\|^2$ and $w_k^{(0)}=0$. Consider the m^{th} task in the second term $(m\neq k)$, we have:

$$
\begin{aligned}
&\textstyle\sum_t M^{(t)}Z^{(t)}|\hat{p}_m^{(t)}|\big(\alpha\gamma+|\hat{p}_k^{(t)}|-\frac{X^2}{2}\big)\\
&\leq\sum_t M^{(t)}Z^{(t)}\big[\alpha|\hat{p}_m^{(t)}|\ell_{kk}^{(t)*}+\frac{\alpha^2}{2}|\hat{p}_m^{(t)}|\|w_k^*\|^2\big]\\
&\leq\sum_t M^{(t)}Z^{(t)}\big[\alpha\big(\gamma b_2-|\hat{p}_m^{(t)}|y^{(t)}\langle x^{(t)},w_k^*\rangle\big)_+\frac{\alpha^2 b_2}{2}\|w_k^*\|^2\big]\\
&\leq\sum_t M^{(t)}Z^{(t)}\big[\alpha\big(\gamma b_2-(y^{(t)}\langle x^{(t)},w_m^*\rangle-\frac{1}{2}\|w_m^*\|^2)(y^{(t)}\langle x^{(t)},w_k^{(t-1)}\rangle+\frac{1}{2}\|w_k^*\|^2+\frac{X^2}{2})\big)_+\frac{\alpha^2 b_2}{2}\|w_k^*\|^2\big]\\
&\leq\sum_t M^{(t)}Z^{(t)}\big[\alpha\big(\gamma b_2-(y^{(t)}\langle x^{(t)},w_m^*\rangle-\frac{1}{2}\|w_m^*\|^2)b_2\big)_+\frac{\alpha^2 b_2}{2}\|w_k^*\|^2\big]\\
&\leq\sum_t M^{(t)}Z^{(t)}\big[b_2\big(\alpha\ell_{km}^{(t)*}+\frac{\alpha^2}{2}\|w_m^*\|^2+\frac{\alpha^2}{2}\|w_k^*\|^2\big)\big]
\end{aligned}
$$

where we have used the inequalities in Lemma 5 for both the tasks k and m $(m\neq k)$. We set $b_2=b_1+\frac{1}{2}\|w_k^*\|^2+\frac{X^2}{2}$. We choose $\alpha=(2b_1+X^2)/2\gamma$. Now, combining both the terms, we have:

$$
\begin{aligned}
\sum_{t=1}^{T}M^{(t)}Z^{(t)}\Big[\big(b_1+|\hat{p}_k^{(t)}|\big)\big(b_2+\sum_{\substack{m\in[K]\\ m\neq k}}\tau_{km}^{(t)}|\hat{p}_m^{(t)}|\big)\Big]\leq b_2\Big[&\frac{(2b_1+X^2)^2}{8\gamma^2}\big(\|w_k^*\|^2+\sum_{\substack{m\in[K]\\ m\neq k}}\tau_{km}^{(t)}\|w_m^*\|^2\big)\\
&\frac{(2b_1+X^2)}{2\gamma}\Big(\sum_t M^{(t)}Z^{(t)}\ell_{kk}^{(t)*}+\sum_t\sum_{\substack{m\in[K]\\ m\neq k}}\tau_{km}^{(t)}M^{(t)}Z^{(t)}\ell_{km}^{(t)*}\Big)\Big]
\end{aligned}
$$

Taking expectation on both side and using $\tilde{L}_{kk}=\mathbb{E}\Big[\sum_t M^{(t)}Z^{(t)}\ell_{kk}^{(t)*}\Big]$ gives the desired result.

□

Appendix C

Application: Industrial Scientific Workplace Incident Prediction

C.1 Varying Coefficient Models for Temporal Data

Many real world applications, such as stock market, social network, network traffic, genomics, etc, exhibit a time-varying characteristics. In this chapter, we consider one such application, where the data is associated with temporal information (time stamp) and a complex, time-varying dependency structure over the feature set (correlation graph). We consider a novel problem closely related to the lifelong learning setting called *Varying Coefficient* models. This chapter focuses on an application that involves the prediction of number of workplace incidents based on the safety inspection data. Typically, the data is collected manually from different locations for different projects by safety inspectors, but these data may also be collected from sensors deployed in different locations.

The observation are recorded based on *safety checklist*, a set of simple yes/no questions such as 'Is the person wearing a head protection?' or 'Are there proper ventilation for the job?'. These questions are categorized and they do overlap for different projects handled in a location. The severity of the incidents ranges from simple sprain to person's death and are collected based on *OSHA* standards [1]. In this year alone, there have been 825 fatalities reported to OSHA from across the country. There is a necessity to predict the number of incidents in the future to prevent the actual injuries.

As we can see in Figure C.1, the number of incidents change with time and, clearly, varying coefficient model can explain this behavior better than any other parametric model. Similar trends are seen in number of inspections and projects. In addition, we observe that the feature set can be represented as a graph (for e.g., correlation graph), where each project in a location uses a subgraph/subset of this feature set. Each project exhibit a different dependency structure over the feature set and start and end dates of a project are unknown or not available.

The varying coefficient model is an important generalization of linear regression model and has been studied extensively over the past decade [45, 48]. The model assumes that the regression coefficients are an unknown function of some other variables, called effect modifiers [46]. In this

[1] https://www.osha.gov/

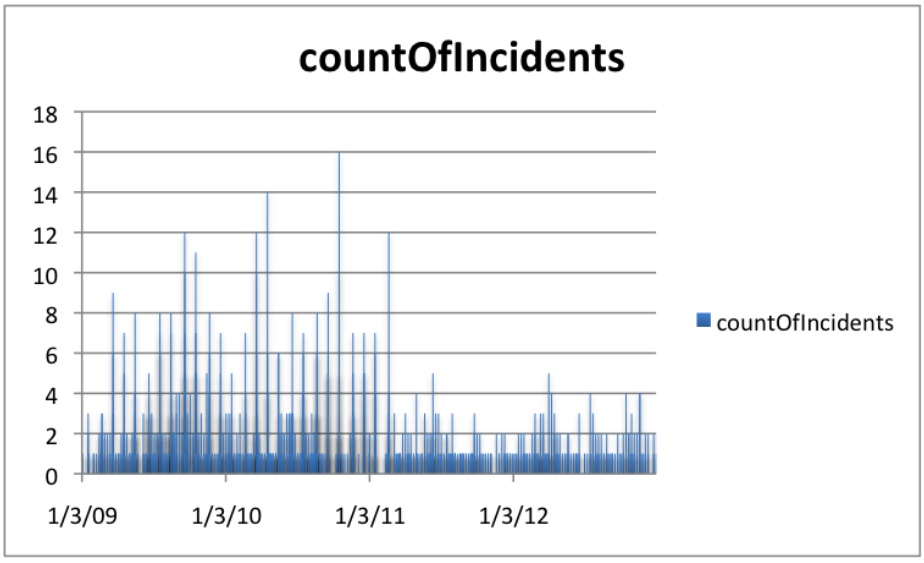

Figure C.1: Number of Incidents/Inspections Vs Week for Workplace Incident Dataset.

work, we consider the regression coefficients as functions of time. For a set of random variables $\mathbf{Y}, \mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_p}$ and $\mathbf{T}$, a time-varying coefficient model has the following form:

$$Y_i = \mathbf{X}_i' \boldsymbol{\beta}(t_i) + \epsilon_i, i = 1, \ldots, N$$

We assume that the regression coefficient function is multidimensional piecewise constant with coefficient values and the dependency structure, i.e, there exists a K-way partition of the sample $\{1, \ldots, N\}$ i.e., $[t_1 = \tau_1 < \tau_2 < \ldots < \tau_K = t_N]$, such that, $\boldsymbol{\beta}(t) = \boldsymbol{\beta}_{\tau_k}$, for $t \in [\tau_{(k-1)}, \tau_k)$ and the correlation graph associated with this interval, G_{τ_k} is same for all $\mathbf{X}_i, i \in [\tau_{(k-1)}, \tau_k)$. Since we are dealing with high dimensional problem, we use the $l1-$sparsity constraint to select the subset of features (or subgraph of the features) relevant to the outcome ([108]).

Our problem involves two unknowns: the K-way partition set and the graph constrained and sparsity constrained regression coefficient function. Note that both the size of the partition set (K) and the actual partition set are unknown. Several papers have discussed this problem (multiple change-point detection, [11]). Most recently, [67] and [110] proposed change-point detection for signal approximation with lasso and group lasso. While the former is proposed for one dimensional signal approximation, the latter can be extended to multidimensional signal approximation.

