Journal of Machine Learning Research 12 (2011) 75-110 Submitted 4/09; Revised 10/10; Published 1/11

Multitask Sparsity via Maximum Entropy Discrimination

Tony Jebara JEBARA@CS.COLUMBIA.EDU
Department of Computer Science
Columbia University
New York, NY 10027, USA

Editor: Francis Bach

Abstract

A multitask learning framework is developed for discriminative classification and regression where multiple large-margin linear classifiers are estimated for different prediction problems. These classifiers operate in a common input space but are coupled as they recover an unknown shared representation. A maximum entropy discrimination (MED) framework is used to derive the multitask algorithm which involves only convex optimization problems that are straightforward to implement. Three multitask scenarios are described. The first multitask method produces multiple support vector machines that learn a shared sparse feature selection over the input space. The second multitask method produces multiple support vector machines that learn a shared conic kernel combination. The third multitask method produces a pooled classifier as well as adaptively specialized individual classifiers. Furthermore, extensions to regression, graphical model structure estimation and other sparse methods are discussed. The maximum entropy optimization problems are implemented via a sequential quadratic programming method which leverages recent progress in fast SVM solvers. Fast monotonic convergence bounds are provided by bounding the MED sparsifying cost function with a quadratic function and ensuring only a constant factor runtime increase above standard independent SVM solvers. Results are shown on multitask data sets and favor multitask learning over single-task or tabula rasa methods.

Keywords: meta-learning, support vector machines, feature selection, kernel selection, maximum entropy, large margin, Bayesian methods, variational bounds, classification, regression, Lasso, graphical model structure estimation, quadratic programming, convex programming

1. Introduction

In applied domains ranging from biology to vision, inter-related data is collected by researchers for varying scientific purposes. While there are some concerted efforts to ensure that data sets are collected and labeled in consistent ways, it is often the case that many heterogeneous data sets over a given input domain are collected and labeled for different tasks. Most machine learning approaches take a *single-task* perspective where one large homogeneous repository of uniformly collected *iid* (independent and identically distributed) samples is given and labeled consistently. A more realistic, *multitask learning* approach is to combine data from multiple smaller sources and synergistically leverage heterogeneous labeling or annotation efforts.

Consider a group of biologists that are investigating the gene regulatory pathways of a simple species such as yeast. Each biologist may measure the expression levels of a different subset of genes under particular perturbation conditions of interest. In addition, the biologists may annotate or label the gene expression data they collect in different ways. Clearly, each data set has dependencies and redundancies when compared to another data set. Single-task learning from each data set in isolation (in a tabula rasa inductive manner) provides only a narrow view of the phenomenon at hand. Meanwhile, multitask learning (or inductive transfer) uses the collection of data sets simultaneously to exploit the related nature of the problems. For example, a multitask learning approach may involve algorithms that discover shared representations that are useful across several data sets and tasks. For instance, consider a group of doctors each interested in predicting the presence or absence of a particular disease from a set of medical tests that can be performed on a patient. Since medical tests may be invasive and expensive, the doctors may wish to find a small subset of medical tests (the shared representation) that can be performed on a patient once and for all such that each disease of interest can be accurately predicted.

This article explores maximum entropy discrimination approaches to multitask problems and is organized as follows. Section 2 reviews previous work in multitask learning, support vector machine feature selection and support vector machine kernel selection. Section 3 sets up the general multitask problem as learning from data that has been sampled from a set of generative models that are dependent given data observations yet become independent given a shared representation. Section 4 migrates the standard Bayesian treatment of the problem into a large-margin discriminative setting using maximum entropy. The log-linear model, the main classifier of interest in this article, is explicated in Section 5. Section 6 explicates the case where the shared representation is a binary feature selection that removes certain input space features in a consistent manner for all linear classification tasks. Section 7 extends the shared representation such that it explores any conic kernel combination with multiple linear classifiers. Section 8 extends the framework to adaptive data pooling problems. Section 9 illustrates the corresponding derivations in a multitask (scalar) regression setting. Section 10 briefly describes the sequential quadratic programming method which is to be applied to the convex programs derived in the various preceding sections. Experimental results are provided in Section 11. An extension that permits the approach to perform sparse graph structure estimation is described in Section 12 and Section 13 then concludes with a brief summary. The Appendix provides the derivation of a bound which converts all the necessary optimization steps into quadratic programming with a proof of fast convergence for the resulting sequential quadratic programming procedure. The Appendix also discusses connections to other sparse regression methods.

2. Previous Work

Since this article involves the combination of the three research areas, we review previous work in multitask learning, support vector machine (SVM) feature selection and SVM kernel selection.

Multitask learning has many names and incarnations including learning-to-learn, meta-learning, lifelong learning, and inductive transfer (Baxter, 1995; Thrun and Pratt, 1997; Caruana, 1997; Thrun, 1995). It goes beyond the usual assumptions in most learning methods which focus on learning a model from a single training data set. Instead, multitask learning couples multiple models and their individual training sets and tasks. The hope is that the models can benefit from each other synergistically if their tasks are inter-related (predicting if a face is male or female may help when predicting if a face belongs to an adult or a child), the distributions of the training sets are related (transformed versions of each other) or the general domains of the tasks are similar (for instance all tasks involve images of outdoor scenery). Early implementations of multitask learning primarily investigated neural network or nearest neighbor learners (Thrun, 1995; Baxter, 1995; Caruana, 1997). In addition to neural approaches, Bayesian methods have been explored that implement multitask learning by assuming dependencies between the various models and tasks (Heskes, 1998, 2004). For instance, tasks can be clustered via a hierarchical mixture of Gaussians which couples their parameters. In addition, some theoretical arguments for the benefits of multitask learning have been made (Baxter, 2000) showing that the average error of M tasks can potentially decrease inversely with M. More recently, improved generalization guarantees for *each* individual task were provided if the classifiers are related and share a common structure (Ben-David and Schuller, 2003).

Concurrently, kernel methods (Schölkopf and Smola, 2001) and large-margin support vector machines are highly successful in single-task settings and are good candidates for multitask extensions. While multiclass variants of binary classifiers have been extensively explored (Crammer and Singer, 2001), multitask classification differs in that it often involves distinct sets of input data for each task. Furthermore, the concept of shared representation has been less practical to implement for kernel methods and support vector machines. For example, constraining the representation by performing SVM feature selection in a single-task setting may require extensions beyond standard quadratic programming (Jebara and Jaakkola, 2000; Weston et al., 2000). Similarly, constraining a representation to perform SVM kernel selection is also more involved in a single-task setting and requires second-order cone programming or semidefinite programming (Cristianini et al., 2001; Lanckriet et al., 2002).

This article focuses on multitask extensions of both feature selection and kernel selection with support vector machines. The derivations here will closely follow previous work which migrated maximum entropy to single-task SVMs (Jaakkola et al., 1999), to sparse SVMs (Jebara and Jaakkola, 2000) and to multitask SVMs (Jebara, 2003, 2004).[1] This maximum entropy framework led to one of the first convex large margin multitask classification approaches (Jebara, 2004). Convexity was subsequently explored in other multitask frameworks (Argyriou et al., 2008). The present article extends the derivations in the maximum entropy discrimination multitask approach, provides a straightforward iterative quadratic programming implementation and uses tighter bounds for improved runtime efficiency. Other related multitask SVM approaches have also been promising including novel kernel construction techniques to couple tasks (Evgeniou et al., 2005). These permit standard SVM learning algorithms to perform multitask learning while the multitask issues are handled primarily by the kernel itself. Even more recently, online algorithms have been proposed (Dekel et al., 2006) for multitask learning with margin-based predictors and provide interesting worst-case guarantees. Extensions to handle unlabeled data in multitask settings have also been promising (Ando and Zhang, 2005) and enjoyed theoretical generalization guarantees. An alternative perspective to multitask feature and kernel selection can be explored by performing joint covariate or subspace selection for multiple classification problems (Obozinski et al., 2010). Furthermore, feature selection and kernel selection can be seen as sparsity inducing methods. While a survey of sparsity is out of the scope of this article, one of the most popular implementations of sparsity or selection in regression settings is the ℓ_1 regularized Lasso method and its variants (Tibshirani, 1996; Tropp, 2006). Therein, sparsity is usually explored in a single-task setting and is used to remove unnecessary features in a regression problem (although sparsity is equally relevant in classification problems Jebara and Jaakkola, 2000). The multitask extension to such sparse estimation techniques is known as the Group Lasso and allows sparsity to be explored over predefined subsets of variables (Turlach et al., 2005; Yuan and Lin, 2006). Consistency arguments and connections between the Group Lasso and multiple kernel learning were also provided (Bach et al., 2004; Bach, 2008). Spar-

1. This article is the long version of a conference paper (Jebara, 2004).

sity and its connection to maximum entropy discrimination and so-called Laplace Markov networks was also recently explored (Zhu et al., 2008). This article provides another contact point between sparsity, large margins, multitask learning and kernel selection. The next sections formulate the general probabilistic setup for such multitask problems and convert traditional Bayesian solutions into a discriminative large-margin setting using the maximum entropy framework (Jaakkola et al., 1999).

3. Multitask Learning

The general multitask learning setup is as follows. We are given a collection of data sets $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$ covering $m = 1 \ldots M$ tasks. Each task has its training set $\mathcal{D}_m$ of $t = 1 \ldots T_m$ input-output pairs $(\mathbf{x}_{m,t}, y_{m,t})$ that are independent and identically distributed (*iid*) samples from an unknown probability density function $\mathcal{P}_m$ defined jointly over both inputs and outputs. The data for task m is therefore $\mathcal{D}_m = \{(\mathbf{x}_{m,1}, y_{m,1}), \ldots, (\mathbf{x}_{m,T_m}, y_{m,T_m})\}$. The inputs may be in a Euclidean vector space $\mathbf{x}_{m,t} \in \mathbb{R}^D$ or, more generally, $\mathbf{x}_{m,t} \in \mathcal{X}$ are objects that could be mapped to a Hilbert space via a kernel. In a regression setting we assume the outputs are scalars $y_{m,t} \in \mathbb{R}$ while in a classification setting we would assume binary[2] outputs $y_{m,t} \in \{\pm 1\}$.

There are many ways to tie together multiple inter-related tasks synergistically. In this section and in Section 4, it will be helpful to take a Bayesian perspective to the multitask problem although this perspective is not strictly necessary in subsequent sections. From a Bayesian point of view, several model parameters will be estimated and assumed to be random variables governed by a distribution and priors. Assume that there are task-specific model parameters Θ_m associated to each task or data set $\mathcal{D}_m$ for $m = 1 \ldots M$. The single-task or tabula rasa learning approach assumes that the models are independent given their respective data sets and, therefore, can be recovered independently. Such an assumption may be too simple in practice. The more general multitask learning assumption is that there exist dependencies between the tasks. In other words, the likelihood of the models given the data does not factorize,

$$p(\Theta_1, \ldots, \Theta_M | \mathcal{D}) \quad \neq \quad \prod_{m=1}^{M} p(\Theta_m | \mathcal{D}_m).$$

One specific way of coupling the various parameters $\Theta_1, \ldots, \Theta_M$ is to instead assume that there is another parameter $\mathbf{s}$ that is shared across tasks. For example, $\mathbf{s}$ could be a set of binary switches that eliminate all but a few features in the input space. The models then become independent only if the shared parameter[3] or *representation* is observed as follows:

$$p(\Theta_1, \ldots, \Theta_M | \mathbf{s}, \mathcal{D}) \quad = \quad \prod_{m=1}^{M} p(\Theta_m | \mathbf{s}, \mathcal{D}_m).$$

Note that, given data, the models are conditionally independent given the representation yet are dependent otherwise. This lack of factorization is on the *posterior* when data is observed, not on

2. In this article, only the binary classification case will be considered, however, the techniques herein extend easily to multiclass settings where $y_{m,t} \in \{1, \ldots, Y\}$ with $Y \in \mathbb{Z}$ and $Y \geq 3$. Alternatively, it is straightforward to use binary classification methods on multiclass problems by using Y one-versus-all binary classifiers, by using $Y(Y-1)/2$ one-versus-one binary classifiers, or by using error-correcting codes (Dietterich and Bakiri, 1995).

3. A more general approach is to assume a hierarchy of shared variables which couples the various learning tasks in more subtle ways (Heskes, 1998; Dudik et al., 2007). This hierarchical setting is out of the scope of this article but is of interest for future work.

the *prior*. We may still make the assumption that $p(\Theta)$ factorizes a priori. However, observing data with a latent shared parameter $\mathbf{s}$ induces dependencies across the multiple tasks. In terms of a directed acyclic graph where the joint probability density function factorizes as a product of nodes given their parents, the following dependency structure emerges in the (simplest) case of multitask learning with two models: $\Theta_1 \rightarrow \mathcal{D}_1 \leftarrow \mathbf{s} \rightarrow \mathcal{D}_2 \leftarrow \Theta_2$. Therefore, observing the data $\mathcal{D}_1$ and $\mathcal{D}_2$ couples the two models unless the shared representation $\mathbf{s}$ is also observed.

