Graphical Multi-Task Learning
(Manuscript)

Daniel Sheldon

June 1, 2010

Abstract

We investigate multi-task learning in a setting where relationships between tasks are modeled by a graph structure. Most existing methods treat all pairs of tasks as being equally related, which can be hurt performance when the true structure of task relationships is more complex. Our method uses regularization to encourage models for task pairs to be similar whenever they are connected in the network. This idea was first proposed by Evgeniou, Micchelli and Pontil, who pointed out that the multi-task problem can be reduced to a single-task learning problem with a modified kernel constructed from the graph Laplacian. We extend their work to fully incorporate non-linear kernels and make a practical modification to the kernel that performs better in practice. We also present the first empirical evaluation of graphical multi-task learning on one synthetic and two real applications to demonstrate its value over methods that treat tasks relationships symmetrically.

1 Introduction

It is well established that *multi-task learning* — solving multiple related learning tasks simultaneously — often outperforms methods that treat each task in isolation. Task similarity is leveraged by allowing the examples from one task to influence the training of other tasks: for example, in a hierarchical Bayes framework, tasks share hyperparameters [1, 2]; in neural networks, tasks share hidden layers, allowing joint learning of the feature space [3]; in kernel methods, tasks are coupled by regularization [4].

Most previous work on multi-task learning treats all pairs of tasks as being equally related. However, a modeler often has *a priori* knowledge of asymmetries in task relationships, and would like to use this knowledge to inform the learning process. In this work, we present a technique called graphical multi-task learning (GMTL) in which task relationships are described by a *task network*. There are many real scenarios when such a network is available, and we describe two in our experiments: in a species distribution application, the network represents spatial or temporal adjacency between tasks; in an application to predict the votes of US senators on legislative bills, the network is a nearest neighbor network derived from past voting similarity.

GMTL uses graph-based regularization to encourage the learned models for two tasks to be similar whenever they are neighbors in the task network. This idea was first proposed by Evgeniou, Micchelli and Pontil in their work on multi-task kernels [4, 5]. They observed that multi-task learning with graph-based regularization can be reduced to *single-task* learning by introducing a multi-task kernel that captures both task similarity and the relationship between feature vectors. The multi-task kernel in this case is a *product kernel*, meaning that kernel evaluations are the product between evaluations of a discrete task kernel (constructed from the graph Laplacian of the task network) and a linear kernel on the input space. One of our main contributions it to extend this idea to non-linear kernels using facts products kernels that we develop in Section 3.1.

The other primary contribution is a practical development of graph-based regularization. We make an important modification to the task kernel proposed by Evgeniou, Micchelli and Pontil to overcome a problem of "translation invariance" in a robust way (see Section 3.2). With this modification, GMTL has a natural

task-coupling parameter μ such that the two extremes $\mu = 0$ and $\mu \to \infty$ correspond to different single-task approaches: one that treats all tasks separately, and one that learns a single model for all tasks. We also present the first empirical evaluation of GMTL, exploring a number of issues on synthetic data and demonstrating the value of GMTL on two real applications.

1.1 Related Work

Our work builds on ideas in the sequence of papers by Evgeniou, Micchelli and Pontil about regularization and kernels for multi-task learning [4–6]. Kato et al. also proposed a similar method for multi-task learning with a network of tasks [7]; their graph-based regularization is very similar to GMTL except that it penalizes the maximum of a number of terms (one for each edge) instead of the sum. They present a support vector machine (SVM) formulation that can be solved by second order cone programming. A significant advantage of GMTL is that it can be incorporated in any kernelized learning method by the kernel trick without custom-purpose algorithms.

As with previous kernel methods for multi-task learning, we adopt a slightly restricted view where the functions to be learned all have the same output space and there are relationships among the outputs themselves. Applications commonly comprise many related instances of the same learning task, e.g., predicting responses of different individuals on the same survey. The more general body of work on multi-task learning includes problems where the tasks have different output spaces but may still benefit from shared representation of the input space [3]. We discuss product kernels at some length in Section 3.1. These arise frequently in multi-task learning: see [8–10], and nearly all examples from [4]. The collaborative filtering method in [11] also uses product kernels and can be interpreted as multi-task learning.

2 The Formulation

We assume there are T tasks numbered $1, 2, \ldots, T$, each with the same input domain $\mathcal{X}$ and output domain $\mathcal{Y}$. Examples for task t are drawn from distribution $\mathcal{P}_t$ on $\mathcal{X} \times \mathcal{Y}$. A total of m training examples $\{(\mathbf{x}_i, y_i, t_i)\}_{i=1}^m$ are available, where $(\mathbf{x}_i, y_i)$ is an independent sample from $\mathcal{P}_{t_i}$ for all i. The goal is to learn functions f_t such that $f_t(x) \approx y$ for future samples $(\mathbf{x}, y)$ drawn from $\mathcal{P}_t$. Let $\mathcal{I}_t = \{i : t_i = t\}$ be the set of indices corresponding to training examples for task t, and let $\ell(a, y)$ be a loss function that measures the loss incurred when a is predicted and the true label is y.