Based on [67], Kolar et al. (2009) estimated the partition boundaries for the varying coefficient models with fused lasso, which penalize the coefficient values of adjacent temporal difference in each dimension separately. In our application, the model requires that both the coefficient values and their dependency structure changes at the (unknown) partition boundaries. To enforce this constraint, we estimate the multiple change points with fused group lasso, instead of fused lasso penalty. It ensures that all the coefficient values change at the partition boundaries, with which we can estimate correlation graph for each partition separately [110]. From these partitions, we can estimate the regression coefficient with graph and sparsity constraints. In this work, we use graph-structured fusion penalty to estimate the coefficients with network constraints [57, 68].

In the next section, we present temporally-fused lasso and its variants. We provide the parameter estimation for our temporal models. Finally, we demonstrate the performance of our temporal models on simulated and workplace incident dataset, followed by conclusion.

C.1.1 The Methodology

Notations

Suppose we have a sample of N observations, each represented by a p-dimensional feature vector. Let $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_p}) \in \mathbb{R}^{N \times p}$ be the input matrix and $\mathbf{Y} \in \mathbb{R}^N$ represent the outcome variable. We assume a varying coefficient model discussed in the introduction. Let $\boldsymbol{\beta}(t_i) \in \mathbb{R}^p$ be the regression coefficient vector associated with the ith observation. We assume that the feature variables $\mathbf{x_j}$ are centered to have zero mean and unit variance, and the outcome variable $\mathbf{y}$ has mean 0, so that, we consider the model without an intercept.

Temporal Graph-guided Fused Lasso

For our model assumptions, we consider the temporal graph-guided fused lasso penalty to predict workplace incidents. We analyze the following estimator for our application:

$$\min_{\boldsymbol{\beta}} \frac{1}{2} \sum_{i=1}^{N} (Y_i - \mathbf{X}_i \boldsymbol{\beta}(t_i))^2 + \lambda \sum_i ||\boldsymbol{\beta}(t_i)||_1 + \gamma_1 \sum_i ||\boldsymbol{\beta}(t_i)' G(\boldsymbol{\beta}(t_i)||^{\alpha} + \gamma_2 \sum_{i=2}^{N} ||\boldsymbol{\beta}(t_i) - \boldsymbol{\beta}(t_{(i-1)})||_2 \tag{C.1}$$

where $\{\lambda, \gamma_1, \gamma_2\}$ are regression coefficients that depend on N. The loss function estimates the square loss between the true number of incidents and the estimated number of incidents. The first penalty term promotes sparsity in the regression coefficients, the second penalty induces graph structured constraints over the feature set and the third penalty term identifies the partition boundaries.

For $V = \{1, \ldots, p\}$ and $E = \{(m, l) : \forall (m, l) \in V \times V \text{ and } w(r_{ml}) \geq \rho\}$, $G(\boldsymbol{\beta}(t_i))$ is a $|V| \times |E|$ matrix with entries: $(m, (m, l)) = \sqrt{(w(r_{ml}))}$ and $(l, (m, l)) = -sign(r_{ml})\sqrt{(w(r_{ml}))}$. $w(r_{ml}) \in \mathbb{R}$ denote the weight of the edge $e = (m, l) \in E$. We choose r_{ml} to represent the strength of correlation between $\mathbf{x_m}$ and $\mathbf{x_l}$. In this work, we use a simple strategy for constructing the feature graph G by computing a pairwise correlation between $\mathbf{x_m}$ and $\mathbf{x_l}$ and taking $w(r_{ml}) = |r_{ml}|$.

The parameter α takes the value in $\{1, 2\}$. When $\alpha = 1$, the graph constrained penalty forces the features sharing an edge to take the same coefficient values and when $\alpha = 2$, their coefficient values are closer to each other.

Estimation Procedures

When the partition boundaries and the graph structures of each partition are available a priori, equation C.1 can be easily optimized. With the partition boundaries unknown (and hence the

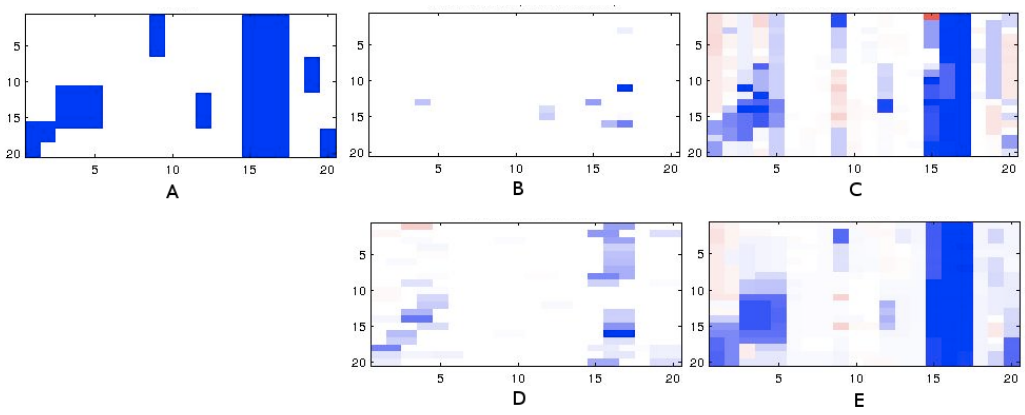

Figure C.2: Regression coefficients estimated by different models for a simulated dataset ($p = 20, K = 20$). We used threshold $\rho = 0.6$ for correlation graph and signal strength $b = 0.8$. Blue pixels indicate positive values. (A) True coefficient matrix with each row corresponds to $\boldsymbol{\beta}(t_{\tau_k})$ (B) *TESLA* (C) *Our Procedure*.

associated correlation graph), we consider an adaptive procedure to estimate the values of the regression coefficients in equation C.1. Due to the bias introduced by the fusion penalty [90], we use a two-stage procedure: (1) estimate the partition boundaries (2) estimate the regression coefficients with *l1*-sparsity and graph constraints.

Estimating Partition Boundaries

We first estimate the partition boundaries with group fused lasso.

$$\min_{\boldsymbol{\beta}} \frac{1}{2} \sum_{i=1}^{N} (Y_i - \mathbf{X}_i \boldsymbol{\beta}(t_i))^2 + \gamma_2 \sum_{i=2}^{N} ||\boldsymbol{\beta}(t_i) - \boldsymbol{\beta}(t_{(i-1)})||_2 \tag{C.2}$$

Proposition 6. *Given dataset* $(\mathbf{X}, \mathbf{Y})$*, we define*

$$\mathbf{X}^{\dagger} = (\mathbf{x}_1^{\dagger}, \mathbf{x}_2^{\dagger}, \ldots, \mathbf{x}_p^{\dagger}) \in \mathbb{R}^{N \times Np},$$

$$\mathbf{x}_i^{\dagger} = \begin{pmatrix} X_1 & 0 & \cdots & 0 \\ X_2 - X_1 & X_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ X_N - X_{N-1} & X_N - X_{N-1} & \cdots & X_N - X_{N-1} \end{pmatrix}$$

(we removed the subscript i for clarity)

$$\boldsymbol{\beta}^{\dagger} = (\boldsymbol{\beta}_{t_1}^{\dagger} \boldsymbol{\beta}_{t_2}^{\dagger} \ldots \boldsymbol{\beta}_{t_N}^{\dagger})' \in \mathbb{R}^{Np}$$
$$\boldsymbol{\beta}_{t_i}^{\dagger} = \boldsymbol{\beta}_{t_i} - \boldsymbol{\beta}_{t_{i-1}}, \text{ for } i = 2 \ldots N$$
$$\text{and } Y_i^{\dagger} = Y_i - Y_{i-1}, \; Y_1^{\dagger} = Y_1$$

Algorithm 12: FISTA for detecting multiple change points with group fused lasso

Data: $\boldsymbol{\beta}^0, \mathbf{X}, \mathbf{Y}, \gamma_2, L-$Lipschitz constant
Result: $\hat{\boldsymbol{\beta}}$
$\boldsymbol{w}^0 = \boldsymbol{\beta}^0, \alpha^1 = 1;$
while *Convergence met* **do**
$g^k = -\mathbf{X}'(\mathbf{Y} - \mathbf{X}\boldsymbol{w^k})$
$\boldsymbol{\beta}^k = \left(1 - \frac{\gamma_2}{L||w^k - \frac{1}{L}g^k||_2}\right)_+ (w^k - \frac{1}{L}g^k)$
$\alpha^k = (w^{k+1} - \boldsymbol{\beta}^k)'(\boldsymbol{\beta}^{k+1} - \boldsymbol{\beta}^k) > 0?1 : \alpha^k$
$\alpha^{k+1} = \frac{1}{2}1 + \sqrt{1 + 4(\alpha^k)^2}$
$w^{k+1} = \boldsymbol{\beta}^k + \frac{\alpha^k - 1}{a^{k+1}}(\boldsymbol{\beta}^k - \boldsymbol{\beta}^{k-1})$

Equation C.2 can be written as

$$\min_{\boldsymbol{\beta}^\dagger \in \mathbb{R}^{Np}} \frac{1}{2}||\mathbf{Y}^\dagger - \mathbf{X}^\dagger \boldsymbol{\beta}^\dagger||_2^2 + \gamma_2 \sum_{i=2}^{N} ||(\boldsymbol{\beta}^\dagger)_i||_2 \tag{C.3}$$

with ith group containing elements of the vector $(\boldsymbol{\beta}_{t_i} - \boldsymbol{\beta}_{t_{i-1}})$.

