Graphical Multi-Task Learning

Daniel Sheldon DSHELDON@CS.CORNELL.EDU
Cornell University, Ithaca, NY

Abstract

We investigate the problem of learning multiple tasks that are related according to a network structure, using the multi-task kernel framework proposed in (Evgeniou et al., 2006). Our method combines a graphical task kernel with an arbitrary base kernel. We demonstrate its effectiveness on a real ecological application that inspired this work.

1. Introduction

It is well established that learning multiple related tasks together performs better than learning them independently. 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 (Rossi & Allenby, 2003; Yu et al., 2005); in neural networks, tasks share hidden layers, allowing joint learning of the feature space (Caruana, 1997).

Most previous work does not make any *a priori* distinction among relationships; tasks simply come from a pool of related tasks.[1] However, one often has important structural information. For example, consider a set of tasks predicting preferences for different users in a social network; it is natural to employ the network structure rather than treating this as a pool of tasks. This work was inspired by a problem in species distribution modeling: we are predicting the presence or or absence of a species of migratory bird at locations within its summer range. Because of the seasonal pattern of migration and life history, it is logical to treat this as a sequence of monthly tasks arranged in a cycle (December is adjacent to January).

[1]Some methods make *a posteriori* structural distinctions about task relationships, e.g., the task clustering method of (Xue et al., 2007).

Preliminary work.

To our knowledge, two previous works have proposed multi-task learning with networks of tasks. In (Evgeniou et al., 2006), the authors suggest a multi-task kernel that combines two kernels: (1) a task kernel built from the graph Laplacian of the task network and (2) a base kernel on the input features. However, the authors do not evaluate the technique. More recently, (Kato et al., 2007) proposed a formulation for multi-task SVMs using second order cone programming.

Our work builds on the framework of (Evgeniou et al., 2006). Our main contribution is a practical development of multi-task kernels using the graph Laplacian, including:

- a focus on non-linear kernels, facilitated by a conceptual simplification that works directly in a reproducing kernel Hilbert space (RKHS)
- important practical considerations that are essential for applications
- a demonstration of the value of these techniques for a real ecological application

2. Multi-Task Kernels

This section develops the basic multi-task kernel framework. Our presentation differs from (Evgeniou et al., 2006) by working directly in a RKHS instead of using feature expansions. This allows a simple unified presentation of linear and non-linear multi-task kernels.

Imagine a set $\mathcal{T}$ of T related learning tasks with the same input domain $\mathcal{X}$ and output domain $\mathcal{Y}$; examples for task t are drawn from distribution $\mathcal{P}_t$ on $\mathcal{X} \times \mathcal{Y}$. Given training examples for all tasks, the goal is to learn task-specific functions f_t such that $f_t(x)$ is a good predictor of y for future samples (x, y) drawn from $\mathcal{P}_t$. There are two natural approaches to try first:

1. Learn each task separately — this may work well but is limited if the total number of training examples is small,

2. Pool all the training examples and treat this as a single task, perhaps throwing in the task identifier as a feature to let your learner sort it out.

Multi-task kernels can be seen as a version of the second approach, with special attention paid to task representation. Following this idea, imagine using all the data to learn a single function f that takes both x and t as inputs, and simply using $f(x,t)$ in place of the task-specific function $f_t(x)$. To employ a kernel method, since our input space is $\mathcal{X} \times \mathcal{T}$, we must supply a kernel $\tilde{k} : (\mathcal{X} \times \mathcal{T}) \times (\mathcal{X} \times \mathcal{T}) \to \mathrm{R}$ on this joint space. The kernel $\tilde{k}$ is the multi-task kernel.

Given the multi-task kernel, candidate functions for the learner are those functions in the RKHS $\tilde{\mathcal{H}}$ with kernel $\tilde{k}$, i.e., functions f of the form

$$f(x,t) = \sum_{i=1}^{n} \alpha_i \tilde{k}((x,t),(x_i,t_i)), \quad (x_i,t_i) \in \mathcal{X} \times \mathcal{T}. \quad (1)$$

We are specifically interested in regularized learning in $\tilde{\mathcal{H}}$, that is, learners that optimizing a tradeoff between the norm of f and its empirical risk:

$$\hat{f} = \arg\min_{f} \frac{1}{2}\|f\|^2_{\tilde{\mathcal{H}}} + \frac{C}{M}\sum_{m=1}^{M} \ell(f(x_m,t_m),y_m)$$

Here, m ranges over the M training examples, and $\ell(a,b)$ is the loss incurred when we predict a and the true value is b. In the next section we'll see that regularization has a special interpretation for certain multi-task kernels.