Thus, a natural way of exploring dependencies between tasks is to assume a shared representation variable $\mathbf{s}$ is implicated in the learning problem. We then have a total set of parameters $\Theta = \{\Theta_1, \ldots, \Theta_M, \mathbf{s}\}$ to *jointly* estimate from all the data sets. We explore the following scenarios:

- **Feature Selection:** Consider M individual models $\Theta_m = \{\theta_m, b_m\}$ which are linear classifiers where $\theta_m \in \mathbb{R}^D$ and $b_m \in \mathbb{R}$. The shared representation $\mathbf{s} \in \mathbb{B}^D$ is a binary feature selection vector that either keeps ($\mathbf{s}(d) = 1$) or eliminates ($\mathbf{s}(d) = 0$) each input vector dimension.

- **Kernel Selection:** Consider M individual models $\Theta_m = \{\theta_{m,1}, \ldots, \theta_{m,D}, b_m\}$ where each model Θ_m consists of D linear classifiers in D different Hilbert spaces and one scalar $b_m \in \mathbb{R}$. The shared configuration $\mathbf{s} \in \mathbb{B}^D$ is a binary feature selection vector that either keeps (when $\mathbf{s}(d) = 1$) or eliminates (when $\mathbf{s}(d) = 0$) the candidate Hilbert space from the classifiers.

- **Adaptive Pooling:** Consider $M+1$ different linear classification models where M tasks have to choose between using their own specialized classifier $\theta_1, \ldots, \theta_M$ or a communal classifier θ by estimating $\mathbf{s} \in \mathbb{B}^M$, a binary selection vector.

- **Graphical Model Structure:** Consider estimating from sample data a graphical model structure over D random variables by finding D classifiers that predict each variable from all others.

The following sections detail these multitask learning scenarios and show how we can learn discriminative classifiers (that predict outputs accurately and with large margin) from multiple tasks. To tackle this problem, we will apply the maximum entropy discrimination framework (Jaakkola et al., 1999) since it produces convex optimization problems where global optima can be reliably recovered. Furthermore, the framework produces large margin discrimination and thus inherits the performance benefits of support vector machines.

4. Bayes and Maximum Entropy

The standard Bayesian approach to inference begins with a prior $p(\Theta)$ over a model class Θ (which can be possibly uncountable or continuous). The prior is then refined given the data to obtain a posterior $p(\Theta|\mathcal{D})$ via Bayes' rule $p(\Theta|\mathcal{D}) \propto p(\mathcal{D}|\Theta)p(\Theta)$. Subsequently, the posterior is used to make predictions for new observations. The Bayesian prediction of a label for a new query input $\mathbf{x}$ for task m is as follows:

$$\hat{y} = \arg\max_y \int p(y|\mathbf{x}, \Theta_m) p(\Theta|\mathcal{D}) d\Theta. \tag{1}$$

In the above, a prediction $\hat{y}$ is obtained from the predictive distribution $p(y|\mathbf{x}, \Theta_m)$ by integrating over all models Θ while weighing each predictive distribution by the posterior $p(\Theta|\mathcal{D})$. This posterior, according the Bayes rule, is simply the product of the prior and the likelihood as follows:

$$p(\Theta|\mathcal{D}) \quad = \quad \frac{1}{Z} p(\Theta) \prod_{m=1}^{M} \prod_{t=1}^{T_m} p(y_{m,t}|\mathbf{x}_{m,t}, \Theta_m).$$

Previous approaches (Heskes, 2004) followed such a Bayesian treatment for multitask learning and obtained promising results. In this article, however, we will modify the standard Bayesian posterior to learn a more discriminative solution. Instead of using Bayes' rule to infer the posterior, we consider a posterior which produces predictions $\hat{y}$ with *large margin* as in the support vector machine (SVM) framework (Cortes and Vapnik, 1995). In other words, we will construct a discriminative posterior density which yields both accurate classification and large margins when used in Equation 1. Accurate classification on the observed data is obtained by forcing the marginal likelihood of the correct label $y_{m,t}$ to be larger than that of incorrect labels for each observation $t = 1, \ldots, T_m$ in all $m = 1, \ldots, M$ data sets:

$$\int p(y_{m,t}|\mathbf{x}_{m,t}, \Theta_m) p(\Theta|\mathcal{D}) d\Theta - \max_{y \neq y_{m,t}} \int p(y|\mathbf{x}_{m,t}, \Theta_m) p(\Theta|\mathcal{D}) d\Theta \quad \geq \quad 0.$$

This ensures that the posterior gives good predictions on average since the correct label $y_{m,t}$ has a higher probability than the wrong label (Crammer and Singer, 2001; Taskar et al., 2004). The above constraints require that the likelihood of the correct label remain larger than the likelihood of the incorrect label on average under the posterior over Θ. We consider one additional simplification for computational considerations. Instead of comparing *likelihoods*, we will require that the *log-likelihood* of the correct label is larger than the *log-likelihood* of the incorrect label on average under the posterior over Θ. Furthermore, to achieve large margin, we will force the posterior to not only make correct predictions but to also produce a score for the correct label that is at least a constant γ above the value obtained by incorrect labels:

$$\int \log p(y_{m,t}|\mathbf{x}_{m,t}, \Theta_m) p(\Theta|\mathcal{D}) d\Theta - \max_{y \neq y_{m,t}} \int \log p(y|\mathbf{x}_{m,t}, \Theta_m) p(\Theta|\mathcal{D}) d\Theta \quad \geq \quad \gamma.$$

In many parts of this article, without loss of generality, we will assume that $\gamma = 1$. These correct-classification constraints are applied to all training data $t = 1, \ldots, T_m$ for all tasks $m = 1 \ldots M$. Such classification or discrimination constraints were first introduced in the so-called maximum entropy discrimination (MED) framework (Jaakkola et al., 1999) and give rise to posterior distributions that mimic support vector machines and large-margin learning. The MED framework also conveniently leads to analytic expressions and closed-form solutions for all the necessary integrals. Instead of using Bayes rule to obtain the posterior, MED finds a posterior that is as close as possible to the prior in terms of Kullback-Leibler Divergence. In other words, it minimizes the relative entropy to the prior $\mathrm{KL}(p(\Theta|\mathcal{D}) \| p(\Theta))$ but still also *satisfies the above classification constraints*. This produces the following primal optimization problem:

$$O_{primal} \left\{ \begin{array}{l} \min_{P(\Theta|\mathcal{D})} \mathrm{KL}(p(\Theta|\mathcal{D}) \| p(\Theta)) \\ \text{s.t. } \int \log \left(\frac{p(y_{m,t}|\mathbf{x}_{m,t}, \Theta_m)}{p(y|\mathbf{x}_{m,t}, \Theta_m)} \right) p(\Theta|\mathcal{D}) d\Theta \geq \gamma \ \ \forall y \neq y_{m,t}, m, t. \end{array} \right.$$

The solution is straightforward and gives the following posterior:

$$p(\Theta|\mathcal{D}) \quad = \quad \frac{1}{Z(\boldsymbol{\lambda})} p(\Theta) \prod_{m=1}^{M} \prod_{t=1}^{T_m} \prod_{y \neq y_{m,t}} \left(\frac{p(y_{m,t}|\mathbf{x}_{m,t}, \Theta_m)}{p(y|\mathbf{x}_{m,t}, \Theta_m)} \right)^{\lambda_{m,t}} \exp(-\gamma \lambda_{m,t}). \tag{2}$$

Here, $\boldsymbol{\lambda}$ is a collection (or a vector) of non-negative Lagrange multipliers $\{\lambda_{m,t}\}$ for $m = 1, \ldots, M$ and $t = 1, \ldots, T_m$ that are used to enforce the inequality constraints. The normalizer for the above posterior is $Z(\boldsymbol{\lambda})$. Maximum entropy solves for the Lagrange multipliers by maximizing $J(\boldsymbol{\lambda}) = -\log Z(\boldsymbol{\lambda})$. This is simply the dual optimization

$$O_{dual}\left\{\ \max_{\boldsymbol{\lambda}\geq\mathbf{0}} -\log\int p(\Theta)\prod_{m=1}^{M}\prod_{t=1}^{T_m}\prod_{y\neq y_{m,t}}\left(\frac{p(y_{m,t}|\mathbf{x}_{m,t},\Theta_m)}{p(y|\mathbf{x}_{m,t},\Theta_m)}\right)^{\lambda_{m,t}}\exp(-\gamma\lambda_{m,t})d\Theta.\right.$$

If all $\lambda_{m,t}$ are set to 1, the posterior resembles the standard Bayesian estimate. However, MED estimates different weights $\lambda_{m,t}$ for each datum (or classification constraint) in the posterior. This ensures that the classification constraints are achieved. Instead of treating all points equally, the MED solution explores weights on each datum to adjust the Bayesian solution such that it obtains better classification on the training data. The expected *log*-likelihood of the data under the MED posterior satisfies the classification constraints while staying close to the prior. Furthermore, MED uses the expected *log*-likelihood of a new query point to make predictions as follows:

$$\hat{y} = \arg\max_{y} \mathrm{E}_{p(\Theta|\mathcal{D})}[\log p(y|\mathbf{x},\Theta_m)] = \arg\max_{y}\int \log p(y|\mathbf{x},\Theta_m)p(\Theta|\mathcal{D})d\Theta.$$

This simple reformulation of the standard Bayesian posterior will give rise to large margin learning as explicated in the next section.

5. From Log-Linear Models to Support Vector Machines

We next make more specific assumptions on the form of the predictive distribution $p(y|\mathbf{x},\Theta_m)$. Assume that the predictive distribution is log-linear as follows:

$$p(y|\mathbf{x},\Theta_m) \quad\propto\quad \exp\left(\frac{y}{2}\langle\mathbf{x},\theta_m\rangle + b_m\right).$$

This permits us to rewrite the above posterior $p(\Theta|\mathcal{D})$ more specifically as:

$$p(\Theta|\mathcal{D}) \quad=\quad \frac{1}{Z(\boldsymbol{\lambda})}p(\Theta)\prod_{m=1}^{M}\prod_{t=1}^{T_m}\exp(y_{m,t}(\langle\mathbf{x}_{m,t},\theta_m\rangle + b_m))^{\lambda_{m,t}}\exp(-\gamma\lambda_{m,t}).$$

We integrate the above over $\Theta = \{\Theta_1,\ldots,\Theta_M,\mathbf{s}\}$ to obtain the partition function $Z(\boldsymbol{\lambda})$. The objective function we need to maximize is the negative logarithm of the partition function:

$$J(\boldsymbol{\lambda}) \quad=\quad -\log\int p(\Theta)\exp\left(\sum_{m=1}^{M}\sum_{t=1}^{T_m}\lambda_{m,t}y_{m,t}(\langle\mathbf{x}_{m,t},\theta_m\rangle + b_m) - \gamma\lambda_{m,t}\right)d\Theta.$$

We will next *assume* that the prior over models factorizes as follows:

$$p(\Theta) \quad=\quad p(\mathbf{s})\prod_{m=1}^{M}p(\Theta_m) = p(\mathbf{s})\prod_{m=1}^{M}\mathcal{N}(\theta_m|\mathbf{0},I)\mathcal{N}(b_m|0,\sigma^2).$$

and assume that the priors over parameters are all white Gaussians with zero mean and identity covariance (take $\mathbf{0}$ to be the vector of all zeros and I to be the identity matrix). This factorization assumption on the prior will be kept throughout this article. Although the prior factorizes, this does not necessarily mean that the posterior will factorize too. The likelihood terms in Equation 2 may couple the models in the posterior. However, in this first example we will not obtain any coupling per se. This is clear once we evaluate the integrals to obtain the objective function:

$$\begin{aligned} J(\boldsymbol{\lambda}) &= -\sum_{m=1}^{M} \log \int \exp(\langle \theta_m, \sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \mathbf{x}_{m,t} \rangle) \mathcal{N}(\theta_m | \mathbf{0}, I) d\theta_m \\ & \quad - \sum_{m=1}^{M} \log \int \exp(b_m \sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t}) \mathcal{N}(b_m | 0, \sigma^2) db_m + \sum_{m=1}^{M} \sum_{t=1}^{T_m} \gamma \lambda_{m,t}. \end{aligned}$$

Simple algebra and completion of squares[4] yields the objective function $J(\boldsymbol{\lambda})$ which is maximized as follows

$$\max_{\boldsymbol{\lambda} \geq \mathbf{0}} \sum_{m=1}^{M} \left(\sum_{t=1}^{T_m} \gamma \lambda_{m,t} - \frac{1}{2} \sum_{t,\tau=1}^{T_m} \lambda_{m,t} \lambda_{m,\tau} y_{m,t} y_{m,\tau} \langle \mathbf{x}_{m,t}, \mathbf{x}_{m,\tau} \rangle - \frac{\sigma^2}{2} \left(\sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \right)^2 \right).$$