Equation C.3 is an objective function with group lasso penalty, which has been studied extensively. We use FISTA given in Appendix to solve the above optimization problem. From the estimates of $\boldsymbol{\beta}^\dagger$, we can get partition boundaries where $\boldsymbol{\beta}^\dagger \neq 0$.

Estimating Regression Coefficients Let $\tau_k, k = 1 \dots K$ be the partition boundaries estimated from the previous step. We compute the correlation graph for each partition $G(\beta(\tau_k))$. We construct $X^+ = diag(X_{\tau_1}, X_{\tau_2}, \dots, X_{\tau_k})$, a block diagonal matrix with diagonal elements correspond to the matrix with observations from each partition, estimated in the previous step.

$$\min_{\boldsymbol{\beta}} \frac{1}{2} \sum_{i=1}^{N} (Y_i - \mathbf{X}_i \boldsymbol{\beta}(t_i))^2 + \lambda \sum_i ||\boldsymbol{\beta}(t_i)||_1 + \gamma_1 \sum_i ||\boldsymbol{\beta}(t_i)' G(\boldsymbol{\beta}(t_i)||^\alpha$$

When $\alpha = 1$, we can use a coordinate descent algorithm given in Annex A.

Proposition 7. *Given* $\alpha = 2$*, we write*

$$\mathbf{X}^* = \frac{1}{\sqrt{(1+\gamma_1)}} \begin{pmatrix} \mathbf{X}^+ \\ \sqrt{\gamma_1} G' \end{pmatrix}, \mathbf{Y}^* = \begin{pmatrix} \mathbf{Y} \\ 0 \end{pmatrix} \tag{C.4}$$

where $G \in \mathbb{R}^{p \times \sum_\tau |E|_\tau}$ *is a block diagonal with* $|V| \times |E|$ *matrices, one for each partition. Let* $\gamma = \frac{\lambda}{\sqrt{1+\gamma_1}}$ *and* $\boldsymbol{\beta}^* = \sqrt{1+\gamma_1}\boldsymbol{\beta}$*, then equation C.4 is equivalent to:*

$$min_{\boldsymbol{\beta}^*} ||\mathbf{Y}^* - \mathbf{X}^* \boldsymbol{\beta}^*||_2^2 + \gamma ||\boldsymbol{\beta}^*||_1 \tag{C.5}$$

and the solution to equation C.4 is $\boldsymbol{\beta} = \frac{1}{\sqrt{1+\gamma_1}} \boldsymbol{\beta}^*$

Optimizing Equation C.1

When $\alpha = 1$**,** equation 4 contains a non-smooth function which cannot be optimized. There are efficient methods to solve this equation (Such as FISTA, ADMM, etc.), but we use a simplest approach. We consider a smooth approximation of the non-smooth function by introducing additional variables $d_{\tau,j}, d'_{\tau,\tau-1,j}, d''_{\tau,m,l}$ that need to be estimated, along with the regression coefficients.

$$\begin{aligned}\min_{\mathbf{B}} \sum_{\tau=1}^{T} ||\mathbf{Y}_\tau - \mathbf{X}_\tau \boldsymbol{\beta}_\tau||_2^2 + \lambda \sum_{\tau=1}^{T}\sum_{j=1}^{p} \frac{\beta_\tau^j}{d_{\tau,j}} \\ + \gamma_1 \sum_{\tau=2}^{T}\sum_{j=1}^{p} \frac{(\beta_\tau^j - \beta_{\tau-1}^j)^2}{d'_{\tau,\tau-1,j}} + \gamma_2 \sum_{\tau=1}^{T} \sum_{(m,l)\in E_\tau} w^2(r_{ml}) \frac{(\beta_\tau^m - sign(r_{ml})\beta_\tau^m)^2}{d''_{\tau,m,l}}\end{aligned} \tag{C.6}$$

$$\begin{aligned}\text{s.t} \sum_{\tau=2}^{T}\sum_{j=1}^{p} d'_{\tau,\tau-1,j} = 1, \quad \sum_{\tau=1}^{T} \sum_{(m,l)\in E_\tau} d''_{\tau,m,l} = 1, \quad \sum_{\tau=1}^{T}\sum_{j=1}^{p} d_{\tau,j} = 1 \\ d'_{\tau,\tau-1,j} > 0, \quad d''_{\tau,m,l} > 0, \\ d_{\tau,j} > 0, \quad \forall\ \tau, j, m, l\end{aligned}$$

We solve the above problem by optimizing $d_{\tau,j}, d'_{\tau,\tau-1,j}, d''_{\tau,m,l}$ and $\beta_{\tau,j}$. In each iteration, we fix the value of $\beta_{\tau,j}$ and estimate $d_{\tau,j}, d'_{\tau,\tau-1,j}, d''_{\tau,m,l}$, by taking their derivatives and setting it to 0. We get the following update equations for $d_{\tau,j}, d'_{\tau,\tau-1,j}, d''_{\tau,m,l}$:

$$\begin{aligned} d_{\tau,j} &= \frac{|\beta_\tau^j|}{\sum_{\tau',j'} |\beta_{\tau'}^{j'}|} \\ d'_{\tau,\tau-1,j} &= \frac{|\beta_\tau^j - \beta_{\tau-1}^j|}{\sum_{\tau',j'} |\beta_{\tau'}^{j'} - \beta_{\tau'-1}^{j'}|} \\ d''_{\tau,m,l} &= \frac{w(r_{ml})|\beta_\tau^m - sign(r_{ml})\beta_\tau^m|}{\sum_{\tau',(m',l')\in E_\tau} w(r_{m'l'})|\beta_{\tau'}^{m'} - sign(r_{m'l'})\beta_{\tau'}^{l'}|} \end{aligned} \tag{C.7}$$

Based on the current estimates for $d_{\tau,j}, d'_{\tau,\tau-1,j}, d''_{\tau,m,l}$, we optimize over $\beta_{\tau,j}$ using the following update equation:

$$\begin{aligned}\beta_\tau^j = \Bigg\{ \sum_{n=1}^{N_\tau} x_n^j \big(y_n - \sum_{j'\neq j} x_n^{j'} \beta_\tau^{j'}\big) + \gamma_1 \left(\frac{\beta_{\tau-1}^j}{d'_{\tau,\tau-1,j}} \mathbb{1}(\tau > 1) + \frac{\beta_{\tau+1}^j}{d'_{\tau+1,\tau,j}} \mathbb{1}(\tau < T) \right) \\ + \gamma_2 \left(\sum_{(m,j)\in E_\tau} w^2(r_{mj}) \frac{sign(r_{mj})\beta_\tau^m)^2}{d''_{\tau,m,j}} \right) + \gamma_2 \left(\sum_{(j,l)\in E_\tau} w^2(r_{jl}) \frac{sign(r_{jl})\beta_\tau^l)^2}{d''_{\tau,j,l}} \right) \Bigg\} \\ \Bigg/ \Bigg\{ \sum_{n=1}^{N_\tau} x_j^{n2} + \frac{\lambda}{d_{\tau,j}} + \gamma_1 \left(\frac{1}{d'_{\tau,\tau-1,j}} + \frac{1}{d'_{\tau+1,\tau,j}} \right) + \gamma_2 \left(\sum_{(m,j)\in E_\tau} \frac{w^2(r_{mj})}{d''_{\tau,m,j}} + \sum_{(j,l)\in E_\tau} \frac{w^2(r_{jl})}{d''_{\tau,j,l}} \right) \Bigg\}\end{aligned}$$

We repeat the above two steps iteratively until there is a little improvement in the objective function. Regularization parameters λ and $\boldsymbol{\gamma}$ can be learned automatically using K-fold cross-validations. We can choose the regularization parameters that minimizes the BIC criterion.