2.1. Product Multi-Task Kernels

All examples of multi-task kernels given in (Evgeniou et al., 2006), with the exception of Section 5.1, have a particularly simple form — they are products of a *task kernel* $k_{\mathcal{T}}$ and a *base kernel* $k_{\mathcal{X}}$:

$$\tilde{k}((x,s),(y,t)) = k_{\mathcal{T}}(s,t)k_{\mathcal{X}}(x,y).$$

Because $\mathcal{T}$ is finite, we can use a matrix K in place of $k_{\mathcal{T}}$. We also drop subscripts when there is no ambiguity, and write the product kernel as

$$\tilde{k}((x,s),(y,t)) = K_{st}k(x,y). \quad (2)$$

Any such product is a valid kernel if K is positive semidefinite and k is a valid kernel (e.g., see (Rasmussen, 2006) p. 95). We advocate multi-task kernels of this form due to their simplicity, and because in this case, as we will show below, regularization of f is easily interpreted in terms of the task-specific functions.

Let us re-examine the task-specific function f_t given a kernel of the form (2). We now have

$$f_t(x) = f(x,t) = \sum_{i=1}^{n} \underbrace{\alpha_i K_{tt_i}}_{\beta_i} k(x,x_i). \quad (3)$$

We see from this representation that the function f_t belongs to the RHKS $\mathcal{H}$ with kernel $k_{\mathcal{X}}$. Hence we can compare any two task-specific functions f_s and f_t using the inner product in $\mathcal{H}$:

$$\langle f_s, f_t\rangle_{\mathcal{H}} = \sum_i \sum_j \alpha_i\alpha_j K_{st_i}K_{tt_j}k(x_i,x_j).$$

For example, we can compute $\|f_s - f_t\|_{\mathcal{H}}$. What does it mean to regularize f in $\tilde{\mathcal{H}}$? This has a simple interpretation in terms of the task-specific functions f_t and the task kernel K.

Proposition 1. *Let $\tilde{k}$ be a product kernel of the form in (2). Let K^- be any matrix such that $KK^-K = K$. Then*

$$\|f\|^2_{\tilde{\mathcal{H}}} = \sum_{s,t\in\mathcal{T}} K^-_{st}\langle f_s, f_t\rangle_{\mathcal{H}}.$$

Proof. Write out the right side and rearrange. □

Proposition 1 is an extension of equations (19) and (20) in (Evgeniou et al., 2006) to the case when $k_{\mathcal{X}}$ is not linear, and allowing for the possibility that K is singular. Of course, if K is non-singular, we must have $K^- = K^{-1}$.

2.2. Graphical Multi-Task Kernels

Suppose we are given a graph $G = (\mathcal{T}, E)$ where edges between tasks represent similarity. One way to enforce similarity is to add a penalty when learning in the case that s and t are connected, but the distance between f_s and f_t is large. We will construct a task kernel that does exactly that. Let L be the graph Laplacian of G, i.e., the matrix with entries

$$L_{st} = \begin{cases} \deg(t) & s = t \\ -1 & s \neq t, (s,t) \in E \\ 0 & s \neq t, (s,t) \notin E \end{cases}$$

The Laplacian is not invertible, but if we set K equal to L^+, the Moore-Penrose pseudoinverse of L, then L satisfies the conditions of Proposition 1, and we have

$$\|f\|^2_{\tilde{\mathcal{H}}} = \sum_{s,t} L_{st}\langle f_s, f_t\rangle_{\mathcal{H}} = \sum_{(s,t)\in E} \|f_s - f_t\|^2_{\mathcal{H}} \quad (4)$$

The last equality is a well-known property of the graph Laplacian and is straightforward to verify. The Laplacian is easily extended to weighted graphs; with edge weights λ_{st}, the regularization term becomes $\|f\|^2_{\tilde{\mathcal{H}}} = \sum_{s,t} \lambda_{st} \|f_s - f_t\|^2_{\mathcal{H}}$.

2.3. Practical Considerations

As presented, the graphical multi-task kernel has a serious flaw. The regularization term is translation invariant in the sense that we can add any function f_0 to all of the task-specific functions and the quantity in (4) does not change, since it only involves differences between functions. This is undesirable: the learner can choose a very complex function f_0 and set all task-specific functions equal to f_0, badly overfitting the data, even though $\|f\|_{\tilde{\mathcal{H}}} = 0$. In (Evgeniou et al., 2006), the authors propose to resolve this by restricting all task-specific functions (linear in their case) to lie in a certain subspace; for unweighted graphs, their requirement boils down to requiring that $\sum_{t \in C} f_t = 0$ for all connected components C of G. This is also undesirable: for example, with this requirement it is impossible for tasks to agree on the sign of any input. For any task s that predicts $f_s(x) > 0$, some task t must predict $f_t(x) < 0$; moreover, within each component, these numbers must be balanced!