The above dual optimization problem is simply a quadratic program and is straightforward to solve. If we further assume that $\sigma^2 \to \infty$, which corresponds to using a non-informative prior on the bias scalar terms b_m, the objective function above gives the constraints $\sum_t \lambda_{m,t} y_{m,t} = 0$ for all $m = 1 \ldots M$. We then get an objective function that is exactly the sum of the dual objective functions of M independent support vector machines (if $\gamma = 1$). Thus, our dual optimization is:

$$\max_{\boldsymbol{\lambda} \geq \mathbf{0}} \quad \sum_{m=1}^{M} \left(\sum_{t=1}^{T_m} \gamma \lambda_{m,t} - \frac{1}{2} \sum_{t,\tau=1}^{T_m} \lambda_{m,t} \lambda_{m,\tau} y_{m,t} y_{m,\tau} k(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau}) \right) \text{ s.t.} \sum_{t=1}^{T_m} y_{m,t} \lambda_{m,t} = 0 \ \forall m.$$

Here, we have also replaced all inner products of two inputs $\mathbf{x}$ and $\bar{\mathbf{x}}$ of the form $\langle \mathbf{x}, \bar{\mathbf{x}} \rangle$ with Mercer kernel evaluations $k(\mathbf{x}, \bar{\mathbf{x}})$. This allows us to readily accommodate nonlinear classification. Finally, the prediction rule for a query input $\mathbf{x}$ given the current setting of the $\boldsymbol{\lambda}$ values for the m'th model involves integrating over the posterior which produces the following prediction:

$$\hat{y} = \arg\max_{y} \mathrm{E}_{p(\Theta|\mathcal{D})} [\log p(y | \mathbf{x}, \Theta_m)] = \operatorname{sign} \left(\sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} k(\mathbf{x}, \mathbf{x}_{m,t}) + \hat{b}_m \right),$$

where the $\hat{b}_m$ scalars are given by the Karush Kuhn Tucker (KKT) conditions. Whenever a constraint or Lagrangian is active, the corresponding Lagrange multiplier must be strictly positive $\lambda_{m,t} > 0$ and we expect the inequalities in the primal problem to hold exactly. Therefore, we can obtain each $\hat{b}_m$ by solving

$$y_{m,t} = \sum_{\tau=1}^{T_m} \lambda_{m,\tau} y_{m,\tau} k(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau}) + \hat{b}_m$$

for any datum t which has a corresponding Lagrange multiplier (once the dual program halts) that satisfies $\lambda_{m,t} > 0$.

Clearly, because of additivity, updating $\lambda_{m,1}, \ldots, \lambda_{m,T_m}$ can be done independently of $\lambda_{n,1}, \ldots,$ λ_{n,T_n} for any $n \neq m$. In other words, we have tabula rasa independent learning of M independent SVMs on all the tasks. Even the scalar biases $\hat{b}_m$ are obtained independently via the KKT conditions. Therefore, to obtain multitask learning, we will need some shared representation $\mathbf{s}$ to couple the learning problems and give rise to a non-factorized posterior over models.

4. Recall that $\int \exp(\langle \theta, w \rangle) \mathcal{N}(\theta | \mathbf{0}, I) d\theta = \exp(\langle w, w \rangle / 2)$.

5.1 Non-Separable Case

For thoroughness, this section details the case where the classification problems are not separable; in other words, not all inequalities in the maximum entropy formulation can be achieved. In this case, we introduce non-negative slack variables $\xi = \{\xi_{m,t}\}$ on each constraint with a cost of C per unit of slack leading to the following primal optimization:

$$O_{primal}\left\{\begin{array}{l} \min_{P(\Theta|\mathcal{D}),\xi} \mathrm{KL}(p(\Theta|\mathcal{D})\|p(\Theta)) + C\sum_{m=1}^{M}\sum_{t=1}^{T_m}\sum_{y\neq y_{m,t}}\xi_{m,t,y} \\ \text{s.t. } \int \log\left(\frac{p(y_{m,t}|\mathbf{x}_{m,t},\Theta_m)}{p(y|\mathbf{x}_{m,t},\Theta_m)}\right) p(\Theta|\mathcal{D})d\Theta \geq \gamma - \xi_{m,t} \text{ and } \xi_{m,t,y} \geq 0 \ \ \forall y \neq y_{m,t}, m, t. \end{array}\right.$$

The above produces the same type of solution as Equation 2 but has a slightly different dual optimization:

$$O_{dual}\left\{\begin{array}{l} \max_{\boldsymbol{\lambda}} \sum_{m=1}^{M}\left(\sum_{t=1}^{T_m}\gamma\lambda_{m,t} - \frac{1}{2}\sum_{t,\tau=1}^{T_m}\lambda_{m,t}\lambda_{m,\tau}y_{m,t}y_{m,\tau}k(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})\right) \\ \text{s.t. } 0 \leq \lambda_{m,t} \leq C \ \ \forall m,t \text{ and } \sum_{t=1}^{T_m} y_{m,t}\lambda_{m,t} = 0 \ \ \forall m \end{array}\right.$$

which merely bounds the Lagrange multipliers from above by C. Once again, MED mimics support vector machines (Cortes and Vapnik, 1995) in the non-separable case.

6. Feature Selection

We next explore feature selection and require $\mathbf{x} \in \mathbb{R}^D$ where $D \in \mathbb{Z}$. To couple the tasks, modify the predictive distribution for the label given the model such that it also depends on a *shared* variable $\mathbf{s}$ as follows:

$$p(y|\mathbf{x},\Theta_m,\mathbf{s}) \quad \propto \quad \exp\left(\frac{y}{2}\left(\sum_{d=1}^{D}\mathbf{s}(d)\mathbf{x}(d)\theta_m(d) + b_m\right)\right),$$

where $\mathbf{s}$ is a binary D-dimensional vector and the argument of a vector $\mathbf{s}(d)$ refers to its d'th entry. Thus, the shared representation consists of binary switches that delete or censor various entries of the $\mathbf{x}$ vector. If $\mathbf{s}(d) = 0$, then the $\mathbf{x}(d)$ entry is effectively set to zero. Meanwhile, if $\mathbf{s}(d) = 1$, the $\mathbf{x}(d)$ entry remains intact. In other words, the binary vector $\mathbf{s}$ performs a feature selection. In addition, assume the prior for $p(\mathbf{s})$ is a product of Bernoulli distributions for each element of $\mathbf{s}$,

$$p(\mathbf{s}) \quad = \quad \prod_{d=1}^{D}\rho^{\mathbf{s}(d)}(1-\rho)^{1-\mathbf{s}(d)},$$

where ρ is the a priori probability of keeping the features on. For example, setting $\rho = 1$ suggests that all features should be on and no feature selection is to be performed. Alternatively, we can reparametrize the prior as $\alpha = \frac{1-\rho}{\rho}$ where increasing α corresponds to sparser feature selection. A value of $\alpha = 0$ indicates no feature selection is being performed (no sparsity). Meanwhile a value of $\alpha \to \infty$ encourages the models to discard almost all features. If we perform feature selection and use a predictive distribution with shared $\mathbf{s}$, the $m = 1,\ldots,M$ tasks will become coupled and the posterior over models no longer factorizes. The MED solution is then

$$p(\Theta|\mathcal{D}) \quad = \quad \frac{1}{Z(\boldsymbol{\lambda})}p(\Theta)\prod_{m=1}^{M}\prod_{t=1}^{T_m}\exp\left(\lambda_{m,t}y_{m,t}\left(\sum_{d=1}^{D}\mathbf{s}(d)\mathbf{x}_{m,t}(d)\theta_m(d) + b_m\right) - \gamma\lambda_{m,t}\right).$$

We compute the corresponding partition function by integrating over all models $\Theta_1, \ldots, \Theta_m$ as well as summing over all binary settings of $\mathbf{s}$ which yields

$$\begin{aligned} Z(\boldsymbol{\lambda}) &= \int p(\Theta) \exp\left(\sum_{m=1}^{M} \sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \left(\sum_{d=1}^{D} \mathbf{s}(d) \mathbf{x}_{m,t}(d) \theta_m(d) + b_m \right) - \gamma \lambda_{m,t} \right) d\Theta \\ &= \exp\left(\sum_m \frac{\sigma^2}{2} \left(\sum_t \lambda_{m,t} y_{m,t} \right)^2 - \sum_t \gamma \lambda_{m,t} \right) \prod_d \left(1 - \rho + \rho e^{\frac{1}{2} \sum_m (\sum_t \lambda_{m,t} y_{m,t} \mathbf{x}_{m,t}(d))^2} \right). \end{aligned}$$

Taking $\sigma^2 \to \infty$ gives a new objective function $J(\boldsymbol{\lambda}) = -\log(Z(\boldsymbol{\lambda}))$ which is no longer a quadratic program yet is still a convex program as follows:

$$\begin{cases} \max_{\boldsymbol{\lambda}} \sum_{m=1}^{M} \sum_{t=1}^{T_m} \gamma \lambda_{m,t} - \sum_{d=1}^{D} \log\left(\alpha + e^{\frac{1}{2} \sum_{m=1}^{M} \left(\sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \mathbf{x}_{m,t}(d) \right)^2} \right) + D \log(\alpha + 1) \\ \text{s.t. } 0 \leq \lambda_{m,t} \leq C \ \ \forall m,t \text{ and } \sum_{t=1}^{T_m} y_{m,t} \lambda_{m,t} = 0 \ \ \forall m. \end{cases}$$

Note the property that $J(\mathbf{0}) = 0$. Clearly, the objective function is no longer additive across $m = 1 \ldots M$ which means that learning is coupled across tasks. This is due to the non-linearity in the function $f(x) = \log(\alpha + \exp(-x))$ which involves a summation over $m = 1 \ldots M$. We will refer to this function as the log-sigmoid function. If we set $\alpha = 0$, the log-sigmoid becomes linear and we get back the independent optimization problems in Section 5. Therein, the tasks decouple completely (i.e., the objective function becomes additive over tasks $m = 1 \ldots M$). However, larger settings of α encourage some coupling between the SVMs (or large margin log-linear models) as they search for a joint feature selection.

Note the presence of logarithmic terms which prevent the direct application of quadratic programming to $J(\boldsymbol{\lambda})$. Fortunately, the log-sigmoid function $f(x) = \log(\alpha + \exp(-x))$ is known to be a convex function (more precisely, our objective involves a negated sum of such functions which is concave overall). Recently, new computational tools have been proposed for solving convex programs that involve such terms (Koh et al., 2007). In our implementation, we instead apply a bound on the log-sigmoid to reformulate the optimization as a sequential quadratic program. Optimization details are deferred to Section 10 but it will be assumed that a (nearly) optimal $\boldsymbol{\lambda}$ solution can be recovered.

Given the recovered $\boldsymbol{\lambda}$ setting, the prediction rule is straightforward to derive as follows:

$$\hat{y} = \arg\max_y \mathrm{E}_{p(\Theta|\mathcal{D})}[\log p(y|\mathbf{x}, \Theta_m, \mathbf{s})] = \operatorname{sign}\left(\sum_{d=1}^{D} \sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \hat{\mathbf{s}}(d) \mathbf{x}(d) \mathbf{x}_{m,t}(d) + \hat{b}_m \right).$$

The $\hat{\mathbf{s}}(d)$ above are expected values of $\mathbf{s}(d)$ under the posterior and are given by:

$$\hat{\mathbf{s}}(d) = \frac{1}{1 + \alpha \exp\left(-\frac{1}{2} \sum_{m=1}^{M} \left(\sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \mathbf{x}_{m,t}(d) \right)^2 \right)}.$$

In fact, $\hat{\mathbf{s}}(d)$ are scalars in $[1/(1+\alpha), 1]$ which give a soft feature selection as values close to the bottom of the range are candidates for removal after thresholding. The values of $\hat{\mathbf{s}}(d)$ multiplicatively scale the input domain features and are close to $1/(1+\alpha)$ for features that are not useful for prediction in the multiple tasks. Once again, the $\hat{b}_m$ scalars are given by the KKT conditions which require that the value inside the sign() function evaluation above exactly equals $y_{m,t}$ for the query $\mathbf{x} = \mathbf{x}_{m,t}$ whenever the corresponding Lagrange multiplier strictly satisfies $0 < \lambda_{m,t} < C$.