Implementation Details

1. We use position independent weights $\sqrt{\frac{N}{i*(N-i)}}$ for group lasso in equation (3) to avoid boundary effects.
2. We can see that the results are highly sensitive to the correlated graph. But we have very limited data in each partition. One approach might be to use the data from other location to estimate the correlation graph. This might be misleading the observations in each location are different.
3. To estimate, we need to use an adaptive version of the estimator in equation 3. One approach might be to estimate $\beta^{\dagger}$ and use this to minimize $||\beta^{\dagger} - \tilde{\beta}||_2 + ||\tilde{\beta}||_2$. Alternatively, we used bootstrap samples to estimate change points. Each change point is chosen with probability $\frac{\#ChangePoints\tau_k appearedinbootstrapsamples}{\#Samples}$

C.1.2 Experiments

In this section, we show the performance of the our procedure discussed in the previous section on simulated and real datasets. We use *Lasso* [108], *GRACE* [68] and *TESLA* [60] as our baselines. In addition to our procedure, we consider a model where $G(\beta(t_1)) = G(\beta(t_2)) = \ldots = G(\beta(t_N)$ in equation C.4, i.e., we use the same correlation graph estimated on $(\mathbf{X})$ for all partition instead of estimating correlation graph on $(\mathbf{X}_{\tau_k})$ for each partition. We call it *tGRACE* for temporal data. We choose the regularization parameters λ and $\boldsymbol{\gamma}$ by minimizing BIC criterion. We assume that the input matrix $\mathbf{X}$ and the outcome variable $\mathbf{Y}$ are standardized before the experiments.

Simulation Study

We conduct a simulation study to compare the performance of *Lasso*, *GRACE*, *TESLA* and our procedure. We use $K = 20$ partitions of the input matrix with 20 features for generating simulated datasets. We choose the number of observations in each temporal segment τ_k, $N_{\tau_k} \approx 10$, for both training and test data.

We generate the data with both temporal dependencies and feature correlations using the following procedure: we choose subsets $\{\{1\}, \{1,2\}, \{3,4,5\}, \{1,2,3,4,5\}, \{9\}, \{15,16,17\}, \{12\}, \{19\}, \{20\}\}$ of features as groups, and used these groups to sample a covariance matrix Σ for each group separately. We generate the observations for each partition from a multivariate Gaussian distribution with mean $\mathbf{0}$ and covariance Σ, based on the sparsity pattern of $\boldsymbol{\beta}(t_{\tau_k})$ given in Figure (C.2A). Each non-zero values of $\boldsymbol{\beta}(t_{\tau_k})$ is set to 0.8.

For the first part of the simulated experiment, we restrict our attention to *TESLA* and our procedure for recovering the true sparsity pattern in temporal domain. We estimate the regression coefficients $\boldsymbol{\beta}$ on the simulated training data to compare them. Figure C.2(A) shows the true

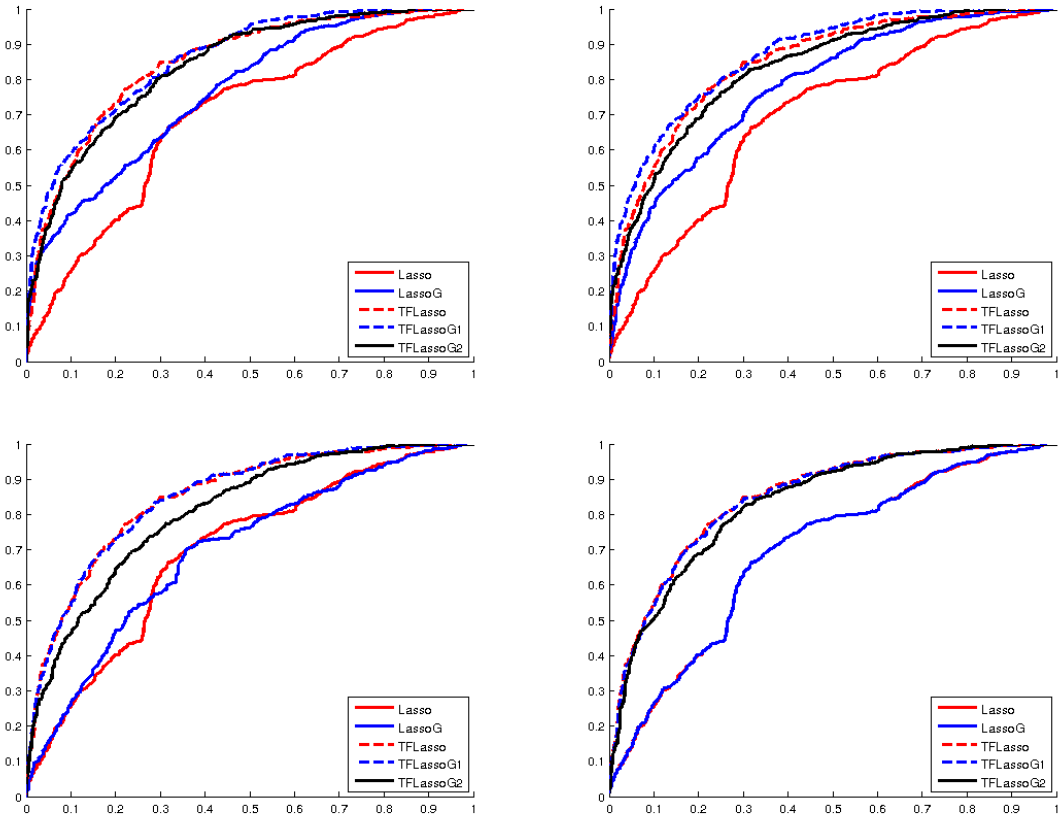

Figure C.3: Comparison of ROC curves for the recovery of true sparsity patterns with varying threshold ρ for feature correlation graph. (a) $\rho = 0.1$, (b) $\rho = 0.3$, (c) $\rho = 0.5$, (d) $\rho = 0.7$. Results are averaged over 50 simulated datasets. We use $b = 0.8$ for signal strength. X-axis: False positive rate; Y-axis: True positive rate.

regression coefficients $\boldsymbol{\beta}$ for each partition. Figures (C.2B) and (C.2C) show the regression coefficients $\hat{\boldsymbol{\beta}}$ estimated by TESLA and our procedure.

For $alpha = 2$, the coefficient estimates of $\boldsymbol{\beta}$, for example $\{15, 16, 17\}$, will be closer to each other, but may not be same. In order to recover the exact sparsity pattern, we use $\alpha = 1$ in equation C.4 for the simulated experiment. Note that there is no need to use $\alpha = 1$ unless the application requires it. We use it in the simulated experiment for the better visualization. We will use $alpha = 2$ for the real data. Clearly we can see that our procedure succeeds in recovering the true model. *TESLA* recovers the temporal smoothness, but without the feature correlation, *TESLA* couldn't recover the exact support of the regression coefficients.

We evaluate the models on test data with the receiver operating characteristics (ROC) curves for the recovery of true sparsity pattern and prediction errors. performance of the our procedure discussed in the previous section on simulated and real datasets. We use *Lasso* , *GRACE* and *TESLA* to compare it against our procedure. We study the importance of threshold ρ for correlation graph used in *GRACE*, *tGRACE* and our procedure.As we can see in Figure C.3, *TESLA*, *tGRACE* and our procedure consistently outperform *Lasso* and *GRACE*, regardless of the threshold ρ, which

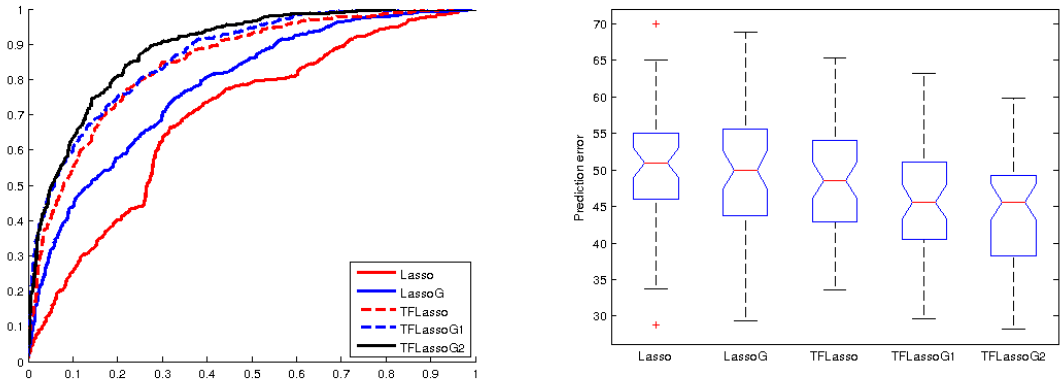

Figure C.4: ROC Curve (left) and prediction error (right) estimated for (A) *Lasso*, (B) *GRACE*, (C) *TESLA* and (D) *tGRACE* with $\rho = 0.3$ and for our procedure (E) with $\rho = 0.85$. The results are averaged over 50 simulated datasets. (Left) X-axis: False positive rate; Y-axis: True positive rate. (Right) Box shows the lower quartile, median, and upper quartile values, and the whiskers show the range of prediction errors in the 50 simulated datasets.

make sense since we are dealing with temporal data.