We suggest instead to add small diagonal entries α_t to L and use $(L + \mathrm{diag}(\alpha))^{-1}$ as the task kernel. As long as α_t is positive for some task in each connected component, this inverse exists, and the new regularization term is

$$\|f\|^2_{\tilde{\mathcal{H}}} = \sum_{s,t} \lambda_{st} \|f_s - f_t\|^2_{\mathcal{H}} + \sum_t \alpha_t \|f_t\|^2.$$

This has the effect of adding some individual regularization to each of the task-specific functions. For the experiments below, we choose $\alpha_t = \alpha$ for all t, and leave the single parameter α to be set by cross-validation.

We also introduce another practical extension: normalizing K to have unit diagonal. Then, our final task kernel K is obtained as follows:

$$\begin{gathered} \hat{K} = (L + \alpha I)^{-1}, \quad D = \mathrm{diag}(\hat{K}), \\ K = D^{-1/2} \hat{K} D^{-1/2}. \end{gathered} \tag{5}$$

Normalization is not strictly necessary — the main function is to restrict the entries of K to the interval $[0, 1]$, and ease the interpretability of K_{st} as the degree of similarity between s and t. Note that we recover the two naive approaches we suggested initially by choosing particular task kernels K of this form: if K is the identity matrix, we learn each task separately, and if K is equal to the all-ones matrix, we learn identical models for each task.

3. Experiments

We report preliminary experiments using SVMs with graphical multi-task kernels to solve a problem in species distribution modeling. The dataset consists of bird observations from eBird (`http://ebird.org`), a citizen science project where birdwatchers submit checklists of their observations online. Our problem is to predict whether the observer counted any Tree Swallows (*Tachycineta bicolor*), given information about the trip such as date, time, location, effort and terrain and habitat features.

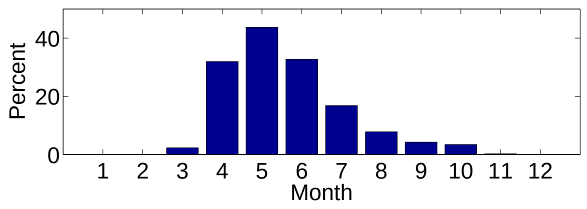


Figure 1. Percent positive examples by month.

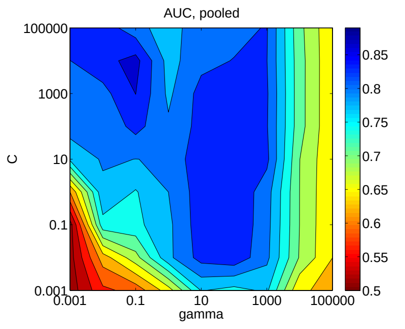
Our dataset consists of 5645 submissions from the year 2006 in a region spanning central New York and northern Pennsylvania. Of these, only 14% reported Tree Swallows; however there is a major seasonal shift (see Figure 1) due to migration. We treat each month of the year as a separate task, and measure performance using AUC, a common performance metric for species distribution modeling (Elith et al., 2006).

We tested three different methods:

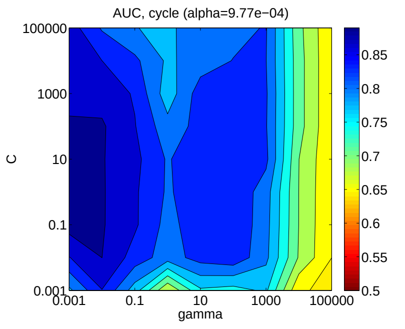
1. POOL trains a single SVM using all the training data — we include month as a feature but make no special treatment.
2. INDIV trains individual SVMS for each task.
3. CYCLE trains a single SVM using a graphical multi-task kernel as defined in equation (5). The task network is a cycle with unit edge weights, and we set $\alpha = 2^{-8}$.