7. Kernel Selection

Feature selection and sparsity are not the only types of shared representation $\mathbf{s}$ one may consider. One crucial design issue of nonlinear SVMs is the choice of a kernel function. Also, kernels permit SVMs to handle non-vectorial inputs so we relax the assumption that the $\mathbf{x}$ inputs are Euclidean vectors and only require that they are objects from some sample space $\mathbf{x}_{m,t} \in \mathcal{X}$ for all $m = 1, \ldots, M$ and $t = 1, \ldots, T_m$. Typically, in kernel learning (Lanckriet et al., 2002), we are given a set of $d = 1, \ldots, D$ base Mercer kernels $k_1, \ldots, k_D$ where each kernel function $k_d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ accepts two inputs and produces a scalar. We wish to learn a conic combination of the kernels or a sparse selection using the non-negative scalar weights $w_1, \ldots, w_D$ as follows

$$K(\mathbf{x}, \bar{\mathbf{x}}) \quad = \quad \sum_{d=1}^{D} w_d k_d(\mathbf{x}, \bar{\mathbf{x}}).$$

Some base kernels may get a small weight and are thus not selected and others will be averaged with varying weights w_d to produce a potentially better final kernel K. Each base kernel $k_d(\mathbf{x}, \bar{\mathbf{x}})$ can be seen to correspond to a mapping ϕ_d which is applied to both inputs $\mathbf{x}$ and $\bar{\mathbf{x}}$. The function ϕ_d maps an input $\mathbf{x} \in \mathcal{X}$ to some Hilbert space we denote Φ_d. The kernel is then the inner-product of $\phi_d(\mathbf{x})$ and $\phi_d(\bar{\mathbf{x}})$ as follows:

$$k_d(\mathbf{x}, \bar{\mathbf{x}}) \quad = \quad \langle \phi_d(\mathbf{x}), \phi_d(\bar{\mathbf{x}}) \rangle.$$

Kernel selection is equivalent to selecting some mappings and attenuating others. We thus need a shared representation vector $\mathbf{s}$ which is again binary and again D-dimensional to select which kernels will be used. However, now, we have a set of $M \times D$ linear models $\theta_{m,d} \in \Phi_d^*$ for each Hilbert space. A $\theta_{m,d}$ vector is available for each task $m = 1, \ldots, M$ and each mapping $d = 1, \ldots, D$. In other words, task m has the following modeling resources on its own: $\Theta_m = \{\theta_{m,1}, \ldots, \theta_{m,D}, b_m\}$. The prior for the modeling resources for the m'th task is then chosen to be a product of independent white Gaussians on these D vector parameters. This leads to the following general prior for all model parameters:

$$p(\Theta) \quad = \quad \prod_{d=1}^{D} \rho^{\mathbf{s}(d)} (1-\rho)^{1-\mathbf{s}(d)} \prod_{m=1}^{M} \mathcal{N}(\theta_{m,d} | \mathbf{0}, I) \mathcal{N}(b_m | 0, \sigma^2).$$

Once again, all tasks share and have to agree on the binary selector vector $\mathbf{s}$ which inherits the Bernoulli prior used in the previous section. Therefore, we have the following total set of parameters $\Theta = \{\Theta_1, \ldots, \Theta_M, \mathbf{s}\}$.

The predictive distribution for multitask kernel selection is then given by the following log-linear model (for the m'th task):

$$p(y | \mathbf{x}, \Theta_m, \mathbf{s}) \quad \propto \quad \exp\left(\frac{y}{2} \left(\sum_{d=1}^{D} \mathbf{s}(d) \langle \theta_{m,d}, \phi_d(\mathbf{x}) \rangle + b_m \right) \right).$$

We once again recover the MED posterior using Equation 2. The normalizer for the posterior $Z(\boldsymbol{\lambda})$ is then found by integrating over the parameters Θ. This multitask kernel selection objective function $J(\boldsymbol{\lambda})$ is the following convex program:

$$\begin{cases} \max_{\boldsymbol{\lambda}} \sum_{m=1}^{M} \sum_{t=1}^{T_m} \gamma \lambda_{m,t} + D \log(\alpha + 1) \\ \qquad - \sum_{d=1}^{D} \log\left(\alpha + e^{\frac{1}{2} \sum_{m=1}^{M} \sum_{t=1}^{T_m} \sum_{\tau=1}^{T_m} \lambda_{m,t} \lambda_{m,\tau} y_{m,t} y_{m,\tau} k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau})} \right) \\ \text{s.t. } 0 \leq \lambda_{m,t} \leq C \;\; \forall m,t \text{ and } \sum_{t=1}^{T_m} y_{m,t} \lambda_{m,t} = 0 \;\; \forall m. \end{cases}$$

Similarly, given the $\boldsymbol{\lambda}$ setting, we obtain the following prediction rule:

$$\hat{y} = \text{sign}\left(\sum_{d=1}^{D}\sum_{t=1}^{T_m}\lambda_{m,t}y_{m,t}\hat{\mathbf{s}}(d)k_d(\mathbf{x},\mathbf{x}_{m,t}) + \hat{b}_m\right)$$

where $\hat{\mathbf{s}}(d)$ are scalars that weight each kernel and are usually close to $1/(1+\alpha)$ for kernels that are not beneficial for our multiple classification tasks. The weights for each kernel are recovered as:

$$\hat{\mathbf{s}}(d) = \frac{1}{1+\alpha\exp\left(-\frac{1}{2}\sum_{m=1}^{M}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m}\lambda_{m,t}\lambda_{m,\tau}y_{m,t}y_{m,\tau}k_d(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})\right)}.$$

The scalar biases $\hat{b}_m$ are once again recovered from the KKT conditions. From the prediction rule, it is clear that kernel learning is effectively creating a new kernel from the base kernels as follows:

$$K(\mathbf{x},\overline{\mathbf{x}}) = \sum_{d=1}^{D}\hat{\mathbf{s}}(d)k_d(\mathbf{x},\overline{\mathbf{x}}).$$

When $\alpha = 0$, there is no coupling of tasks or sparse selection of kernels. The solution simply corresponds to setting $\hat{\mathbf{s}}(d) = 1$ and forces the final kernel K to equal a simple sum of all base kernels for $d = 1,\ldots,D$. In general, however, a more appropriate final kernel could potentially be recovered if $\alpha > 0$. Given such an aggregate kernel $K(\mathbf{x},\overline{\mathbf{x}})$, we can now write an SVM-like prediction rule for the m'th task:

$$\hat{y} = \text{sign}\left(\sum_{t=1}^{T_m}\lambda_{m,t}y_{m,t}K(\mathbf{x},\mathbf{x}_{m,t}) + \hat{b}_m\right).$$

Another interesting fact is that feature selection is just an instance of kernel selection. If we choose the $d = 1,\ldots,D$ kernels as follows

$$k_d(\mathbf{x},\overline{\mathbf{x}}) = \mathbf{x}(d)\overline{\mathbf{x}}(d).$$

we are effectively replacing kernel evaluations in this section with the scalar product of the d'th dimension of the input that was needed for feature selection. Thus, the kernel selection problem in this section clearly subsumes the feature selection problem derived in Section 6.

7.1 Independent Kernel Selection

It is possible to break the above multitask framework by allowing each task to select a combination of kernels independently. This means that we introduce a separate $\mathbf{s}_m$ vector for each task $m = 1,\ldots,M$ instead of having a shared representation $\mathbf{s}$. The derivation is straightforward and produces the following convex program:

$$\begin{cases} \max_{\lambda} \sum_{m=1}^{M}\sum_{t=1}^{T_m}\gamma\lambda_{m,t} + MD\log(\alpha+1) \\ \qquad -\sum_{m=1}^{M}\sum_{d=1}^{D}\log\left(\alpha + e^{\frac{1}{2}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m}\lambda_{m,t}\lambda_{m,\tau}y_{m,t}y_{m,\tau}k_d(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})}\right) \\ \text{s.t. } 0 \le \lambda_{m,t} \le C \ \ \forall m,t \text{ and } \sum_{t=1}^{T_m}y_{m,t}\lambda_{m,t} = 0 \ \ \forall m \end{cases}$$

which is once again additive in $m = 1,\ldots,M$ indicating that the Lagrange multipliers for each task are estimated independently in a tabula rasa learning method. As usual, the prediction rule is given by $\hat{y} = \arg\max_y \mathrm{E}_{p(\Theta|\mathcal{D})}[\log p(y|\mathbf{x},\Theta_m)]$ and the following formula emerges for the expected switches:

$$\hat{\mathbf{s}}_m(d) \quad = \quad \frac{1}{1+\alpha\exp\left(-\frac{1}{2}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m}\lambda_{m,t}\lambda_{m,\tau}y_{m,t}y_{m,\tau}k_d(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})\right)}.$$

The prediction function for each task then simply uses its own $\hat{\mathbf{s}}_m(d)$ weights to combine the base kernels. This approach resembles the multiple kernel learning method (Lanckriet et al., 2002) since each task performs its own kernel selection in isolation.

7.2 Metric Learning

It is known that a Mercer kernel $k(\mathbf{x},\overline{\mathbf{x}})$ or affinity can be used to construct a distance metric $\Delta(\mathbf{x},\overline{\mathbf{x}})$ that satisfies standard requirements such as the triangle inequality. Consider constructing a base distance metric $\Delta_d(\mathbf{x},\overline{\mathbf{x}})$ from each base kernel $k_d(\mathbf{x},\overline{\mathbf{x}})$ as follows:

$$\Delta_d(\mathbf{x},\overline{\mathbf{x}}) \quad = \quad \sqrt{k_d(\mathbf{x},\mathbf{x})-2k_d(\mathbf{x},\overline{\mathbf{x}})+k_d(\overline{\mathbf{x}},\overline{\mathbf{x}})}.$$

Given this multitask kernel selection framework, it is possible to use the above formula to perform multitask metric learning. By applying the algorithm in Section 7, we obtain the kernel weights $\hat{\mathbf{s}}(1),\ldots,\hat{\mathbf{s}}(D)$. This permits us to learn an overall kernel as a conic combination of the set of base kernels. This solution can then be mapped into a learned distance metric as follows:

$$\Delta(\mathbf{x},\overline{\mathbf{x}}) \quad = \quad \sqrt{\sum_{d=1}^{D}\hat{\mathbf{s}}(d)\left(\Delta_d(\mathbf{x},\overline{\mathbf{x}})\right)^2}.$$

Thus, metric learning can be performed using the multitask kernel selection setup. Once a new kernel is learned, it is then possible to reconstruct the corresponding distance metric and apply any kernel or distance-based learning algorithm. For instance, kernel principal components analysis (Schölkopf et al., 1999) or any distance-based learning algorithm such as kernel nearest neighbors and kernel clustering can be used with such learned kernels and distance functions.

8. Shared Classifiers and Adaptive Pooling

Another interesting multitask learning approach involves shared classifiers or shared models. For example, if we have very few training examples for each task, we may consider *pooling* all tasks together and learning a single classifier for all. This may help initially yet some tasks with more training examples than others may want to specialize and form their own independent classifiers once we are confident these tasks have enough supporting data. Once again assume we have $m = 1,\ldots,M$ tasks. These tasks have to choose between using their own specialized classifier $\theta_1,\ldots,\theta_M$ or a communal classifier θ. To avoid a trivial solution, only some of the tasks are allowed to become specialized and use their own linear model. Consider a binary feature selection vector $\mathbf{s}\in\mathbb{B}^M$. For each task m, the element of the vector $\mathbf{s}(m)\in\mathbb{B}$ determines if the task will use its own specialized θ_m model (when $\mathbf{s}(m)=1$) or use the communal θ model (when $\mathbf{s}(m)=0$) for discrimination. This setup is clarified by the following log-linear predictive distribution for the m'th task:

$$p(y|m,\mathbf{x},\Theta,\mathbf{s}) \quad \propto \quad \exp\left(\frac{y}{2}\left(\mathbf{s}(m)\langle\theta_m,\phi_m(\mathbf{x})\rangle+\langle\theta,\phi(\mathbf{x})\rangle+b_m\right)\right).$$

The communal classifier is over a single Hilbert space mapping $\phi(\mathbf{x})$ while the specialized classifiers may operate over their own distinct Hilbert space mapping $\phi_m(\mathbf{x})$. Inner products in these Hilbert spaces are computed using kernels as usual $k(\mathbf{x},\bar{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\bar{\mathbf{x}})\rangle$ and $k_m(\mathbf{x},\bar{\mathbf{x}}) = \langle \phi_m(\mathbf{x}), \phi_m(\bar{\mathbf{x}})\rangle$. Another subtlety is that each task still has its own dedicated b_m constant scalar bias. The complete set of models is therefore $\Theta = \{\theta, \theta_1, \ldots, \theta_M, b_1, \ldots, b_M\}$. We can assume the priors on all models are white Gaussian distributions. We also continue to use Bernoulli priors for $\mathbf{s}(m)$ and zero-mean Gaussian priors for the biases. The normalizer for the posterior is recovered as:

$$\begin{aligned}
Z(\boldsymbol{\lambda}) &= \int p(\Theta) e^{\sum_{m=1}^M \sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} (\mathbf{s}(m)\langle \theta_m, \phi_m(\mathbf{x}_{m,t})\rangle + \langle \theta, \phi(\mathbf{x}_{m,t})\rangle + b_m) - \gamma\lambda_{m,t}} d\Theta \\
&= e^{\sum_m \frac{\sigma^2}{2}(\sum_t \lambda_{m,t} y_{m,t})^2} e^{-\sum_m \sum_t \gamma\lambda_{m,t}} e^{\frac{1}{2}\sum_m \sum_n \sum_t \sum_\tau \lambda_{m,t}\lambda_{n,\tau} y_{m,t} y_{n,\tau} k(\mathbf{x}_{m,t},\mathbf{x}_{n,\tau})} \\
&\quad \times \sum_{\mathbf{s}(1)} \cdots \sum_{\mathbf{s}(M)} p(\mathbf{s}) \prod_m e^{\frac{1}{2}\mathbf{s}(m)\sum_t \sum_\tau \lambda_{m,t}\lambda_{m,\tau} y_{m,t} y_{m,\tau} k_m(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})}.
\end{aligned}$$