For lower values of ρ, more edges are added to the correlation graph that includes edges E with strong correlations and edges with weak correlations due to the added noise in both covariance matrix and the input data. It is worth noting that *tGRACE* exhibits better performances than the other models for $\rho = 0.3$, since it uses the correlation constructed from the entire data. This reduces the number of spurious edges added to the edge set E. When the threshold is higher (say $\rho = 0.7$), there are no edges in the feature graph, and thus removes the graph constrained penalty function. As a result, the performance curves of *TESLA* and *tGRACE* almost entirely overlap. Similar performances can be noticed for *GRACE* and Lasso.

Even though the performances of our procedure are better than *Lasso* and *GRACE*, we can see that our procedure performs worse than *TESLA* and *tGRACE* for all the values of ρ. As mentioned before, since the correlation graph $G(\boldsymbol{\beta}(t_{\tau_k}))$ is estimated on a subset of the observation, we notice that significant number of spurious edges are added to the edge set E_{τ_k}, even when $\rho = 0.7$. To overcome this problem, we choose the threshold value separately for our procedure to filter these noisy edges.

We repeat the experiment with $\rho = 0.85$ for our procedure. For the other models, we choose the threshold that gave the best results in Figure C.3 ($\rho = 0.3$). Figure C.4 shows the updated ROC curve and prediction errors for the regression models. We can see that our procedure now performs significantly better than all the other models, by utilizing correlation between the features in each partition.

The evaluation results shown in Figures C.3 and C.4 are averaged over 50 randomly generated datasets.

Location/ Model	Lasso	Elastic Net	GRACE	TESLA	Our Procedure
Complex 0	5.2803	4.7049	3.2828	5.6586	2.2483
Complex 1	14.0825	11.0319	0.5159	2.6186	0.5057
Complex 2	0.3901	0.3891	0.3075	0.2913	0.2932
Complex 3	10.8767	8.3318	0.3132	1.3766	0.3258
Complex 4	4.3131	4.0839	0.5904	0.9435	0.4996
Complex 5	0.9010	0.7970	0.1666	0.6119	0.1429
Complex 6	2.6956	2.6534	0.3143	0.9058	0.2924
All	0.6186	0.6186	0.6173	0.9194	0.5447

Table C.1: Mean Squared Error (MSE) estimated for *Lasso*, *Elastic Net*, *GRACE*, *TESLA* and *Our Procedure* on all datasets. Threshold $\rho = 0.3$

Workplace Incident Dataset

Workplace incident dataset contains inspections and the records of incidents collected from different projects at different locations in 2011 and 2012. The dataset consists of $\approx$ 4000 observations ($\approx$ 2000 observations in each year) with 51 features collected from different locations, each associated with a time stamp. Each observation is a weekly summary of safety inspections at a location with number of incidents happened in that week as an outcome.

We conduct a preliminary experiment to test our real data with each model. For the first part of our experiment, we consider naive assumptions about the dataset. We assume that the partition boundaries are given. We split the dataset into $K = 12$ partitions based on months. We also assume that the features with shared edge take same coefficient values i.e., we set $\alpha = 1$. We use the observations from 2012 for training/validation and the observations from the first month of 2012 as test set. We use the regression coefficient $\beta(t_{12})$ to predict the number of incidents in the test set.

We use training set to learn the regression coefficients and test set to measure the performance of regression models. As in the simulation study, we notice that smaller value for threshold ρ hurts the performance of our procedure. We pick the threshold value for our procedure separately. We randomly sample data from each partition to build the validation set. We use this validation set to choose $\rho = 0.5$ for *GRACE*, *tGRACE*, and $\rho = 0.9$ for our procedure.

Figure C.5 shows the prediction error for each model. We use sum of absolute errors ($||\mathbf{Y} - \hat{\mathbf{Y}}||_1$) to compare the performance of the models. We removed the *Lasso* results since the estimates were beyond the y-axis scale shown in the figure. As we see in Figure C.5 that our procedure performs significantly better than other models. This is good but these results doesn't give any information to explain the model behavior.

Each partition contains weekly observations collected from different locations, so it is hard to interpret the regression coefficient values, as these values might be influenced by the observations from the locations with larger projects. Moreover, we noticed that there are several locations with temporal gap in the observations, either because there were no projects during that period or that location handles small projects. Nonetheless, even with the shaky assumptions and unreliable data, *TESLA*, *tGRACE* and *Our Procedure* did better than *GRACE* and *Lasso*.

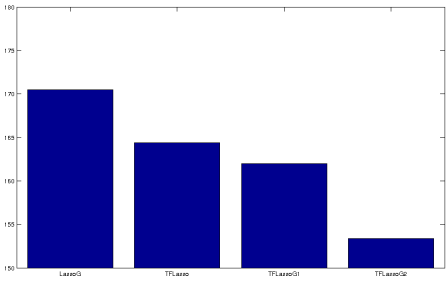

Figure C.5: Prediction errors for the workplace incident dataset. X-axis: (left-to-right) (A) *GRACE*, (B) *TESLA*, (C) *tGRACE* and (D) *Our Procedure*; Y-axis: $||\mathbf{Y} - \hat{\mathbf{Y}}||_1$ (Sum of absolute errors).

In the second part of our experiment, we considered the original problem where the partition set size K and the actual partition set are unknown. In addition, we restricted ourselves to 7 locations for which the observations are available for all two years ($Complex0$, $Complex1$, $Complex2$, $Complex3$, $Complex4$, $Complex5$ and $Complex6$). Each dataset now contains 104 observations, one for each week. We used every other observations for test data, i.e., observations with odd week number belong to training data and the observations with even week number belong to test dataset. We used this way to split the dataset so that we can use the regression coefficient $\boldsymbol{\beta}_{\tau_{t_i}}$ of week i in the training data to predict the number of incidents for the week $(i+1)$ in the test data.

We chose *GRACE* and *TESLA* as our baseline models for *our procedure*. We included *Lasso* and *Elastic Net* to compare the results with [68]. We used Mean Squared Error (MSE: $\frac{1}{N}||\mathbf{Y} - \hat{\mathbf{Y}}||_2^2$) to compare the models. Table C.1 shows the results. In addition, we included the result for dataset generated by combining observations from all 7 locations.

Lasso, *Elastic Net* and *GRACE* results are consistent with the results produced in [68]. We can see that our procedure gave good results in most of the locations. Interestingly, *GRACE* outperforms *TESLA* in most of the cases. This explains the importance of using graph constraints to estimate regression coefficients. *TESLA* performs little better than *GRACE* and *Our Procedure* in $Complex2$ due to the limited number of inspections available for that location. This is consistent with [60]'s results for finite sample data. It is worth noting that *Our Procedure* gave much smalled MSE than other models when we use all the datasets. This is due to the availability of better correlation graphs for each partition estimated with additional data from all locations.

We used the regularization path for *TESLA* and *Our Procedure* to study the model behaviors. We noticed that after computing the change points, almost all the locations contain change points around week 40 and week 90. A simple observation revealed that the number of inspections and the number of incidents were significantly low in the last few weeks of an year due to the holidays, compared to the previous week. This shows that change point detection identifies the seasonality which is common in our application.