For all methods, input features were centered and normalized to have unit variance. For INDIV, we used train/validation/test sets for each month of size $300/50/p$, where p is the number of remaining examples for the month, ranging from 37 to 318. For POOL and CYCLE, we used train/validation/test sets of size 3600/500/500 (the same total number of training examples as INDIV). All experiments used a version of libsvm (Chang & Lin, 2001) modified to support multi-task kernels. For all kernels except the task kernel, we used the RBF:

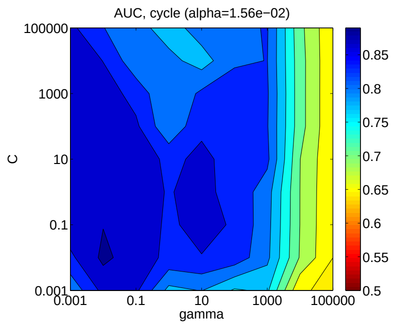
$$k(x, y) = \exp(-\gamma \|x - y\|^2),$$



(a)



(b)



(c)

Figure 2. AUC on validation set for a range of C and γ values: (a) POOL, (b) CYCLE, $\alpha = 2^{-10}$, (c) CYCLE, $\alpha = 2^{-6}$.

We performed a grid search over $C \in \{10^{-1}, 10, 10^3, 10^5\}$ and $\gamma \in \{10^{-3}, 10^{-1}, 10, 10^3\}$, choosing the parameters with best performance on the validation set and measuring AUC on the test set. The results were: .8136 for POOL, .8046 for INDIV, and .8554 for CYCLE, showing considerably better performance for the multi-task method.

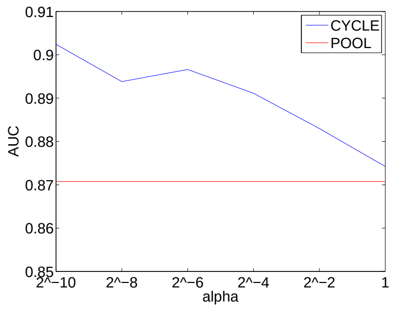


Figure 3. Best AUC of CYCLE vs POOL for for different values of α

We also took a closer look at the impact of α, γ, C for POOL and CYCLE, performing a grid search over C and γ in $\{10^{-3}, 10^{-2}, \ldots, 10^5\}$, and $\alpha \in \{2^{-10}, 2^{-8}, \ldots, 2^{-2}, 1\}$. This time we trained on 4645 examples, and report AUC on the validation set. For all values of α, the peak performance for CYCLE beat that of POOL (see Figure 3). Figure 2 shows AUC as a function of C and γ. Note that tuning is much less sensitive to specific values of C and γ for CYCLE.

4. Future Directions

There are many directions to explore. This framework seems natural for many applications, e.g., in social networks. In the species distribution problem, we would like to learn the distributions for many birds simultaneously using known relationships. Finally, we are developing methods to learn the parameters of the task kernel from data. For example, Figure 1 shows that not all pairs of adjacent months should be equally related: there are periods of rapid change followed by relative stability. Learning this from data would represent a significant advance.

References

Caruana, R. (1997). Multitask Learning. *Machine Learning*, *28*, 41–75.

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines.*

Elith, J., Graham, C. H., Anderson, R. P., Dudik, M., Ferrier, S., Guisan, A., Hijmans, R. J., Huettmann, F., Leathwick, J. R., Lehmann, A., Li, J., Lohmann, L. G., Loiselle, B. A., Manion, G., Moritz, C., Nakamura, M., Nakazawa, Y., Overton, J. M., Peterson, T. A., Phillips, S. J., Richardson, K., Scachetti-Pereira, R., Schapire, R. E., Soberon, J., Williams, S., Wisz, M. S., & Zimmermann, N. E. (2006). Novel methods improve prediction of species distributions from occurrence data. *Ecography*, *29*, 129–151.

Evgeniou, T., Micchelli, C., & Pontil, M. (2006). Learning Multiple Tasks with Kernel Methods. *Journal of Machine Learning Research*, *6*, 615.

Kato, T., Kashima, H., Sugiyama, M., & Asai, K. (2007). Multi-Task Learning via Conic Programming. *Advances in Neural Information Processing Systems.*

Rasmussen, C. (2006). *Gaussian processes for machine learning.* Springer.

Rossi, P., & Allenby, G. (2003). Bayesian Statistics and Marketing. *Marketing Science*, *22*, 304–328.

Xue, Y., Liao, X., Carin, L., & Krishnapuram, B. (2007). Multi-Task Learning for Classification with Dirichlet Process Priors. *The Journal of Machine Learning Research*, *8*, 35–63.

Yu, K., Tresp, V., & Schwaighofer, A. (2005). Learning Gaussian processes from multiple tasks. *Proceedings of the 22nd international conference on Machine learning* (pp. 1012–1019).