Let $\mathcal{T} = \{1, \ldots, T\}$ be the set of tasks. We assume that an undirected graph $G = (\mathcal{T}, E)$ called the *task network* is given, and that tasks that are directly connected in G are believed *a priori* to be similar to each other. We incorporate this belief into the learning problem through regularization. We propose the following multi-task formulation to couple the learning of the tasks:

$$\min_{f_1,\ldots,f_T \in \mathcal{H}} R(f_1, \ldots, f_T) \quad = \quad \mu \sum_{(s,t)\in E} \|f_s - f_t\|_{\mathcal{H}}^2 \quad + \quad \lambda \sum_t \|f_t\|_{\mathcal{H}}^2 \quad + \quad \sum_t L_t(f_t). \quad (\mathrm{OP}_1)$$

The task-specific functions $f_1, \ldots, f_T$ are all selected from $\mathcal{H}$, a reproducing kernel Hilbert space (RKHS). The term $\sum_{(s,t)\in E} \|f_s - f_t\|_{\mathcal{H}}^2$ is an aggregate measure of the dissimilarity between pairs of functions corresponding to edges in the task network — this penalty couples the learning of individual tasks by encouraging f_s and f_t to be similar whenever s and t are neighbors in G. The second and third terms of $R(f_1, \ldots, f_T)$ are just sums of terms from the standard single-task formulations: the penalty $\|f_t\|_{\mathcal{H}}^2$ controls the complexity of f_t, and $L_t(f_t) = \sum_{i\in\mathcal{I}_t} \ell(f_t(\mathbf{x}_i), y_i)$ measures the total loss incurred when f_t is used to make predictions on the training examples for task t. As in [6], the parameter μ controls coupling of the tasks. When $\mu = 0$, the problem decomposes and all tasks are learned independently. As $\mu \to \infty$, then OP_1 becomes a single-task problem where the same function is used for all tasks (if G is connected; we elaborate on this in Section 4).

The Kernel Trick. An important feature of OP_1 is that the problem of minimizing $R(f_1, \ldots, f_T)$ can be reduced to single-task learning by the kernel trick. Rather than learning separate functions for each task, we will instead learn a single function $f : \mathcal{X} \times \mathcal{T} \to \mathcal{Y}$ that takes both the feature vector and the task

identifier as arguments, and predicts $f(\mathbf{x}, t)$ on input $\mathbf{x}$ for the tth task. The challenge is to construct the appropriate kernel $\hat{k}$ on the new input space $\mathcal{X} \times \mathcal{T}$ such that the resulting single-task learning problem is indeed equivalent to OP_1. To preview the upcoming sections, $\hat{k}$ will be a product kernel between a discrete task kernel derived from the graph Laplacian, and the base kernel k for the RKHS $\mathcal{H}$, called the *graphical multi-task kernel*.

3 Reduction to Single-Task Learning

The idea of the reduction is as follows. Suppose a kernel $\hat{k}$ on the expanded input space $\mathcal{X} \times \mathcal{T}$ is chosen, and let $\hat{\mathcal{H}}$ be the corresponding RKHS. We call $f \in \hat{\mathcal{H}}$ a *multi-task function*. For any task t, we can recover a task-specific function $f(\cdot, t)$ as the function that takes value $f(\mathbf{x}, t)$ on input $\mathbf{x}$. We would like to match the functions $f(\cdot, t)$ with the task-specific functions f_t in OP_1. However, it is not even clear *a priori* that $f(\cdot, t)$ belongs to $\mathcal{H}$. The goal is to construct $\hat{k}$ such that: (1) the space of functions $\{f(\cdot, 1), \ldots, f(\cdot, T) : f \in \hat{\mathcal{H}}\}$ is equivalent to the space $\{f_1, \ldots, f_T \in \mathcal{H}\}$, and (2) regularization of f in $\hat{\mathcal{H}}$ is equivalent to the regularization terms in OP_1.

3.1 Product Kernels

In this section we establish facts about product kernels that will be used to construct the graphical multi-task kernel. We consider product kernels $\hat{k}$ of the form

$$\hat{k}\left((\mathbf{x}, s), (\mathbf{z}, t)\right) = K_{st} \cdot k(\mathbf{x}, \mathbf{z}), \tag{1}$$

where K is a valid kernel on the discrete space $\mathcal{T}$ (i.e., K is a $T \times T$ positive semidefinite matrix) and k is a valid kernel on $\mathcal{X}$. Such a product is always a valid kernel on $\mathcal{X} \times \mathcal{T}$ [12, p. 95]. We call K the *task kernel* and k the *base kernel*. We assume henceforth that the base kernel k corresponds to the RKHS $\mathcal{H}$ in OP_1. By standard properties of a RKHS, every function $f \in \hat{\mathcal{H}}$ can be written in the form $f(\mathbf{x}, t) = \sum_{i=1}^{n} \alpha_i \hat{k}\left((\mathbf{x}, t), (\mathbf{x}_i, t_i)\right)$ where $(\mathbf{x}_i, t_i) \in \mathcal{X} \times \mathcal{T}$ for $i = 1, \ldots, n$. Substituting (1) into this expression, we have for product kernels that

$$f(\mathbf{x}, t) = \sum_{i=1}^{n} \alpha_i K_{tt_i} k(\mathbf{x}, \mathbf{x}_i). \tag{2}$$