Based on our previous observations, we analyzed the regularization paths for the last two partitions under *TESLA* and *Our Procedure*. Figure C.6 shows the regularization paths for *TESLA* and *Our Procedure*. Left column contains the regularization paths for $(K-1)th$ partition and

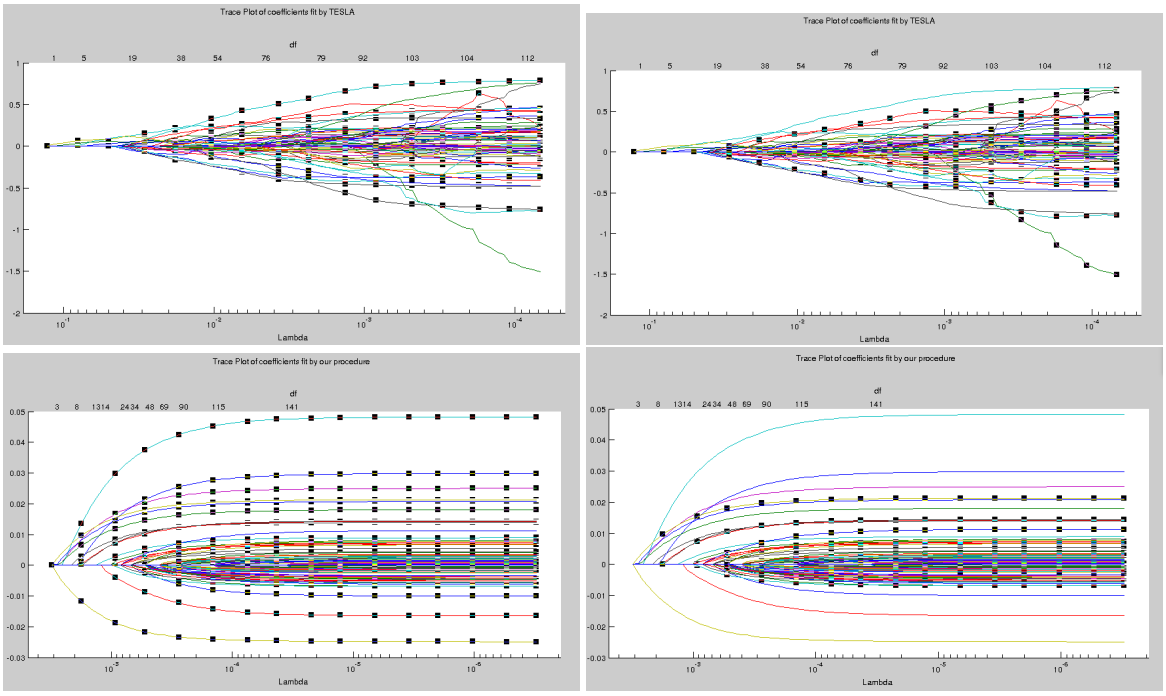

Figure C.6: Regularization Path for *TESLA* (Top) and *Our Procedure* (Bottom) showing the last two partitions for the location (Complex 5)

right column contains the regularization paths for Kth partition for $Complex5$. We can see that with graph constraints over feature set, the regularization paths of our procedure are well formed. The correlated features often travel together in the regularization path. Similar behavior can be noticed in regularization path of *GRACE*. The dotted lines in the left and right paths correspond to the features that belong to $(K-1)th$ and Kth partitions respectively. We can clearly see that the correlation between the features changed between the partition boundaries, which is captured by our procedure.

C.2 Discussions

We considered sparse regression models for our application that encourage temporal smoothness and utilize the graph structure over the feature set to learn the true sparsity pattern in the regression coefficients. We demonstrated with simulated and real datasets that using additional information about the feature correlation can improve the prediction performance.

We noticed that the performance of *tGRACE* and *our procedure* depends on the quality of the available feature graph. In this work, we have used a simple strategy based on pair-wise correlation to construct the feature graph, more sophisticated methods can be employed for constructing this graph. For example, we can learn both structure and regression coefficients together. This is especially useful when we consider a dynamic network where the structure changes frequently over time. Although this work focused on one application, this procedure can be easily extended to other applications such as modeling disease progression, survival analysis, financial data and

genome analysis, etc.

Bibliography

[1] Jacob Abernethy, Peter Bartlett, and Alexander Rakhlin. Multitask learning with expert advice. In *Learning Theory*, pages 484–498. Springer, 2007. 1, 1.1.1, 2.1, 2.1.1, 4.3

[2] Alekh Agarwal. Selective sampling algorithms for cost-sensitive multiclass prediction. *ICML (3)*, 28:1220–1228, 2013. 3, 3.1.1

[3] Alekh Agarwal, Alexander Rakhlin, and Peter Bartlett. Matrix regularization techniques for online multitask learning. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-138*, 2008. 2.1.1, 3.1.1

[4] Arvind Agarwal, Samuel Gerber, and Hal Daume. Learning multiple tasks using manifold regularization. In *Advances in neural information processing systems*, pages 46–54, 2010. 4.1.2, 4.1.3, 4.1.4

[5] Pierre Alquier, The Tien Mai, and Massimiliano Pontil. Regret bounds for lifelong learning. *arXiv preprint arXiv:1610.08628*, 2016. 4

[6] Yonatan Amit, Michael Fink, Nathan Srebro, and Shimon Ullman. Uncovering shared structures in multiclass classification. In *Proceedings of the 24th international conference on Machine learning*, pages 17–24. ACM, 2007. 4.2.1

[7] Haitham B Ammar, Rasul Tutunov, and Eric Eaton. Safe policy search for lifelong reinforcement learning with sub-linear regret. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2361–2369, 2015. 4

[8] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853, 2005. 4.1.2, 4.1.3

[9] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008. 1, 1.1.1, 2.1, 2.2, 2.2.7, 2.2.7, 3, 4, 4.1.4, 4.2, 4.2.1, 4.2.2, 4.2.3

[10] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004. 2.2

[11] Jushan Bai. Estimation of a change point in multiple regression models. *Review of Economics and Statistics*, 79(4):551–563, 1997. C.1

[12] Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. Efficient representations for lifelong learning and autoencoding. In *Conference on Learning Theory*, pages 191–210,

2015. 4

[13] Peter L Bartlett and Marten H Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(Aug):1823–1840, 2008. 3.1.1, 3.2

[14] Aviad Barzilai and Koby Crammer. Convex multi-task learning by clustering. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS-15)*, 2015. 4.2, 4.2.1, 4.2.3, 5, 4.2.3, 4.3.2

[15] Jonathan Baxter et al. A model of inductive bias learning. *J. Artif. Intell. Res.(JAIR)*, 12 (149-198):3, 2000. 4.1.2

[16] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010. 1.1.3, 2.1.1

[17] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM, 2009. 4, 4.1

[18] John Blitzer, Mark Dredze, Fernando Pereira, et al. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, volume 7, pages 440–447, 2007. 2.1.6

[19] Serhat Bucak, Rong Jin, and Anil K Jain. Multi-label multiple kernel learning by stochastic approximation: Application to visual object recognition. In *Advances in Neural Information Processing Systems*, pages 325–333, 2010. 2.2.7

[20] Rich Caruana. Multitask learning. *Machine Learning*, 1(28):41–75, 1997. 4.1.2

[21] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998. 1, 4

[22] Giovanni Cavallanti and Nicolo Cesa-Bianchi. Memory constraint online multitask classification. *arXiv preprint arXiv:1210.0473*, 2012. 4.3.3, 4.3.5

[23] Giovanni Cavallanti, Nicolo Cesa-Bianchi, and Claudio Gentile. Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69(2-3):143–167, 2007. 4.3.3, 4.3.5

[24] Giovanni Cavallanti, Nicolo Cesa-Bianchi, and Claudio Gentile. Linear classification and selective sampling under low noise conditions. In *Advances in Neural Information Processing Systems*, pages 249–256, 2009. 3, 3.1.1

[25] Giovanni Cavallanti, Nicolo Cesa-Bianchi, and Claudio Gentile. Linear algorithms for online multitask classification. *The Journal of Machine Learning Research*, 11:2901–2934, 2010. 1, 1.1.1, 2.1, 2.1.1, 2.1.3, 2.1.6, 3.1.1, 3.2, 4, 4.3, 4.3.5

[26] Nicolo Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Worst-case analysis of selective sampling for linear classification. *Journal of Machine Learning Research*, 7(Jul):1205–1230, 2006. 3, 3.1.1, 3.2, 17, 3.4, B.1

[27] Haim Cohen and Koby Crammer. Learning multiple tasks in parallel with a shared annotator. In *Advances in Neural Information Processing Systems*, pages 1170–1178, 2014. 3.2, 3.3, 21

[28] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Generalization bounds for learning kernels. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 247–254, 2010. 2.2