The final summations over the binary switch settings above distribute and become straightforward. We assume that $\sigma \to \infty$ and obtain the following objective function $J(\boldsymbol{\lambda})$

$$\left\{ \begin{array}{l}
\max_{\boldsymbol{\lambda}} \sum_{m=1}^M \sum_{t=1}^{T_m} \gamma\lambda_{m,t} - \frac{1}{2}\sum_{m=1}^M \sum_{n=1}^M \sum_{t=1}^{T_m} \sum_{\tau=1}^{T_n} \lambda_{m,t}\lambda_{n,\tau} y_{m,t} y_{n,\tau} k(\mathbf{x}_{m,t},\mathbf{x}_{n,\tau}) \\
\qquad -\sum_{m=1}^M \log\left(\alpha + e^{\frac{1}{2}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m} \lambda_{m,t}\lambda_{m,\tau} y_{m,t} y_{m,\tau} k_m(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})}\right) + M\log(\alpha+1) \\
\text{s.t. } 0 \le \lambda_{m,t} \le C \ \ \forall m,t \text{ and } \sum_{t=1}^{T_m} y_{m,t}\lambda_{m,t} = 0 \ \ \forall m
\end{array} \right.$$

which clearly shows that the tasks cannot be solved independently (since the quadratic term above sums over both m and n which couples all pairs of tasks). The solution of the above is once again a convex program. Given the optimal Lagrange multiplier solution, the prediction rule for an input $\mathbf{x}$ for the m'th task is given by:

$$\hat{y} = \text{sign}\left(\hat{\mathbf{s}}(m)\sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} k_m(\mathbf{x},\mathbf{x}_{m,t}) + \sum_{n=1}^M \sum_{t=1}^{T_n} \lambda_{n,t} y_{n,t} k(\mathbf{x},\mathbf{x}_{n,t}) + \hat{b}_m\right).$$

We recover the expected $\mathbf{s}(m)$ value which measures our confidence in using a specialized classifier for the m'th task as follows

$$\hat{\mathbf{s}}(m) = \frac{1}{1 + \alpha\exp\left(-\frac{1}{2}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m} \lambda_{m,t}\lambda_{m,\tau} y_{m,t} y_{m,\tau} k_m(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})\right)}.$$

It is interesting to note that if α is infinity, then all the $\hat{\mathbf{s}}(m)$ values go to zero and the method performs complete pooling. Conversely, if $\alpha = 0$, then $\hat{\mathbf{s}}(m) = 1$ and each classifier mixes its specialized linear model equally with the communal model. It is natural to use a smaller scale for $k(\mathbf{x},\bar{\mathbf{x}})$ than $k_m(\mathbf{x},\bar{\mathbf{x}})$ such that the choice $\alpha = 0$ leads to a more specialized setting with M independent classifiers while larger α leads to a more communal setting with a single classifier. For instance, in the absence of any domain-specific knowledge, a good heuristic is to choose $k_m(\mathbf{x},\bar{\mathbf{x}}) = \omega k(\mathbf{x},\bar{\mathbf{x}})$ with $\omega = 10M$. Ultimately, the benefits of adaptive pooling will emerge if there is a natural trade-off between specialization and sharing at different rates for each of the M tasks as embodied by the non-uniform estimator of $\hat{\mathbf{s}}$ above.

9. Regression

It is easy to convert multitask feature selection, kernel selection and pooling problems to a regression setup where outputs are scalars $y_{m,t} \in \mathbb{R}$. While this article will only show experiments with classification problems, the multitask regression setting is briefly summarized here for completeness. The main decision in regression problems is what loss function to impose on output predictions. While many loss functions may be considered in regression problems, a popular one is the epsilon-tube loss.

In this type of regression, the goal is to predict the targets within $\pm\varepsilon$. Recall the close similarity between the dual learning problems for SVM classification and SVM regression (Schölkopf and Smola, 2001). The maximum entropy posterior can also be used to reproduce support vector machine regression (Jebara and Jaakkola, 2000; Jebara, 2003). Instead of following the MED derivations in detail, this subsection simply shows the resulting objective function which largely agrees with the standard quadratic program for (single-task) SVM regression with an ε-tube:

$$\max_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} \sum_{t=1}^{T} y_t(\lambda_t - \lambda'_t) - \varepsilon \sum_{t=1}^{T} (\lambda_t + \lambda'_t) - \frac{1}{2} \sum_{t=1}^{T} \sum_{\tau=1}^{T} (\lambda_t - \lambda'_t)(\lambda_\tau - \lambda'_\tau) k(\mathbf{x}_t, \mathbf{x}_\tau)$$

$$\text{s.t. } 0 \le \lambda_t, \lambda'_t \le C, \text{ and } \sum_{t=1}^{T} \lambda_t = \sum_{t=1}^{T} \lambda'_t$$

which is solved over Lagrange multipliers $\boldsymbol{\lambda} = \{\lambda_t\}$ and $\boldsymbol{\lambda}' = \{\lambda'_t\}$ for all $t = 1 \ldots T$. SVM regression then applies the following prediction rule:

$$\hat{y} \;\; = \;\; \sum_{t=1}^{T} (\lambda_t - \lambda'_t) k(\mathbf{x}, \mathbf{x}_t) + \hat{b}.$$

It is straightforward to adapt this regression problem to multitask kernel selection (which once again subsumes feature selection if we select $k_d(\mathbf{x}, \overline{\mathbf{x}}) = \mathbf{x}(d)\overline{\mathbf{x}}(d)$). MED yields the following multitask objective function which is a convex program:

$$\begin{cases} \max_{\boldsymbol{\lambda},\boldsymbol{\lambda}'} \sum_{m=1}^{M} \sum_{t=1}^{T_m} y_{m,t}(\lambda_{m,t} - \lambda_{m,t'}) - \varepsilon \sum_{m=1}^{M} \sum_{t=1}^{T_m} (\lambda_{m,t} + \lambda_{m,t'}) + D \log(\alpha + 1) \\ \quad - \sum_{d=1}^{D} \log \left(\alpha + e^{\frac{1}{2} \sum_{m=1}^{M} \sum_{t=1}^{T_m} \sum_{\tau=1}^{T_m} (\lambda_{m,t} - \lambda'_{m,t})(\lambda_{m,\tau} - \lambda'_{m,\tau}) k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau})} \right) \\ \text{s.t. } 0 \le \lambda_{m,t}, \lambda'_{m,t} \le C \;\; \forall m,t \text{ and } \sum_{t=1}^{T_m} \lambda_{m,t} - \lambda'_{m,t} = 0 \;\; \forall m. \end{cases}$$

The above is solved by adjusting the Lagrange multipliers $\boldsymbol{\lambda} = \{\lambda_{t,m}\}$ and $\boldsymbol{\lambda}' = \{\lambda'_{t,m}\}$ for all $t = 1 \ldots T_m$ and all $m = 1, \ldots, M$. The resulting prediction rule for a query datum $\mathbf{x}$ for the m'th regression task is then:

$$\hat{y} \;\; = \;\; \sum_{t=1}^{T_m} (\lambda_{m,t} - \lambda'_{m,t}) K(\mathbf{x}, \mathbf{x}_{m,t}) + \hat{b}_m$$

with the kernel $K(\mathbf{x}, \overline{\mathbf{x}}) = \sum_{d=1}^{D} \hat{\mathbf{s}}(d) k_d(\mathbf{x}, \overline{\mathbf{x}})$ as a conic combination of base kernels with weights

$$\hat{\mathbf{s}}(d) \;\; = \;\; \frac{1}{1 + \alpha \exp(-\frac{1}{2} \sum_{m=1}^{M} \sum_{t=1}^{T_m} \sum_{\tau=1}^{T_m} (\lambda_{m,t} - \lambda'_{m,t})(\lambda_{m,\tau} - \lambda'_{m,\tau}) k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau}))}.$$

Finally, the biases $\hat{b}_m$ for each task are obtained by solving for the KKT conditions at active Lagrange constraints. Appendix C discusses other choices for the MED loss function in regression settings and connections to previous sparse approaches (Ridge, Lasso and Elastic-Net regression).

10. Sequential Quadratic Programming

In all the optimization problems introduced so far, the optimization appears to be extremely similar to a quadratic program (QP) except for the presence of a handful of log-sigmoid functions. In fact, if the parameter α is set to zero, all the above optimization problems simplify into quadratic programs. It will be shown that the $\alpha > 0$ case can also be easily handled by quadratic programming as well. More precisely, it can be optimized using a sequential quadratic programming (SQP) method. This is a procedure which iteratively solves a QP for a number of iterations. In fact, if the QP is of a simple SVM-type form, much faster SVM solvers can be used instead of QP (Joachims, 2006; Shalev-Shwartz et al., 2007; Bottou and Bousquet, 2008; Shalev-Shwartz and Srebro, 2008). The next section explicates how all MED optimization problems encountered so far can be solved via SQP (or sequential SVM solutions) by bounding the log-sigmoid terms with quadratic functions.

For brevity, we focus on the multitask kernel selection problem which strictly subsumes multitask feature selection. Other learning problems in the previous sections can be implemented with sequential quadratic programming in a similar manner. Recall the kernel selection optimization:

$$\left\{ \begin{array}{l} \max_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}) = \sum_{m=1}^{M} \sum_{t=1}^{T_m} \gamma \lambda_{m,t} + D \log(\alpha + 1) \\ \qquad - \sum_{d=1}^{D} \log \left(\alpha + e^{\frac{1}{2} \sum_{m=1}^{M} \sum_{t=1}^{T_m} \sum_{\tau=1}^{T_m} \lambda_{m,t} \lambda_{m,\tau} y_{m,t} y_{m,\tau} k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau})} \right) \\ \text{s.t. } 0 \leq \lambda_{m,t} \leq C \;\; \forall m,t \text{ and } \sum_{t=1}^{T_m} y_{m,t} \lambda_{m,t} = 0 \;\; \forall m. \end{array} \right.$$

This is a convex problem and generic methods exist for solving it including the ellipsoid method. The latter is a polynomial time algorithm requiring $\mathrm{O}((\sum_{m=1}^{M} T_m)^3)$ time yet may still be impractically slow in practice due to large scaling constants (Boyd and Vanderberghe, 2004). Some related optimization methods involving logistic terms have been explored with the Lasso problem (Tibshirani, 1996). Logistic terms often emerge in algorithms that learn sparse (feature-selected) linear classifiers by maximizing the logistic likelihood while enforcing an ℓ_1 regularization on the linear model parameters. This is the approach followed by the ℓ_1 regularized sparse logistic regression technique (Koh et al., 2007). Interestingly, this recent work has developed fast interior-point optimization methods which may be eventually applicable to MED problems. Instead, we solve the MED problem by exploiting a convenient upper bound on logistic-quadratic functions that converts them into plain quadratic functions. In previous work, a looser version of the bound was proposed (Jebara and Jaakkola, 2000). This article refines the bound and provides a tight variational quadratic upper bound on a logistic function *of* a quadratic function. This conversion to quadratic functions permits us to use standard quadratic programming. In fact, the actual optimizations ultimately decouple into the solution of M separate support vector machines and prevent cubic growth in the number of tasks. Bounding is interleaved with the solution of support vector machines to iteratively maximize $J(\boldsymbol{\lambda})$. Due to the availability of fast SVM solvers, this optimization approach is potentially more promising than more generic convex programming tools (Koh et al., 2007). The necessary bound is derived in detail in Theorem 1 in the Appendix. The theorem states that $\log\left(\alpha + \exp\left(\frac{\mathbf{u}^\top \mathbf{u}}{2}\right)\right)$ is less than or equal to a convex quadratic function in $\mathbf{u}$ for all vectors $\mathbf{u}$ and achieves strict equality when $\mathbf{u} = \mathbf{v}$ for some vector $\mathbf{v}$. We will apply the above bound to each log-sigmoid term in the sum over $d = 1 \ldots D$ in $J(\boldsymbol{\lambda})$. We slightly abuse notation and interchangeably use $\boldsymbol{\lambda}_m$ to denote the vector of Lagrange multipliers $(\lambda_{m,1}, \ldots, \lambda_{m,T_m})^\top$ for each $m = 1 \ldots M$. Similarly, we will take $\boldsymbol{\lambda} \in \mathbb{R}^\Gamma$ where $\Gamma = \sum_{m=1}^{M} T_m$ to be a concatenation of all Lagrange multipliers. Consider the d'th log-sigmoid function in the sum $\sum_{d=1}^{D} \log(\ldots)$ in $J(\boldsymbol{\lambda})$. Denote the Hessian of the quadratic term inside the d'th log-sigmoid as $H_d \in \mathbb{R}^{\Gamma\times\Gamma}$ which is given element-wise as follows:

$$H_d([m,t],[n,\tau]) \quad = \quad y_{m,t} y_{m,\tau} k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau}) \delta_{m=n}.$$