Then, for a particular task t, we can define constants $\beta_i^{(t)} = \alpha_i K_{tt_i}$, and write $f(\mathbf{x}, t) = \sum_i \beta_i^{(t)} k(\mathbf{x}, \mathbf{x}_i)$, from which it is clear that $f(\cdot, t)$ belongs to $\mathcal{H}$. Hence the functions $f(\cdot, t)$ are comparable to the functions f_t from OP_1. In particular, we can compute inner products and distances between task-specific functions $f(\cdot, s)$ and $f(\cdot, t)$ using the inner product in $\mathcal{H}$. This leads to the first lemma, which states that regularization of f in $\hat{\mathcal{H}}$ has a simple interpretation in terms of the task-specific functions and the task kernel.

Lemma 1. *Let K^- be any matrix such that $KK^-K = K$. Then*

$$\|f\|_{\hat{\mathcal{H}}}^2 = \sum_{s,t \in \mathcal{T}} K_{st}^- \langle f(\cdot, s), f(\cdot, t) \rangle_{\mathcal{H}}. \tag{3}$$

If K is non-singular, the only matrix that satisfies the conditions of the lemma is K^{-1}. It is also important to understand which combinations of task-specific functions can be obtained from some $f \in \hat{\mathcal{H}}$. Grouping the terms in (2) by task and defining $g_t(\mathbf{x}) = \sum_{i:t_i=t} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$, we have

$$f(\mathbf{x}, t) \;=\; \sum_u K_{tu} \sum_{i:t_i=u} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \;=\; \sum_u K_{tu} \cdot g_u(\mathbf{x}). \tag{4}$$

Equation (4) can be viewed as a matrix-vector multiplication for vector-valued functions [5]. To elaborate, we introduce some notation: for functions $h_1, \ldots, h_T \in \mathcal{H}$, write $\bar{h} = (h_1, \ldots, h_T)' \in \mathcal{H}^T$ to mean that $\bar{h}$ is

the vector-valued function $\bar{h}(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_T(\mathbf{x}))'$. For a vector-valued function $\bar{h}$ and a matrix A of the appropriate size, define $A\bar{h}$ to be the vector-valued function such that $(A\bar{h})(\mathbf{x}) = A \cdot \bar{h}(\mathbf{x})$ for all $\mathbf{x}$. From (4), we derive the following lemma that explains what combinations of task-specific functions can be obtained from some multi-task function.

Lemma 2. *Let $f_1, \ldots, f_T \in \mathcal{H}$. There exists $f \in \hat{\mathcal{H}}$ such that $f(\cdot, t) = f_t$ for all t if and only if $(f_1, \ldots, f_T)' = K \cdot (g_1, \ldots, g_T)'$ for some $g_1, \ldots, g_T \in \mathcal{H}$.*

In other words, the vectors of task-specific functions obtainable from some $f \in \hat{\mathcal{H}}$ are those that lie in the "column space" $\{K\bar{g} : \bar{g} \in \mathcal{H}^T\}$ of K. It will be more convenient to work with an equivalent statement in terms of the null space of K.

Lemma 3. *Let $f_1, \ldots, f_T \in \mathcal{H}$. There exists $f \in \hat{\mathcal{H}}$ such that $f(\cdot, t) = f_t$ for all t if and only if $\sum_t v_t f_t = 0$ for all $v \in \text{null}(K)$.*

Lemma 3 provides a useful way to think about the space $\{f(\cdot, 1), \ldots, f(\cdot, T) : f \in \hat{\mathcal{H}}\}$. It is equal to the space $\{f_1, \ldots, f_T \in \mathcal{H}\}$, subject to the additional linear constraints that $\sum_t v_t f_t = 0$ for v in the null space of K. If K is non-singular, then the two spaces are equal.

3.2 The Graphical Multi-Task Kernel

For GMTL, we construct the task kernel from the graph Laplacian of the task network. Let $\deg(t)$ be the degree of task t in the G. The graph Laplacian of G is the matrix $L = D - A$, where D is a diagonal matrix with entries $D_{tt} = \deg(t)$ and A is the adjacency matrix of G.