[29] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Two-stage learning kernel algorithms. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 239–246, 2010. 2.2, 2.2.5

[30] Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. Learning with rejection. In *International Conference on Algorithmic Learning Theory*, pages 67–82. Springer, 2016. 3.1.1

[31] Koby Crammer and Yishay Mansour. Learning multiple tasks using shared hypotheses. In *Advances in Neural Information Processing Systems*, pages 1475–1483, 2012. 1, 1.1.1, 2.1, 2.1.1, 2.1.3, 3, 4.2, 4.2.3, 3, 4.2.3

[32] Koby Crammer, Jaz Kandola, and Yoram Singer. Online classification on a budget. In *Advances in neural information processing systems*, pages 225–232, 2004. 4.3.3

[33] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7: 551–585, 2006. 2.1.6, 4.3.5

[34] N Cristianini. On kernel-target alignment. *Advances in Neural Information Processing Systems*, 2002. 2.2, 2.2.5

[35] Ofer Dekel, Philip M Long, and Yoram Singer. Online learning of multiple tasks with a shared loss. *Journal of Machine Learning Research*, 8(10):2233–2264, 2007. 1, 1.1.1, 2.1, 2.1.1, 2.1.6, 3.1.1, 4.3

[36] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37(5):1342–1372, 2008. 4.3.3, 4.3.5

[37] Ofer Dekel, Claudio Gentile, and Karthik Sridharan. Selective sampling and active learning from single and multiple teachers. *Journal of Machine Learning Research*, 13(Sep): 2655–2697, 2012. 3, 3.1.1, 3.2

[38] Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *Proceedings of the twenty-first international conference on Machine learning*, page 29. ACM, 2004. 4.2

[39] Chris Ding, Tao Li, Wei Peng, and Haesun Park. Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 126–135. ACM, 2006. 4.2

[40] Francesco Dinuzzo, Cheng S Ong, Gianluigi Pillonetto, and Peter V Gehler. Learning output kernels with block coordinate descent. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 49–56, 2011. 4, 4.2, 4.3.1

[41] Pinar Donmez and Jaime G Carbonell. Proactive learning: cost-sensitive active learning with multiple imperfect oracles. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 619–628. ACM, 2008. 3.1.1

[42] Pinar Donmez, Jaime G Carbonell, and Jeff Schneider. Efficiently learning the accuracy

of labeling sources for selective sampling. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 259–268. ACM, 2009. 3.1.1

[43] A Evgeniou and Massimiliano Pontil. Multi-task feature learning. *Advances in neural information processing systems*, 19:41, 2007. 4.1.2, 4.1.3

[44] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117. ACM, 2004. 4.1.2, 4.1.2

[45] Jianqing Fan and Wenyang Zhang. Statistical estimation in varying coefficient models. *Annals of Statistics*, pages 1491–1518, 1999. C.1

[46] Jianqing Fan and Wenyang Zhang. Statistical methods with varying coefficient models. *Statistics and its Interface*, 1(1):179, 2008. C.1

[47] Quanquan Gu and Jie Zhou. Co-clustering on manifolds. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 359–368. ACM, 2009. 4.2

[48] Trevor Hastie and Robert Tibshirani. Varying-coefficient models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 757–796, 1993. C.1

[49] Jingrui He and Rick Lawrence. A graph-based framework for multi-task multi-view learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 25–32, 2011. 2.2

[50] Chris Hinrichs, Vikas Singh, Jiming Peng, and Sterling Johnson. Q-mkl: Matrix-induced regularization in multi-kernel learning with applications to neuroimaging. In *Advances in neural information processing systems*, pages 1421–1429, 2012. 2.2

[51] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 4

[52] Ali Jalali, Sujay Sanghavi, Chao Ruan, and Pradeep K Ravikumar. A dirty model for multi-task learning. In *Advances in Neural Information Processing Systems*, pages 964–972, 2010. 2.2

[53] Pratik Jawanpuria and J Saketha Nath. Multi-task multiple kernel learning. In *Society for Industrial and Applied Mathematics. Proceedings of the SIAM International Conference on Data Mining*, page 828. Society for Industrial and Applied Mathematics, 2011. 2.2, 2.2.1, 2.2.7, 2.2.7

[54] Pratik Jawanpuria, Maksim Lapin, Matthias Hein, and Bernt Schiele. Efficient output kernel learning for multiple tasks. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2015. 4, 4.3.1, 4.3.2

[55] Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. Self-paced learning with diversity. In *Advances in Neural Information Processing Systems*, pages 2078–2086, 2014. 2.1.2, 4.1.1

[56] Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *Proceedings of the 28th International Conference on Machine Learning*

(ICML-11), pages 521–528, 2011. 4.1.4

[57] Seyoung Kim and Eric P Xing. Statistical estimation of correlated genome associations to a quantitative trait network. *PLoS genetics*, 5(8):e1000587, 2009. C.1

[58] Marius Kloft, Ulf Brefeld, Pavel Laskov, Klaus-Robert Müller, Alexander Zien, and Sören Sonnenburg. Efficient and accurate lp-norm multiple kernel learning. In *Advances in neural information processing systems*, pages 997–1005, 2009. 2.2, 2.2.1

[59] Marius Kloft, Ulf Brefeld, Sören Sonnenburg, and Alexander Zien. Lp-norm multiple kernel learning. *The Journal of Machine Learning Research*, 12:953–997, 2011. 2.2

[60] Mladen Kolar, Le Song, and Eric P Xing. Sparsistent learning of varying-coefficient models with structural changes. In *Advances in Neural Information Processing Systems*, pages 1006–1014, 2009. C.1, C.1.2, C.1.2

[61] Meghana Kshirsagar, Jaime Carbonell, and Judith Klein-Seetharaman. Multisource transfer learning for host-pathogen protein interaction prediction in unlabeled tasks. In *NIPS Workshop on Machine Learning for Computational Biology*, 2013. 2.1.1

[62] Meghana Kshirsagar, Jaime Carbonell, and Judith Klein-Seetharaman. Multitask learning for host–pathogen protein interactions. *Bioinformatics*, 29(13):i217–i226, 2013. 2.1.1, 2.2, 4, 4.1.2

[63] Abhishek Kumar and Hal Daume. Learning task grouping and overlap in multi-task learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1383–1390, 2012. 4.1.4, 4.2, 4.2.1, 4.2.3, 4.2.3

[64] Abhishek Kumar, Alexandru Niculescu-Mizil, Koray Kavukcuoglu, and Hal Daume III. A binary classification framework for two-stage multiple kernel learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 2012. 2.2

[65] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010. 2.1.2, 4, 4.1, 4.1.1

[66] Gert RG Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. Learning the kernel matrix with semidefinite programming. *The Journal of Machine Learning Research*, 5:27–72, 2004. 2.2

[67] Céline Levy-leduc and Zaïd Harchaoui. Catching change-points with lasso. In *Advances in Neural Information Processing Systems*, pages 617–624, 2008. C.1

[68] Caiyan Li and Hongzhe Li. Network-constrained regularization and variable selection for analysis of genomic data. *Bioinformatics*, 24(9):1175–1182, 2008. C.1, C.1.2, C.1.2

[69] Changsheng Li, Fan Wei, Junchi Yan, Weishan Dong, Qingshan Liu, and Hongyuan Zha. Self-paced multi-task learning. In *AAAI*, pages 2175–2181, 2017. 4.1

[70] Tao Li and Chris Ding. The relationships among various nonnegative matrix factorization methods for clustering. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 362–371. IEEE, 2006. 4.2

[71] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert.

Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007. 2.2.6

[72] Jun Liu, Shuiwang Ji, and Jieping Ye. Multi-task feature learning via efficient l 2, 1-norm minimization. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 339–348. AUAI Press, 2009. 2.2

[73] Gábor Lugosi, Omiros Papaspiliopoulos, and Gilles Stoltz. Online multi-task learning with hard constraints. *arXiv preprint arXiv:0902.3526*, 2009. 1, 1.1.1, 2.1, 2.1.1, 3.1.1, 4.3

[74] Andreas Maurer, Massimiliano Pontil, and Bernardino Romera-Paredes. Sparse coding for multitask and transfer learning. In *ICML (2)*, pages 343–351, 2013. 4.1.2, 4.2.1

[75] Tom M Mitchell, William W Cohen, Estevam R Hruschka Jr, Partha Pratim Talukdar, Justin Betteridge, Andrew Carlson, Bhavana Dalvi Mishra, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, et al. Never ending learning. In *AAAI*, pages 2302–2310, 2015. 4

[76] Keerthiram Murugesan and Jaime Carbonell. Active learning from peers. In *Advances in Neural Information Processing Systems*, pages 0–0, 2017. 2

[77] Keerthiram Murugesan and Jaime Carbonell. Multi-task multiple kernel relationship learning. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 687–695. SIAM, 2017. 1, 2

[78] Keerthiram Murugesan and Jaime Carbonell. Self-paced multitask learning with shared knowledge. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2522–2528, 2017. doi: 10.24963/ijcai.2017/351. URL https://doi.org/10.24963/ijcai.2017/351. 3

[79] Keerthiram Murugesan, Hanxiao Liu, Jaime Carbonell, and Yiming Yang. Adaptive smoothed online multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4296–4304, 2016. 1, 2, 3.1.1, 3.2, 17, 4, 4.1.2, 4.3, 4.3.5

[80] Keerthiram Murugesan, Jaime Carbonell, and Yiming Yang. Co-clustering for multitask learning. *arXiv preprint arXiv:1703.00994*, 2018. 4

[81] A-S Nemirovsky, D-B Yudin, and E-R Dawson. Problem complexity and method efficiency in optimization. 1982. 2.1.4

[82] Francesco Orabona and Nicolo Cesa-Bianchi. Better algorithms for selective sampling. In *Proceedings of the 28th international conference on Machine learning (ICML-11)*, pages 433–440, 2011. 3, 3.1.1

[83] Francesco Orabona, Joseph Keshet, and Barbara Caputo. The projectron: a bounded kernel-based perceptron. In *Proceedings of the 25th international conference on Machine learning*, pages 720–727. ACM, 2008. 4.3.3, 4.3.4, 4.3.5

[84] Anastasia Pentina and Shai Ben-David. Multi-task and lifelong learning of kernels. In *International Conference on Algorithmic Learning Theory*, pages 194–208. Springer, 2015. 4

[85] Anastasia Pentina and Ruth Urner. Lifelong learning with weighted majority votes. In *Advances in Neural Information Processing Systems*, pages 3612–3620, 2016. 4

[86] Anastasia Pentina, Viktoriia Sharmanska, and Christoph H Lampert. Curriculum learning of multiple tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5492–5500, 2015. 4.1.4, 4.1.4

[87] Rajat Raina, Andrew Y Ng, and Daphne Koller. Constructing informative priors using transfer learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 713–720. ACM, 2006. 4.2.3

[88] Alain Rakotomamonjy, Francis Bach, Stéphane Canu, and Yves Grandvalet. Simplemkl. *Journal of Machine Learning Research*, 9:2491–2521, 2008. 2.2

[89] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H Lampert. icarl: Incremental classifier and representation learning. *arXiv preprint arXiv:1611.07725*, 2016. 4

[90] Alessandro Rinaldo et al. Properties and refinements of the fused lasso. *The Annals of Statistics*, 37(5B):2922–2952, 2009. C.1.1

[91] Bernardino Romera-Paredes, Andreas Argyriou, Nadia Berthouze, and Massimiliano Pontil. Exploiting unrelated tasks in multi-task learning. In *AISTATS*, volume 22, pages 951–959, 2012. 4.1.2

[92] Adam J Rothman, Elizaveta Levina, and Ji Zhu. Sparse multivariate regression with covariance estimation. *Journal of Computational and Graphical Statistics*, 19(4):947–962, 2010. 2.2, 2.2.7

[93] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 4

[94] Paul Ruvolo and Eric Eaton. Active task selection for lifelong machine learning. In *AAAI*, 2013. 4.1.4

[95] Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. *International Conference on Machine Learning*, 28:507–515, 2013. 4, 4.1.4, 4.1.4

[96] Avishek Saha, Piyush Rai, Suresh Venkatasubramanian, and Hal Daume. Online learning of multiple tasks and their relationships. In *International Conference on Artificial Intelligence and Statistics*, pages 643–651, 2011. 1, 1.1.1, 2.1, 2.1.1, 2.1.6, 2.1.6, 2.2.6, 3.1.1, 3.2, 4, 4.3, 4.3.5

[97] Shai Shalev-Shwartz and Yoram Singer. Online learning: Theory, algorithms, and applications. *PhD Dissertation*, 2007. 2.1.5

[98] Daniel L Silver, Qiang Yang, and Lianghao Li. Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*, volume 13, page 05, 2013. 4

[99] Vikas Sindhwani, Jianying Hu, and Aleksandra Mojsilovic. Regularized co-clustering with dual supervision. In *Advances in Neural Information Processing Systems*, pages 1505–1512, 2009. 4.2

[100] Vikas Sindhwani, Ha Quang Minh, and Aurélie C Lozano. Scalable matrix-valued kernel learning for high-dimensional nonlinear multivariate regression and granger causality. In

Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, pages 586–595. AUAI Press, 2013. 4, 4.2

[101] Vikas Sindhwani, Minh Ha Quang, and Aurélie C Lozano. Scalable matrix-valued kernel learning for high-dimensional nonlinear multivariate regression and granger causality. In *Proceedings of the twenty-ninth conference on uncertainty in artificial intelligence*. AUAI Press, 2013. 2.2.7

[102] Alex J Smola and Bernhard Schölkopf. *Learning with kernels*. GMD-Forschungszentrum Informationstechnik, 1998. 4.3.1

[103] Zhaonan Sun, Nawanol Ampornpunt, Manik Varma, and Svn Vishwanathan. Multiple kernel learning and the smo algorithm. In *Advances in neural information processing systems*, pages 2361–2369, 2010. 2.2

[104] Grzegorz Swirszcz and Aurelie C Lozano. Multi-level lasso for sparse multi-task regression. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 361–368, 2012. 2.2, 2.2.1

[105] Lei Tang, Jianhui Chen, and Jieping Ye. On multiple kernel learning with multiple labels. In *IJCAI*, pages 1255–1260, 2009. 2.2.7

[106] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646, 1996. 1, 4

[107] Sebastian Thrun and Tom M Mitchell. Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46, 1995. 4

[108] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. C.1, C.1.2

[109] Ruth Urner, Shai Ben-David, and Ohad Shamir. Learning from weak teachers. In *AISTATS*, pages 4–3, 2012. 3.1.1

[110] Jean-Philippe Vert and Kevin Bleakley. Fast detection of multiple change-points shared by many signals using group lars. In *Advances in Neural Information Processing Systems*, pages 2343–2351, 2010. C.1

[111] Hua Wang, Feiping Nie, Heng Huang, and Fillia Makedon. Fast nonnegative matrix tri-factorization for large-scale data co-clustering. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, page 1553, 2011. 4.2

[112] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009. 2.1.1, 2.2.5, 3.1.1, 3.2, 3.4, 4, 4.1.2, 4.2

[113] Linli Xu, Aiqing Huang, Jianhui Chen, and Enhong Chen. Exploiting task-feature co-clusters in multi-task learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1931–1937. AAAI Press, 2015. 4.2

[114] Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-task learning for classification with dirichlet process priors. *The Journal of Machine Learning Research*, 8: 35–63, 2007. 1, 1.1.1, 2.1, 2.1.6, 2.2, 3

[115] Yan Yan, Glenn M Fung, Rómer Rosales, and Jennifer G Dy. Active learning from crowds. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1161–1168, 2011. 3.1.1

[116] Chicheng Zhang and Kamalika Chaudhuri. Active learning from weak and strong labelers. In *Advances in Neural Information Processing Systems*, pages 703–711, 2015. 3.1.1

[117] Jian Zhang, Zoubin Ghahramani, and Yiming Yang. Learning multiple related tasks using latent independent component analysis. In *Advances in neural information processing systems*, pages 1585–1592, 2005. 4.2

[118] Jintao Zhang and Jun Huan. Inductive multi-task learning with multiple view data. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 543–551. ACM, 2012. 2.2

[119] Yi Zhang and Jeff G Schneider. Learning multiple tasks with a sparse matrix-normal penalty. In *Advances in Neural Information Processing Systems*, pages 2550–2558, 2010. 2.2, 2.2.5, 2.2.7, 4.2.2, 5

[120] Yu Zhang and Dit-Yan Yeung. A regularization approach to learning task relationships in multitask learning. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(3): 12, 2014. 1, 1.1.1, 2.1, 2.2, 2.2.4, 2.2.7, 2.2.7, 2.2.7, 3, 4, 4.2.2, 4.2.2, 4.2.3

[121] Wenliang Zhong and James T Kwok. Convex multitask learning with flexible task clusters. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 49–56, 2012. 4.2