Here we use $\delta_{m=n}$ as an indicator function that is 1 if $m = n$ and is zero otherwise. We also use the operator $[m,t]$ to compute the index value $[m,t] = (t + \sum_{n=1}^{m-1} T_n)$ to select the appropriate row and column entries of the matrix H_d. This allows us to write the dual objective function as

$$\begin{cases} \max_{\boldsymbol{\lambda}} J(\boldsymbol{\lambda}) = D\log(\alpha+1) - \sum_{d=1}^{D} \log\left(\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}^\top H_d \boldsymbol{\lambda}\right)\right) + \gamma\boldsymbol{\lambda}^\top \mathbf{1} \\ \text{s.t. } 0 \le \lambda_{m,t} \le C \ \ \forall m,t \text{ and } \sum_{t=1}^{T_m} y_{m,t}\lambda_{m,t} = 0 \ \ \forall m. \end{cases}$$

Assume we have a current setting of the Lagrange multipliers $\tilde{\boldsymbol{\lambda}}$. We apply Theorem 1 in the Appendix after a simple change of variables, $\mathbf{u} = H_d^{1/2}\boldsymbol{\lambda}$ and $\mathbf{v} = H_d^{1/2}\tilde{\boldsymbol{\lambda}}$ which gives:

$$\begin{aligned} \log\left(\alpha + \exp\left(\frac{\boldsymbol{\lambda}^\top H_d \boldsymbol{\lambda}}{2}\right)\right) \quad \le \quad & \log\left(\alpha + \exp\left(\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2}\right)\right) + \frac{\exp(\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2})}{\alpha + \exp(\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2})} \tilde{\boldsymbol{\lambda}}^\top H_d (\boldsymbol{\lambda} - \tilde{\boldsymbol{\lambda}}) \\ & + \frac{1}{2}(\boldsymbol{\lambda} - \tilde{\boldsymbol{\lambda}})^\top \left(\mathcal{G}_d H_d \tilde{\boldsymbol{\lambda}}\tilde{\boldsymbol{\lambda}}^\top H_d + H_d\right)(\boldsymbol{\lambda} - \tilde{\boldsymbol{\lambda}}). \end{aligned}$$

Such a bound is applied to each log-sigmoid term in $J(\boldsymbol{\lambda})$ individually for $d = 1 \ldots D$. The ratio terms in the bound are none other than the expected switch variables at the current setting of $\tilde{\boldsymbol{\lambda}}$:

$$\hat{\mathbf{s}}(d) \quad = \quad \frac{\exp(\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2})}{\alpha + \exp(\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2})} = \frac{1}{\alpha \exp(-\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2}) + 1}.$$

Similarly, we obtain the following for $\mathcal{G}_d$ applying[5] the bound formula:

$$\mathcal{G}_d \quad = \quad \frac{\tanh(\frac{1}{2}\log(\alpha\exp(-\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2})))}{2\log(\alpha\exp(-\frac{\tilde{\boldsymbol{\lambda}}^\top H_d \tilde{\boldsymbol{\lambda}}}{2})}.$$

Other convenient variables to define are the vectors $\hat{\mathbf{y}}_{m,t} \in \mathbb{R}^D$ for $m = 1, \ldots, M$ and $t = 1, \ldots, T_m$. These are the predicted label of the m'th SVM on the t'th datum using the d'th kernel at the current setting of $\tilde{\boldsymbol{\lambda}}$. They are given element-wise as follows:

$$\hat{\mathbf{y}}_{m,t}(d) \quad = \quad \sum_{\tau=1}^{T_m} \tilde{\boldsymbol{\lambda}}_{m,\tau} y_{m,\tau} k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau}).$$

Applying these substitutions and the bound on each log-sigmoid function produces the following variational lower bound on the objective function:

$$\begin{aligned} J(\boldsymbol{\lambda}) \quad \ge \quad & \text{constant} + \sum_{m=1}^{M}\sum_{t=1}^{T_m} \gamma\lambda_{m,t} - \sum_{m=1}^{M}\sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \sum_{d=1}^{D} \hat{\mathbf{s}}(d)\hat{\mathbf{y}}_{m,t}(d) \\ & + \sum_{m=1}^{M}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m} \lambda_{m,t}\tilde{\lambda}_{m,\tau} y_{m,t} y_{m,\tau} \sum_{d=1}^{D} \left(\mathcal{G}_d \hat{\mathbf{y}}_{m,t}(d)\hat{\mathbf{y}}_{m,\tau}(d) + k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau})\right). \\ & - \frac{1}{2}\sum_{m=1}^{M}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m} \lambda_{m,t}\lambda_{m,\tau} y_{m,t} y_{m,\tau} \sum_{d=1}^{D} \left(\mathcal{G}_d \hat{\mathbf{y}}_{m,t}(d)\hat{\mathbf{y}}_{m,\tau}(d) + k_d(\mathbf{x}_{m,t}, \mathbf{x}_{m,\tau})\right). \end{aligned}$$

5. By continuity, take $\tanh(\frac{1}{2}\log(1))/(2\log(1)) = 1/4$ and also take $\lim_{z\to 0^+} \tanh(\frac{1}{2}\log(z))/(2\log(z)) = 0$.

Interestingly, given the current $\tilde{\boldsymbol{\lambda}}$ and the current $\hat{\mathbf{s}}(1),\ldots,\hat{\mathbf{s}}(D)$, the bound effectively *decouples* the learning problem across the M tasks. The objective function becomes quadratic and additive across $m = 1 \ldots M$. Therefore, we can solve each problem individually as a single support vector machine. This provides a simple iterative algorithm for multitask learning which builds on current SVM solvers. The steps[6] are summarized in Algorithm 1.

Algorithm 1 simply performs sequential quadratic programming by interleaving the bound computation with SVM programs. The SVMs are solved separately for $m = 1,\ldots,M$ tasks in Step 3b. If each SVM is solved using standard quadratic programming solvers, each requires $\mathrm{O}(T_m^3)$. However, by exploiting more recent *approximate* SVM solvers, the inner loop SVM problems can potentially complete in linear time or $\mathrm{O}(T_m)$ (Joachims, 2006; Shalev-Shwartz et al., 2007; Bottou and Bousquet, 2008; Shalev-Shwartz and Srebro, 2008). Admittedly, this is true only subject to certain reasonable assumptions (for instance, small approximation errors are allowed and explicit linear feature mappings are used rather then implicit nonlinear kernels). Therefore, under certain assumptions, step 3b in Algorithm 1 can potentially complete in $\mathrm{O}(\sum_{m=1}^M T_m)$ time.[7] Finally, it is also possible to use warm-starting and seed the SVM solver with a previous $\boldsymbol{\lambda}$ result to obtain further speedup. For instance, warm starting can be used from a previous iteration in Algorithm 1. Furthermore, we may warm start from a previous final solution of Algorithm 1 that converged for a smaller setting of C or α. This lets us explore the regularization path efficiently after initializing it at, for instance, the default setting of $\alpha = 0$ and $C = 1$ and increasing both parameters until error is minimized on a cross-validation set. Furthermore, we typically set $\gamma = 1$ to mimic the support vector machine case but that parameter may be adjusted as well (either manually or by cross-validation).

Algorithm 1 Multitask SVM Learning

```
0   Input data set $\mathcal{D}$, $C > 0$, $\alpha \geq 0$, $0 < \varpi < 1$ and kernels $k_d$ for $d = 1,\ldots,D$.
1   Initialize Lagrange multipliers to zero $\boldsymbol{\lambda} = \mathbf{0}$.
2   Store $\tilde{\boldsymbol{\lambda}} = \boldsymbol{\lambda}$.
3   For $m = 1,\ldots,M$ do:
3a  Set $g_d = \alpha \exp\left(-\frac{1}{2}\sum_{m=1}^M \sum_{t=1}^{T_m} \sum_{\tau=1}^{T_m} \lambda_{m,t}\lambda_{m,\tau} y_{m,t} y_{m,\tau} k_d(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})\right)$ for all $d$.
    Set $\mathcal{G}_d = \frac{\tanh(\frac{1}{2}\log(g_d))}{2\log(g_d)}$ for all $d$.
    Set $\hat{\mathbf{s}}(d) = \frac{1}{1+g_d}$ for all $d$.
    Set $\hat{\mathbf{y}}_{m,t}(d) = \sum_{\tau=1}^{T_m} \lambda_{m,\tau} y_{m,\tau} k_d(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})$ for all $t$ and $d$.
3b  Update each of the $\boldsymbol{\lambda}_m$ vectors with the SVM QP:
    $\max_{\boldsymbol{\lambda}_m} \sum_{t=1}^{T_m} \lambda_{m,t} - \sum_{t=1}^{T_m} \lambda_{m,t} y_{m,t} \sum_{d=1}^D \hat{\mathbf{s}}(d)\hat{\mathbf{y}}_{m,t}(d)$
      $+\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m} \lambda_{m,t}\tilde{\lambda}_{m,\tau} y_{m,t} y_{m,\tau} \sum_{d=1}^D \left(\mathcal{G}_d \hat{\mathbf{y}}_{m,t}(d)\hat{\mathbf{y}}_{m,\tau}(d) + k_d(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})\right)$
      $-\frac{1}{2}\sum_{t=1}^{T_m}\sum_{\tau=1}^{T_m} \lambda_{m,t}\lambda_{m,\tau} y_{m,t} y_{m,\tau} \sum_{d=1}^D \left(\mathcal{G}_d \hat{\mathbf{y}}_{m,t}(d)\hat{\mathbf{y}}_{m,\tau}(d) + k_d(\mathbf{x}_{m,t},\mathbf{x}_{m,\tau})\right)$
    s.t. $0 \leq \lambda_{m,t} \leq C \ \ \forall t = 1,\ldots,T_m$ and $\sum_{t=1}^{T_m} y_{m,t}\lambda_{m,t} = 0$.
4   If $\|\boldsymbol{\lambda} - \tilde{\boldsymbol{\lambda}}\| > \varpi\|\boldsymbol{\lambda}\|$ go to 2.
5   Output: $\hat{\mathbf{s}}$ and $\boldsymbol{\lambda}$.
```

Next, we discuss the convergence of the above iterative algorithm. Clearly, since the algorithm maximizes a variational lower bound on the objective function, it must monotonically increase the objective. However, it is still possible that the algorithm can get stuck and produce negligible

6. Code available at `www.cs.columbia.edu/~jebara/code/multisparse/`.
7. Also, under mild *iid* assumptions, step 3a can be well approximated in $\mathrm{O}(\sum_{m=1}^M T_m)$ time using deviation bounds.

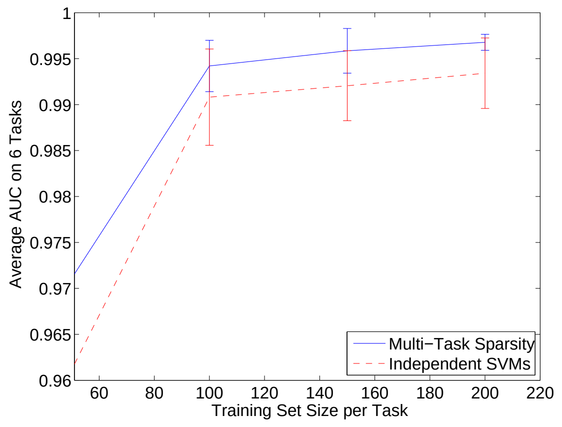


Figure 1: Feature selection on the UCI Dermatology data set. Multitask sparse feature selection and independent SVM classification are compared. Various data set sizes are shown ranging from 20 to 200 samples for each of the 6 tasks. The average area under the ROC curve on test data is shown for all tasks for 5 folds (along with the standard deviation). The values of C and α were obtained by cross-validation on held out data.

progress requiring an unbounded number of iterations. We will show that is not the case and, indeed, the sequential quadratic programming procedure in Algorithm 1 will only require a finite number of iterations (of step 3). The number of iterations is bounded by Theorem 2 which is proved in the Appendix. It guarantees that, for any $\alpha \geq 0, \varepsilon \in (0,1)$, Algorithm 1 finds a $\tilde{\boldsymbol{\lambda}}$ that satisfies $J(\tilde{\boldsymbol{\lambda}}) \geq (1-\varepsilon)J(\boldsymbol{\lambda}^*)$ (where $\boldsymbol{\lambda}^*$ is the constrained maximizer of $J(\boldsymbol{\lambda})$) in no more than

$$\left\lceil \frac{\log(1/\varepsilon)}{\log\left(\min\left(1+\frac{1}{\alpha},2\right)\right)} \right\rceil$$

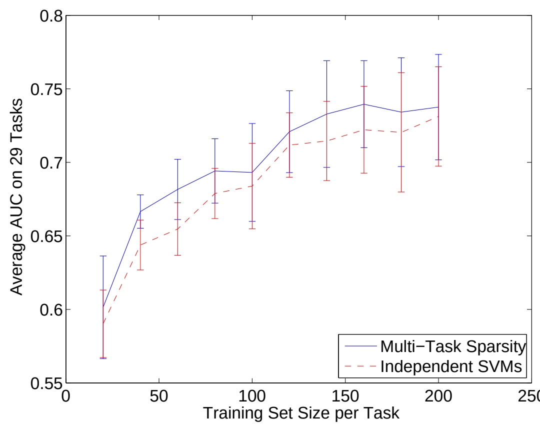
iterations. Here, each iteration involves (possibly warm-started) SVM programs and the expression $\lceil \ldots \rceil$ denotes the integer ceiling function.