Theorem 1. *Define the task kernel to be $K = (\mu L + \lambda I)^{-1}$ in the product kernel $\hat{k}$. Then the following single-task learning problem is equivalent to* OP_1

$$\min_{f \in \hat{\mathcal{H}}} R(f) = \|f\|^2_{\hat{\mathcal{H}}} + \sum_t L_t(f(\cdot, t)). \qquad (\text{OP}_2)$$

Proof. The Laplacian is always singular and positive semidefinite [13]. If $\lambda > 0$, then $\mu L + \lambda I$ is non-singular, so the inverse K exists and by Lemma 3, the feasible sets of OP_1 and OP_2 are equivalent by equating $f(\cdot, t)$ and f_t. The loss terms in OP_1 and OP_2 are identical. By Lemma 1, the regularization terms are also identical:

$$\|f\|^2_{\hat{\mathcal{H}}} \quad = \quad \sum_{s,t} (\mu L + \lambda I)_{st} \langle f(\cdot, s), f(\cdot, t) \rangle_{\mathcal{H}} \quad = \quad \mu \sum_{(s,t) \in E} \|f(\cdot, s) - f(\cdot, t)\|^2 + \lambda \sum_t \|f(\cdot, t)\|^2.$$

The last equality is straightforward to verify by rearranging the sum over the entries of L — it is analogous to a well-known property of the graph Laplacian [13]. □

Translation Invariance. Evgeniou, Micchelli and Pontil originally proposed graph-based regularization without any task-specific regularization [4], i.e., with $\lambda = 0$ in our terminology. They pointed out that OP_1 has a serious flaw in this case. The regularization term $\sum_{(s,t) \in E} \|f_s - f_t\|^2_{\mathcal{H}}$ is translation invariant in the sense that one can add any function f_0 to all of the task-specific functions and it does not change, since it only involves differences between functions. This is undesirable: the optimization may set all task-specific functions equal to a single function f_0 that is very complex, which risks badly overfitting the training sample even though the regularization term is equal to zero.

The authors propose to fix this problem by instead solving OP_2 with the task kernel $K = L^+$, the Moore-Penrose pseudoinverse of the Laplacian. With this choice of K, the regularization terms of OP_1 and OP_2 are still equivalent (with $\lambda = 0$) by Lemma 1. However, the task kernel is now singular, so the feasible set of OP_2 is subject to the additional constraints that $\sum_t v_t f_t = 0$ for all $v \in \text{null}(L^+)$. By properties of the graph Laplacian, this solution boils down to requiring that $\sum_{t \in C} f_t = 0$ for all connected components C of

G,[1] so that the task-specific functions are centered at the origin. This overcomes the problem of translation invariance, but it is also undesirable: for example, it is impossible for tasks to agree on the sign of the prediction for any given x. For any task s that predicts $f_s(x) > 0$, some task t must predict $f_t(x) < 0$. Moreover, within each component, these values must be balanced so they sum to zero.

A much better solution to translation invariance is to incorporate task-specific regularization by setting λ to be nonzero. Rather than a hard constraint that the task-specific functions be centered at the origin, this is an elastic constraint (as in typical regularized learning) that prefers functions that are closer to the origin. This is especially important when the tasks substantially agree on many training examples, so the method is not forced arbitrarily to center the predictions at zero.

A word on the linear case. Several ideas in this section — in particular, Lemma 1 and Theorem 1 — appeared in [4] in a form restricted to linear base kernels. The more general case here is analogous to the linear version, but does not follow from the arguments made in that case. For linear kernels, a multi-task feature expansion $\phi : \mathcal{X} \times \mathcal{T} \to \mathbf{R}^{Td}$ is defined, where d is the dimension of the feature expansion for the space $\mathcal{X}$, and linear algebra is used to establish the results. When no finite-dimensional feature expansion exists, we must work with the generic expansion of f and establish Lemma 3 to compare the feasible sets of OP_1 and OP_2.

4 Evaluation

We will evaluate GMTL both on synthetic data and on two real applications. However, we first introduce a tool called *multi-task curves* to compare the performance of different task networks. A multi-task curve plots performance as a function of μ, the task-coupling parameter. As μ varies from 0 to infinity the problem varies smoothly between two different single-task approaches.[2] When $\mu = 0$, the problem decomposes by task and each is learned independently. We call this the SEPARATE method. When $\mu \to \infty$, the formulation learns a single function for all of the tasks in each connected component of G, a statement we can verify by examining the task kernel as $\mu \to \infty$.

Lemma 4. *Fix λ, and let $K(\mu) = (\mu L + \lambda I)^{-1}$. As $\mu \to \infty$, the matrix $K(\mu) \to K$, a block diagonal matrix with entries K_{st} that are equal to $(\lambda|C|)^{-1}$ whenever s and t are both members of the same connected component C, and zero otherwise.*

Hence, for s and t in different connected components, all kernel evaluations are zero, and for $s, t \in C$, the multi-task kernel is $\hat{k}((s,x),(t,y)) = k(x,y)/(\lambda|C|)$, a scaled version of the base kernel. This means that the learning problem decomposes by connected component, and for the ith component C_i it is a single-task learning problem with regularization parameter $\lambda_i = \lambda|C_i|$.

We are most interested in the case when G is connected, so POOLED learns a single function for all tasks. Then, for all connected task networks, the multi-task curves coincide at both endpoints and we can visually assess the relative performance at intermediate values for μ. Any curve that is peaked in the middle indicates that multi-task learning is advantageous over the single-task alternatives. Figure 3 shows several examples of multi-task curves taken from our applications. We will often use the clique as a baseline to demonstrate the value of GMTL. With the clique as task network, GMTL is equivalent to the symmetric regularized multi-task learning method in [6].

4.1 Synthetic Data

We design a synthetic data set where we have full control over a number of factors that are important to multi-task learning: (1) the number of tasks, (2) the strength of task relationships, (3) the structure of task relationships and (4) the amount of training data. The tasks are classification tasks in the two-dimensional

[1] The null space of L^+ is the same as that of L, and it consists of all vectors v that are constant on each connected component C of the task network [13]. The indicator vectors with entries $I(t \in C)$ form a basis. To be orthogonal to the indicator vector for component C means $\sum_t I(t \in C) f_t = \sum_{t \in C} f_t = 0$.

[2] This is another advantage of task-specific regularization over the Pseudoinverse task kernel, for which μ behaves differently.

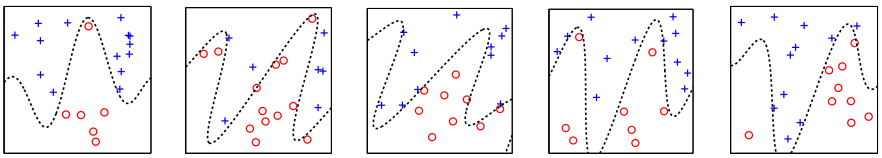

Figure 1: Example tasks for $T = 5, \tau^2 = 1, m = 20$.

plane with non-linear decision boundaries. See Figure 1 for examples. We control task similarity through the parameters — the true decision boundaries vary smoothly with the parameters so that nearby parameter settings yield similar classifiers. However, the parameters are meant to be arbitrary and have no direct relationship to the hypothesis space of the learner (we use an SVM with a Gaussian base kernel).

We first define a simple family of nonlinear functions consisting of the first few terms of an arbitrary Fourier series:

$$h(z; \mathbf{a}) = a_1 \sin(z - a_0) + a_2 \sin(2(z - a_0)) + a_3 \cos(z - a_0) + a_4 \cos(2(z - a_0))$$

The basic classification boundaries will be graphs of these functions in $\mathbf{R}^2$, giving classifiers $g(\mathbf{x}; \mathbf{a}) = \text{sign}(x_2 - h(x_1; \mathbf{a}))$. We also allow rotations of the decision boundaries. Let R_θ be the operator that rotates a vector by θ radians in a counterclockwise direction about the origin. The final family of classifiers has θ as an additional parameter: $f(\mathbf{x}; \mathbf{a}, \theta) = g(R_\theta \mathbf{x}; \mathbf{a})$.

Generating Tasks. Task parameters are generated via a random walk in parameter space with Gaussian increments. Initial values are $\mathbf{a}^{(1)} = (0, 1, 1, 1, 1)$ and $\theta^{(1)} = 0$. For $t = 2, \ldots, T$,

$$\mathbf{a}^{(t)} = \mathbf{a}^{(t-1)} + \epsilon_t, \quad \epsilon_t \sim N(0, \sigma^2 I); \qquad \theta^{(t)} = \theta^{(t-1)} + \delta_t, \quad \delta_t \sim N(0, \sigma^2(\pi/4)^2).$$

The parameter σ^2 controls step sizes and hence task similarity. To separate this effect from that of T, it is useful to reparameterize the step size as $\sigma^2 = \tau^2/T$, so that for a given τ^2, the distributions of $\mathbf{a}^{(T)}$ and $\theta^{(T)}$ do not depend on T. A training sample of size m is generated for each task by choosing m inputs $\mathbf{x}$ uniformly at random from the square $x_1, x_2 \in [-3, 3]$, and then labeling them according to $f(\mathbf{x}; \mathbf{a}^{(t)}, \theta^{(t)})$. Because of the linear structure of the random walk, the "correct" task network is a path on T vertices, and we will use this network in our first round of experiments.

When does multi-task learning help? Multi-task learning can hurt performance if tasks are not sufficiently similar [3]. We do not attempt a formal definition of task similarity, but see [14] for one possibility. However, We do stress that task similarity should be assessed on the example-generating distributions $\mathcal{P}_1, \ldots, \mathcal{P}_T$; however, to evaluate the utility of multi-task versus single-task learning, it is also important to consider the size of the training sample. For example, even if the tasks are very similar, it is better to treat tasks separately if one has access to unlimited samples.