Therefore, a constant number of iterations is needed that depends only on α. In summary, solving multitask feature or kernel selection is only a constant factor more computational effort than solving M independent support vector machines. A similar SQP or iterative SVM algorithm can be derived for the adaptive pooling setup described in Section 8.

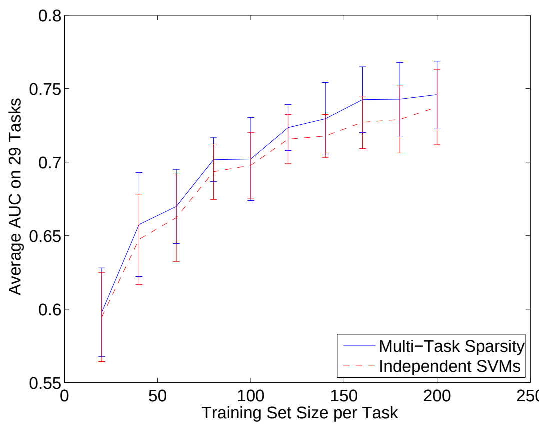
11. Experiments

To evaluate the multitask learning framework, we considered UCI data[8] as well as the Land Mine data set[9] which was developed and investigated in previous work (Xue et al., 2007). The classification accuracy of standard support vector machines learned independently is compared to the accuracy of the multitask kernel selection procedure described in Section 6 and Section 7. In all experiments, we explore multiple values of the regularizer C for the SVM and multiple values of

8. Data available at `http://archive.ics.uci.edu/ml/`.
9. Data available at `http://www.cs.columbia.edu/~jebara/code/multisparse/LandmineData.mat`.



(a) Feature and RBF kernel selection



(b) Feature, polynomial and RBF kernel selection

Figure 2: Feature selection and kernel selection on the Landmine data set. In (a), feature selection is combined with RBF kernel selection. In (b), feature selection is combined with both polynomial and RBF kernel selection. Multitask sparse kernel selection and independent SVM classification are compared. Various data set sizes are shown ranging from 20 to 200 samples for each of the 29 tasks. The average area under the ROC curve on test data is shown for all tasks for 5 folds (along with the standard deviation). The values of C and α were obtained by cross-validation on held out data.

C and α (or, equivalently, ρ) for the multitask learner. The values of C and α are determined by cross-validation on held out data and then tested on an unseen test set.

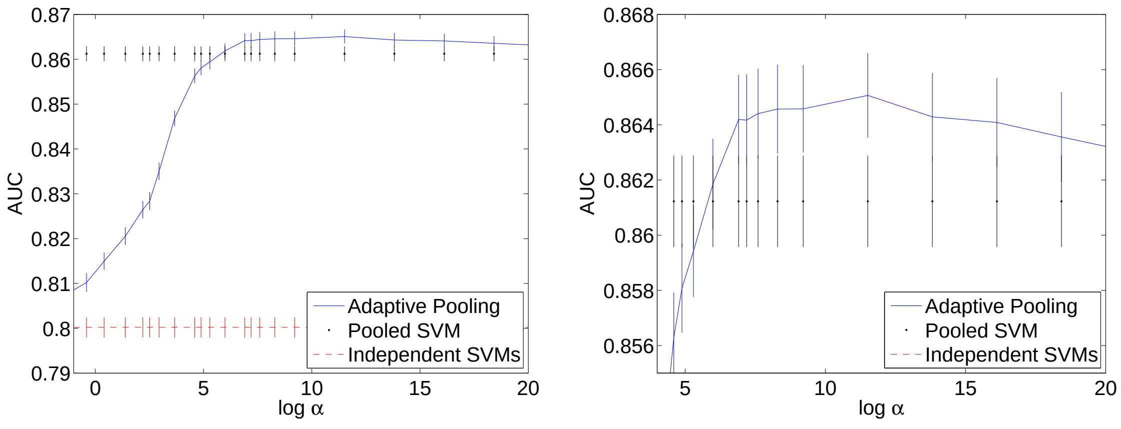
The UCI dermatology data set consists of 6 classes which can be converted into binary classification tasks to be predicted from an input space of 34-dimensional features. A total of 366 instances are available. Both the independent SVMs and the multitask feature selection approach were evaluated by training on various numbers of examples (from 20 to 200) for each task, and the remaining examples (with labels kept unobserved) are split in half for cross-validation and testing. The feature selection method chooses a sparse subset of the 34 features that are consistently good at predicting the label for the 6 different tasks (or classes). All evaluations were done using the average area under the Receiver Operating Characteristic (ROC) curve for the 6 tasks. This score is the MAUC since it involves the mean of M tasks' Area Under the Curve (AUC) scores. Cross-validation was used to select a value of C for the independent SVMs and values of α and C for the multitask feature selection SVMs. Figure 1 shows the MAUC performance of the independent SVMs versus the multitask SVMs with averages and standard deviations across 5 folds. There is a clear and statistically significant advantage (under a paired t-test) for multitask learning over independent SVM classification.

The Landmine data set consists of 29 binary classification tasks involving an input space of $D = 9$ dimensional features. The number of samples for each task varies from 445 to 690. Both independent SVM learning and the multitask kernel selection approach were evaluated by training on various numbers of examples $(20, 40, \dots, 200)$ from each task. The remaining examples were split in half for cross-validation and for testing. We perform feature selection by building a kernel for each feature that is simply the product of a single scalar dimension for a pair of data points. This produces 9 kernels. In addition, 9 radial basis function (RBF) kernels were computed with different settings of the bandwidth parameter. The kernel selection method was then used to choose a sparse subset of these $D = 18$ total kernels. All evaluations were done using the average area under the ROC curve for the 29 tasks. Cross-validation was used to select a value of C for the independent SVM approach and to select values for C and α for the multitask kernel selection SVM. Figure 2(a) shows the performance of the independent SVMs versus the multitask SVMs as an average and standard deviation of MAUC across 5 folds. Tabula rasa learning obtains lower accuracy in general while multitask learning improved accuracy at all sizes of the training data set with statistical significance (a paired t-test produced a p-value below 0.05) on most training set sizes.

Another experiment exploring kernel selection was considered using all the previous kernels as well as linear, quadratic, cubic and quartic kernels for a total of $D = 22$ kernels. Figure 2(b) summarizes the results which again demonstrate an advantage for the multitask setup. These results compare favorably with previous experiments on this data set (Xue et al., 2007).

In all experiments, solving the more elaborate objective function in the MED convex program required only a constant factor more time than solving each task separately with independent SVMs. We verified that the number of iterations of the SVMs only increased as a function of α and required 2 to 40 iterations of Step 3 in Algorithm 1 as α was swept across the range of interest. Since the SVMs were warm-started at their previous solutions, sweeping across a range of α values in the multitask sparsity approach (after starting from an initial SVM solution) never required more than 50 times the run time of the initial SVM solution. Thus, empirically, the multitask sparsity framework, while sweeping over the full regularization path over α, incurs a constant factor (under 50) increase in the computational effort over independent SVM learning. These runtime results agree with Theorem 2.

In another experiment with adaptive pooling, the Heart data set from the UCI repository was used. All features were normalized to within the $[0,1]$ box and a polynomial kernel of degree three was used throughout. The Heart data set was changed into a multitask data set by dividing the data into ten different tasks based on the age of the patient. This division was done by splitting the data along the age variable by forming 10 intervals with equal number of examples in each interval. For each task, the examples were divided into train/test/validation sets with equal number of examples in each. A scaling factor of $\frac{1}{10}$ was applied to the communal kernel and a scaling factor of $\frac{9}{10}M$ was applied to the specialized (task-specific) kernels. Independent learning and full pooling results were obtained by finding the SVM solutions on each data set in isolation and then by finding an SVM on the pooled data from all tasks. The parameter C was chosen based on performance on a validation set. For adaptive pooling, α values were also explored from 0 to e^9. The C value which resulted in the highest AUC on the validation data was used to pick the AUC for each α value. The experiment was repeated 100 times to get the test AUC over different random splits of the data. An advantage for adaptive pooling was evident when $\alpha = 1e5$ and was statistically significant at better than the 5% p-value threshold (using a paired t-test). Figure 3 shows the average test AUC results across 100 folds using independent SVMs, pooling and adaptive pooling for various values of α (after cross-validation only over the value of C for all methods). The Figure reveals an advantage for adaptive pooling compared to full pooling and independent learning.



(a) Average AUC (b) Average AUC zoomed in

Figure 3: Adaptive pooling experiments on the Heart data set. The area under the curve (AUC) for adaptive pooling, pooling and independent SVMs is shown in (a). In (b), a zoomed in version of the plot is shown to focus on the setting with highest average AUC. The value of the regularization parameter C was found using cross-validation for all three methods.

12. Graphical Model Structure Estimation

The multitask sparse discrimination framework is a general tool for large margin classification since most elements of $\hat{\mathbf{s}}$ become vanishingly small (at appropriate settings of ρ or α). This motivates extending the framework to other sparse inference problems including the estimation of graphical model structure which has been explored as an ℓ_1 sparse regression with asymptotic guarantees (Wainwright et al., 2007). The ℓ_1 approach infers a graphical model by learning functions that reconstruct some dimensions given others under sparsity constraints. Assume that we are given T binary vectors $\mathbf{x}_1,\ldots,\mathbf{x}_T$ where $\mathbf{x}_t \in \mathbb{B}^D$ are sampled *iid* from an unknown distribution

$$p(\mathbf{x}) \quad \propto \quad \exp\left(\sum_{m=1}^{D}\eta(m)\mathbf{x}(m) + \sum_{m=1}^{D}\sum_{n=1}^{D}E(m,n)\theta(m,n)\mathbf{x}(m)\mathbf{x}(n)\right).$$

This Ising model is specified by an undirected graph $G=(V,E)$ with D vertices V and edges E, where, without loss of generality, we may assume that $E \in \mathbb{B}^{D\times D}$ is also a binary symmetric adjacency matrix with zero on its diagonal, $\theta \in \mathbb{R}^{D\times D}$ is a symmetric real matrix with zero on its diagonal and $\eta \in \mathbb{R}^D$ is a real vector. The goal of graphical model structure estimation is to recover an estimate $\hat{E}_T$ of the binary matrix E solely from the observations $\mathbf{x}_1,\ldots,\mathbf{x}_T$.

In previous work (Wainwright et al., 2007), a method was provided that achieves $\Pr[\hat{E}_T = E] \to 1$ as $T \to \infty$ by solving independent sparse regression problems as follows,

$$\hat{\theta}_m \;=\; \arg\min_{\theta\in\mathbb{R}^D} \nu \sum_{d\neq m}|\theta(d)| + \sum_{t=1}^{T}\log(1+e^{\sum_{d\neq m}\theta(d)\mathbf{x}_t(d)+\theta(m)}) - \mathbf{x}_t(m)(\sum_{d\neq m}\theta(d)\mathbf{x}_t(d)+\theta(m))$$

for $m=1,\ldots,D$. These tasks reconstruct each dimension from all other dimensions. In other words, the m'task is given $\{\mathbf{x}(1),\ldots,\mathbf{x}(D)\}\setminus\mathbf{x}(m)$ and predicts $\mathbf{x}(m)$. The ℓ_1 sparsity constraint, for appropriate settings of the parameter ν, makes the problem non trivial since only some inputs can be used in the reconstruction. To recover a single consistent set of edges $\hat{E}_T$, the nonzero components of $\hat{\theta}_m$ estimated for various tasks are combined using either an AND or an OR rule. In the AND case, $\hat{E}(m,n)$ is set to 1 if both $\hat{\theta}_m(n)$ is nonzero and $\hat{\theta}_n(m)$ is nonzero. In the OR case, $\hat{E}(m,n)=1$ if either of the terms is nonzero.