The training curves in Figure 2 explore these issues. Each curve shows the performance of one method (GMTL, SEPARATE or POOLED) as a function of the number m of training examples per task. In each trial, a new problem is generated with $m = 100$, and a SVM is trained on increasingly-sized subsets of the total training data. All experiments set $\lambda = .1$ and $\gamma = .1$ in the RBF base kernel [15]. Each curve is the average of 50 trials. The graphs are shaded to indicate regimes where each method is best for different settings of τ^2 and T.

The results show that multi-task learning is most advantageous for small or medium sized training samples. In many cases, we see the following pattern: POOLED (i.e., $\mu = 0$) is best on very small training samples, GMTL ($\mu = 1$) is best for intermediate-sized samples, and SEPARATE ($\mu \to \infty$) is best for large samples. This is consistent with the interpretation of μ as a regularization or complexity parameter: as μ increases the models have greater capacity to model each task separately and hence can learn more accurate models, but require more training data to do so. As the number of tasks increases (top row), learning accelerates in the coupled methods (POOLED and GMTL) because the effective number of training examples

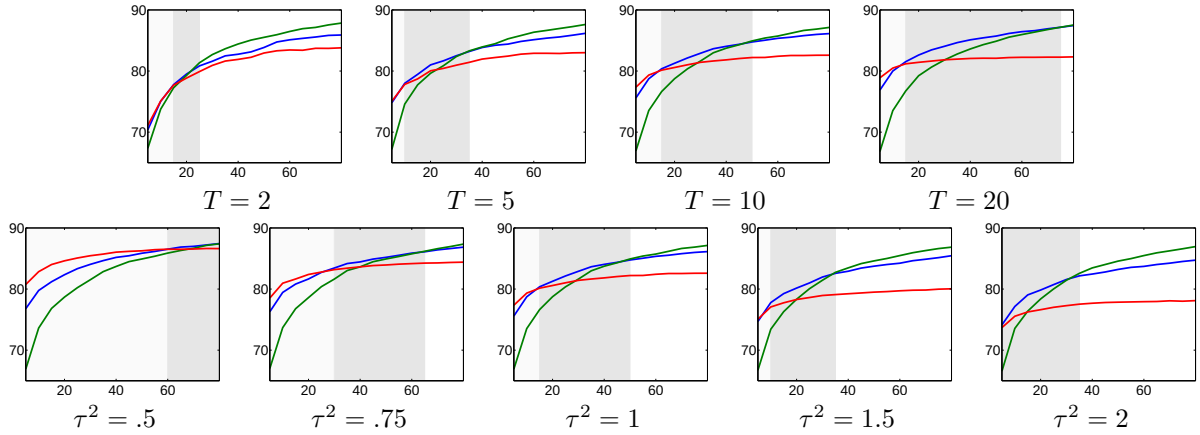

Figure 2: Training curves: accuracy (vertical axis) versus number of training examples (horizontal axis). Top row: effect of number of tasks for $\tau^2 = 1$. Bottom row: effect of task similarity for $T = 10$. Red lines indicate POOLED method, blue lines indicate GMTL, and green lines indicate SEPARATE. Light gray shading indicates region where POOLED method is best. Dark gray shading indicates region where GMTL is best.

per task is multiplied. This prolongs the regime in which they outperform SEPARATE. As tasks become less similar (bottom row), the coupled methods become more biased and perform worse; hence, the SEPARATE method overtakes them more quickly and the regime where GMTL is advantageous shifts to smaller training samples.

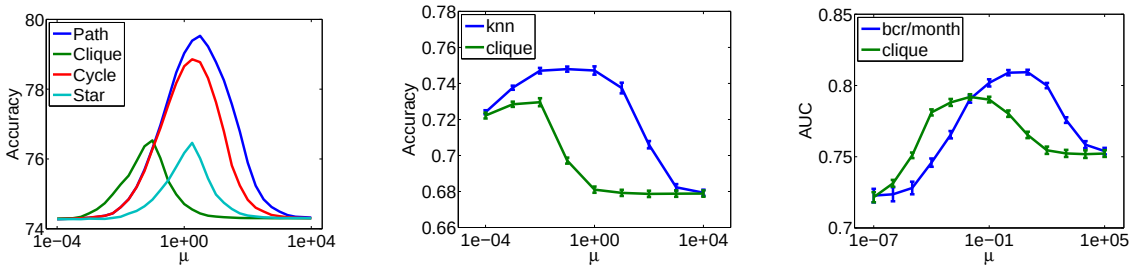

Figure 3: Multi-task curves. (a) Synthetic data ($\tau^2 = 2, T = 5$), (b) Senate data (SENATE-2K, $\lambda = 10^{-3}$), (c) Bird data (TRESWA, $\lambda = 10^{-4}$)

Structure Matters. How does the choice of task network affect the performance of GMTL? Figure 3a shows the multi-task curves for four different task networks. As expected, the path performs best because it matches the true task structure. The cycle is only slightly worse because it the only unnecessary link is the one between tasks 1 and T. The other networks — including the clique which implies symmetric multi-task learning — provide some benefit over single-task learning but are significantly worse than the path and cycle.

4.2 Applications

We also evaluate GMTL on two real applications: predicting votes in the US Senate, and predicting the presence or absence of bird species as reported by human observers. In each case we can derive a network that captures task relationships significantly better than symmetric multi-task learning.

Senate. The problem is to predict votes (aye or nay) of US senators on legislative bills. The data consists of all bills or resolutions that reached a final vote by roll call on the floor of the Senate during the years

1999–2008. Data was downloaded and compiled from the govtrack.us website. We exclude bills without at least a 10% minority vote to avoid a severe class imbalance and to restrict attention to "interesting" bills because most votes are unanimous or near-unanimous. This left 128 bills for approximately 12K training examples. Each senator is treated as a separate task; hence, a training example has the form $(\mathbf{x}_i, y_i, t_i)$ where t_i is the ID of senator, and y_i is the vote of senator t_i on the bill described by feature vector $\mathbf{x}_i$. The feature vector includes the bill title (as a TFIDF bag-of-words vector) and the party of the bill sponsor (Republican, Democrat, or other). For this application, we evaluate task similarity by comparing voting records on the training set: for each pair s, t of senators, we compute a similarity score as the fraction of bills on which they voted identically among all bills on which they both voted. We then construct the task network as a nearest-neighbor graph that connects each senator to the 3 most similar senators with undirected edges.

Birds. This data set consists of observations from eBird [16], a citizen science project where birdwatchers submit checklists of their observations online. Our data comprises 57500 checklists from the contiguous US during 2006. The problem is to predict whether the observer saw any birds of a target species given information about the trip such as date, time, location, effort, and terrain and habitat features. Our three target species — American Goldfinch, Red-winged Blackbird, and Tree Swallow — are all widespread within the US and migratory to some degree. The Tree Swallow migration is most significant with almost no overlap between summer and winter ranges [17]. Seasonal changes pose a significant challenge for species distribution models; to accommodate them we divide the problem into sub-tasks for different regions in space and time. Specifically, we treat each Bird Conservation Region (BCR) [18] in each month as a separate task. The task network G captures spatiotemporal connectivity: two tasks are neighbors if either (1) they belong to the same month and the BCRs are adjacent, or (2) they belong to the same BCR and the months are adjacent (December and January are also considered adjacent).

Experiments. We compared GMTL with the task networks described above versus several other methods on these data sets. CLIQUE is the symmetric method [6] implemented by using a clique as the task network. PSEUDO uses the same task network as GMTL, but constructs the task kernel from the pseudoinverse of the Laplacian (see Section 3.2). The SEPARATE and POOLED methods were described in Section 4.

From the bird data, we generate a problem for each target species: AMEGFI (American Goldfinch), REWBLA (Red-winged Blackbird) and TREWSA (Tree Swallow). In each trial, 5000 random checklists are chosen and split into train, validation and test sets in a 3/1/1 ratio. For the Senate data, first bills are split into train, validation and test sets in a 3/1/1 ratio, and then *examples* (i.e., votes by individual senators) are chosen randomly from the training bills to constitute the final training set. The SENATE-1K, SENATE-2K, SENATE-4K, and SENATE-ALL problems are named to indicate training set size, where SENATE-ALL contains all votes for the training bills. We adopt this design for two reasons: first, it demonstrates the utility of multi-task learning for small training sets. Second, this application is different from the others in that the set of labeled $\mathbf{x}$ values (the bills) is the same for all tasks. For task t, the label applied to input $\mathbf{x}$ by task s is of little additional value once the label for task t is known. Hence, multi-task learning provides little advantage when the training set includes all votes for a fixed set of training bills. With our design, the training set will include the votes of other senators on bills $\mathbf{x}'$ that are not labeled for senator t.

Ten random splits are performed for each problem, and we measure both accuracy and area under the ROC curve (AUC). The latter tends to be a more reliable performance metric for bird problems which have a significant class imbalance (most observations are negative) and noise (repeat observations often yield different results). Linear kernels are used for the senate experiments, and RBF kernels with $\gamma = .001$ for the bird experiments. The λ and μ parameters are selected on the validation set from $\lambda \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$ and $\mu \in \{10^{-3}, 10^{-1}, 1, 10^{1}, 10^{3}\}$ for the bird problems and $\lambda \in \{10^{-4}, \ldots, 10^{0}\}$ and $\mu \in \{10^{-4}, \ldots, 10^{1}\}$ for the senate problems. Parameters ranges were narrowed during preliminary experiments.

Results. Table 1 shows the results, with bold numbers indicating the best performance for each problem (or numbers where the difference from the maximum is not statistically significantly at the .05 level). An asterisk marks cases in which the maximum is unique and significant. The single-task methods almost never outperform multi-task methods — except for PSEUDO, which is competitive on the Senate data but performs poorly on the bird problems. In almost every case, GMTL is best or near-best. CLIQUE is second-best in

Table 1: Performance measurements (averaged over 10 trials).

	GMTL		Clique		Pseudo		Separate		Pooled	
	acc	auc	acc	auc	acc	auc	acc	auc	acc	auc
Senate-1k	**0.732**	**0.758**	0.707	0.705	**0.722**	**0.757**	0.703	0.698	0.685	0.598
Senate-2k	**0.751***	**0.781***	0.736	0.756	0.738	0.764	0.720	0.740	0.681	0.625
Senate-4k	**0.765**	**0.806***	**0.761**	0.795	0.746	0.789	0.753	0.784	0.688	0.640
Senate-All	0.774	**0.820**	**0.782***	**0.823**	0.750	0.801	0.770	**0.818**	0.693	0.639
Amegfi	**0.738**	**0.786***	**0.739**	0.780	0.685	0.707	0.677	0.688	**0.732**	0.779
Rewbla	**0.784***	**0.777***	0.772	0.759	0.763	0.715	0.738	0.687	0.757	0.727
Treswa	**0.908**	**0.800***	**0.909**	0.790	**0.909**	0.745	0.876	0.728	**0.909**	0.778

most cases, and sometimes competitive with GMTL, but often significantly worse. Figure 3, parts (b) and (c) show multi-task curves selected from these experiments to illustrate the advantage of GMTL over symmetric multi-task learning.

5 Conclusion

We explored a method for multi-task learning that exploits a network describing task relationships, and showed that it significantly outperforms symmetric multi-task learning methods in many cases. We described two real applications for which a task network is available, either through domain knowledge or heuristics to estimate task-similarity. A significant question for future research is whether one can learn the structure of task relationships in a general setting.

References

[1] P. Rossi and G. Allenby, "Bayesian Statistics and Marketing," *Marketing Science*, vol. 22, no. 3, pp. 304–328, 2003.

[2] K. Yu, V. Tresp, and A. Schwaighofer, "Learning Gaussian processes from multiple tasks," in *Proceedings of the 22nd international conference on Machine learning*, pp. 1012–1019, ACM New York, NY, USA, 2005.

[3] R. Caruana, "Multitask Learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997.

[4] T. Evgeniou, C. Micchelli, and M. Pontil, "Learning Multiple Tasks with Kernel Methods," *Journal of Machine Learning Research*, vol. 6, no. 1, p. 615, 2006.

[5] C. Micchelli and M. Pontil, "Kernels for multi-task learning," *Advances in Neural Information Processing Systems*, vol. 17, pp. 921–928, 2005.

[6] T. Evgeniou and M. Pontil, "Regularized multi–task learning," in *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 109–117, ACM, 2004.

[7] T. Kato, H. Kashima, M. Sugiyama, and K. Asai, "Multi-Task Learning via Conic Programming," *Advances in Neural Information Processing Systems*, 2007.

[8] K. Chai, C. Williams, S. Klanke, and S. Vijayakumar, "Multi-task Gaussian process learning of robot inverse dynamics," *Advances in Neural Information Processing Systems*, vol. 21, 2009.

[9] E. Bonilla, K. M. Chai, and C. Williams, "Multi-task gaussian process prediction," in *Advances in Neural Information Processing Systems 20* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), pp. 153–160, Cambridge, MA: MIT Press, 2008.

[10] E. Bonilla, F. Agakov, and C. Williams, "Kernel multi-task learning using task-specific features," *Proceedings of the 11th AISTATS*, 2007.

[11] J. Basilico and T. Hofmann, "Unifying collaborative and content-based filtering," in *Proceedings of the twenty-first international conference on Machine learning*, ACM New York, NY, USA, 2004.

[12] C. Rasmussen, *Gaussian processes for machine learning*. Springer, 2006.

[13] F. Chung, *Spectral graph theory*. American Mathematical Society, 1997.

[14] S. Ben-David and R. Schuller, "Exploiting task relatedness for multiple task learning," in *Proceedings of Computational Learning Theory (COLT*, 2003.

[15] T. Joachims, "Making large scale SVM learning practical," 1999.

[16] `http://ebird.org`.

[17] R. Robertson, B. J. Stutchbury, and R. R. Cohen, "Tree Swallow (Tachycineta bicolor)," in *The Birds of North America Online* (A. Poole, ed.), Cornell Lab of Ornithology, Ithaca, NY, 1992.

[18] `http://www.nabci-us.org/map.html`.