The multitask MED approach can potentially circumvent this ad hoc AND/OR step by forcing all sparse predictors to agree on a single undirected edge connectivity matrix E from the outset. The MED approach considers $m=1,\ldots,D$ tasks where the m'th task is given $\mathbf{x}_t$ and must predict $y_{m,t}=2\mathbf{x}_t(m)-1\in\pm 1$ as a classification output. We assume the following predictive distribution:

$$p(y|m,\mathbf{x},\theta,b,\mathbf{s}) \quad\propto\quad \exp\left(\frac{y}{2}\sum_{d\neq m}\mathbf{s}(m,d)\mathbf{x}(d)\theta(m,d)+\mathbf{b}(m)\right).$$

The MED model Θ contains a matrix $\theta\in\mathbb{R}^{D\times D}$ with its diagonal forced to zero. In addition, it contains a binary matrix $\mathbf{s}\in\mathbb{B}^{D\times D}$ (again with its diagonal forced to zero) and finally a scalar vector $\mathbf{b}\in\mathbb{R}^D$. The standard Gaussian priors are applied to the model parameters in $P(\Theta)$ except for the $\mathbf{s}$ variable which obtains a Bernoulli prior over its binary entries and (for sufficiently large α) to encourage its sparsity. In addition, we a priori enforce the symmetry $\mathbf{s}(m,d)=\mathbf{s}(d,m)$. This ensures that, if input $\mathbf{x}(d)$ is used for the prediction of $\mathbf{x}(m)$, $\mathbf{x}(m)$ can also be used for predicting $\mathbf{x}(d)$. However, symmetry is not enforced on the θ parameters which permits us to learn different linear relationships once a consistent dependency structure is determined. Thus, consistency of the edges used by the sparse prediction is enforced up-front in a multitask setting instead of resorting to a post-processing (i.e., the AND or OR steps) as in the previous approach which independently learns D regression functions.

The MED framework computes the partition function by integrating the following:

$$\begin{aligned} Z(\boldsymbol{\lambda}) &= \int p(\Theta)\exp\left(\sum_{m=1}^{D}\sum_{t=1}^{T}\lambda_{m,t}y_{m,t}\left(\sum_{d\neq m}\mathbf{s}(m,d)\mathbf{x}_t(d)\theta(m,d)+\mathbf{b}(m)\right)-\lambda_{m,t}\right)d\Theta \\ &= e^{\sum_m\frac{\sigma^2}{2}\left(\sum_t\lambda_{m,t}y_{m,t}\right)^2-\sum_t\lambda_{m,t}}\sum_{\mathbf{s}}p(\mathbf{s})e^{\frac{1}{2}\sum_{m=1}^{D}\sum_{d=m+1}^{D}\mathbf{s}(m,d)\left(\left(\sum_t\lambda_{m,t}y_{m,t}\mathbf{x}_t(d)\right)^2+\left(\sum_t\lambda_{d,t}y_{d,t}\mathbf{x}_t(m)\right)^2\right)}. \end{aligned}$$

Taking $\sigma\to\infty$ and $J(\boldsymbol{\lambda})=-\log(Z(\boldsymbol{\lambda}))$ produces (up to an additive constant) the dual program:

$$\left\{\begin{array}{l} \max_{\boldsymbol{\lambda}}\ \sum_{m=1}^{D}\sum_{t=1}^{T}\lambda_{m,t}-\sum_{m=1}^{D}\sum_{d=m+1}^{D}\log\left(\alpha+e^{\frac{1}{2}\left(\left(\sum_t\lambda_{m,t}y_{m,t}\mathbf{x}_t(d)\right)^2+\left(\sum_t\lambda_{d,t}y_{d,t}\mathbf{x}_t(m)\right)^2\right)}\right) \\ \text{s.t. } 0\leq\lambda_{m,t}\leq C\ \ \forall m,t \text{ and } \sum_{t=1}^{T}y_{m,t}\lambda_{m,t}=0\ \ \forall m. \end{array}\right.$$

The objective function can be written as

$$\max_{\boldsymbol{\lambda}\in\Lambda}\boldsymbol{\lambda}^\top\mathbf{1}-\sum_{m=1}^{D}\sum_{d=m+1}^{D}\log\left(\alpha+e^{\frac{1}{2}\boldsymbol{\lambda}^\top H_{m,d}\boldsymbol{\lambda}}\right)$$

where the $H_{m,d}\in\mathbb{R}^{DT\times DT}$ matrices for $d>m\in\{1,\ldots,D\}$ are defined element-wise as

$$H_{m,d}([n,t],[o,\tau]) \quad=\quad y_{m,t}y_{m,\tau}\mathbf{x}_t(d)\mathbf{x}_\tau(d)\delta_{m=n=o}+y_{d,t}y_{d,\tau}\mathbf{x}_t(m)\mathbf{x}_\tau(m)\delta_{d=n=o}.$$

It is easy to maximize the objective using sequential quadratic programming which gives an estimate for $\boldsymbol{\lambda}$. The prediction rule is then $\hat{y} = \arg\max_y \mathrm{E}_{p(\Theta|\mathcal{D})}[\log p(y|m, \mathbf{x}, \Theta, \mathbf{b}, \mathbf{s})]$ which involves the sparse variable $\mathbf{s}$. These switch configurations essentially identify the network structure and are obtained from expected $\mathbf{s}(m,d)$ values under the posterior $p(\Theta|\mathcal{D})$ as follows:

$$\hat{\mathbf{s}}(m,d) \quad = \quad \frac{1}{1 + \alpha \exp\left(-\frac{1}{2}\left(\left(\sum_t \lambda_{m,t} y_{m,t} \mathbf{x}_t(d)\right)^2 + \left(\sum_t \lambda_{d,t} y_{d,t} \mathbf{x}_t(m)\right)^2\right)\right)}.$$

For large α, many entries of $\hat{\mathbf{s}}$ are driven towards small values as MED resembles an ℓ_1 regularizer. MED produces sparsity although only in a probabilistic sense since coefficients do not strictly go to zero but typically shrink to small values. The matrix $\hat{\mathbf{s}}$ represents MED's estimate of the unknown adjacency matrix E in the original graphical model.

To test the accuracy of the method, the scalar values of $\hat{\mathbf{s}}$ are used as scalar classification predictions for the presence or absence of an edge. Given the true graph, these predictions are straightforward to evaluate using the AUC. Experimental results with synthetic data are obtained by generating random graphs and obtaining samples from them according to the Ising model above (Wainwright et al., 2007). In Figure 4, the mean area under the curve (MAUC) is reported for the MED technique as well as the independent ℓ_1 regularized regressions with an AND and an OR step. Multiple settings of the regularization parameters are shown in the plot as the value of the regularization ν is explored in the original method (for both the AND and OR setting) and the values of C and α are explored in the proposed method. Since the ℓ_1 regularization method (Wainwright et al., 2007) is asymptotically correct, the experiments here focus on the small sample regime. From ten random graphical models over 5 nodes, 60 samples were drawn using Monte Carlo methods and the average area under the curve for the various methods was reported. To fairly compare results using an AUC measure, we did not only use the support found by the ℓ_1 regularized method but also considered all possible thresholds on the ℓ_1 solution. More specifically, the min or the max operators were first used to symmetrize the absolute value of the regression weights recovered by ℓ_1 regularization. These non-negative scalars were then used in the graphical model to allow all operating points on the receiver operator characteristic to be explored. This can only improve the performance of the ℓ_1 regularization method in terms of AUC (a binary estimate of edges followed by an AND or an OR step can only obtain lower AUC). Despite this, the proposed method [10] performs significantly better possibly due to the explicit symmetry in the edge estimation. These preliminary experiments motivate large scale future empirical work.

13. Discussion

A multitask learning framework was developed for support vector machines and large-margin linear classifiers. Each task-specific classifier is estimated to solve its own problem yet all tasks have to share a common representation $\mathbf{s}$. This common representation included sparse feature selection and conic kernel combination. This common representation couples tasks to go beyond standard tabula rasa learning. To compute the coupled linear models, we applied the maximum entropy discrimination framework which produces support vector machines that share a common sparse representation. The framework combines classification problems non-trivially in a convex dual-space optimization. We presented a simple sequential quadratic programming approach for solving the

10. Code available at `www.cs.columbia.edu/~jebara/code/multisparse/`.

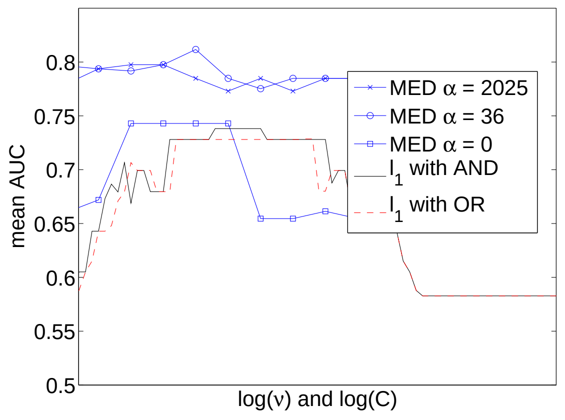


Figure 4: Graphical model structure estimation from data sampled from Ising models. The average area under the curve is shown for ten random models. The proposed method is evaluated across various values of C and α and compared to ℓ_1 regularized logistic regression method across various values of ν with both AND and OR symmetrization.

dual optimization for both multitask feature selection and multitask kernel selection problems. We interleave bound computations with standard SVM updates (either using quadratic programming or, preferably, nearly linear-time modern SVM solvers). In addition, the extensions to adaptive pooling, sparse regression and graphical model reconstruction were illustrated. The MED multitask framework potentially allows flexible exploration of sparsity structure over different groups of variables and is reminiscent of Group Lasso methods (Yuan and Lin, 2006; Bach, 2008). Experiments on real world data sets show that MED multitask learning is advantageous over single-task or tabula rasa learning.

In future work, it would be interesting to investigate theoretical generalization guarantees for multitask sparse MED. This may involve exploiting PAC-Bayesian model selection methods or online mistake bound methods (McAllester, 1999; Langford and Shawe-Taylor, 2002; Long and Wu, 2004) which have already given generalization arguments for the single-task MED approach. Since generalization guarantees in multitask settings have already been provided for other algorithms (Ando and Zhang, 2005; Maurer, 2006, 2009), this may be a fruitful line of work. Finally, it may be useful to explore methods for automatically estimating the hyper-parameters in the MED framework such as α which, as in classical Bayesian approaches, might be handled via optimization or integration rather than cumbersome cross-validation.

Acknowledgments

The author is grateful to the anonymous referees, P. Shivaswamy, T. Jaakkola, M. Meila and D. Rosenberg for valuable comments. The author also acknowledges P. Shivaswamy for help with experiments involving adaptive pooling and P. Ravikumar and M. Wainwright for sharing code to perform ℓ_1-regularized graphical model structure estimation. This work was funded in part by the National Science Foundation (Grant IIS-0347499), the Office of Naval Research (Grant N000140710507) and the Department of Homeland Security (Grant N66001-09-C-0080).

Appendix A. Bounding the Logistic-Quadratic Function

Theorem 1 *For all* $\mathbf{u} \in \mathbb{R}^D$, $\log\left(\alpha + \exp\left(\frac{\mathbf{u}^\top \mathbf{u}}{2}\right)\right)$ *is bounded above by*

$$\log\left(\alpha + \exp\left(\frac{\mathbf{v}^\top\mathbf{v}}{2}\right)\right) + \frac{\mathbf{v}^\top(\mathbf{u}-\mathbf{v})}{1+\alpha\exp(-\frac{\mathbf{v}^\top\mathbf{v}}{2})} + \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top\left(I + \mathcal{G}\mathbf{v}\mathbf{v}^\top\right)(\mathbf{u}-\mathbf{v})$$

for the scalar term $\mathcal{G} = \frac{1}{2}\tanh(\frac{1}{2}\log(\alpha\exp(-\mathbf{v}^\top\mathbf{v}/2)))/\log(\alpha\exp(-\mathbf{v}^\top\mathbf{v}/2))$. *The bound holds for any* $\alpha \geq 0, \mathbf{v} \in \mathbb{R}^D$ *and strict equality is achieved when* $\mathbf{u} = \mathbf{v}$.

Proof The proof proceeds by first making the bound achieve (tangential) equality at $\mathbf{u} = \mathbf{v}$. It then applies a previously known bound on the logistic function using convexity arguments. The logistic-quadratic function $g(\mathbf{u})$ and the general quadratic function $q(\mathbf{u})$ are defined as

$$\begin{aligned} g(\mathbf{u}) &= \log\left(\alpha + \exp\left(\frac{\mathbf{u}^\top\mathbf{u}}{2}\right)\right), \\ q(\mathbf{u}) &= c + \mathbf{b}^\top(\mathbf{u}-\mathbf{v}) + \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top A(\mathbf{u}-\mathbf{v}). \end{aligned}$$

The quadratic function $q(\mathbf{u})$ is parametrized by a scalar $c \geq 0$, a vector $\mathbf{b} \in \mathbb{R}^D$ and a positive semi-definite matrix $A \in \mathbb{R}^{D\times D}$. These parameters must be selected to ensure $q(\mathbf{u}) \geq g(\mathbf{u})$ for all $\mathbf{u} \in \mathbb{R}^D$. Furthermore, the theorem requires that $g(\mathbf{v}) = q(\mathbf{v})$. This determines the additive constant $c = \log\left(\alpha + \exp\left(\frac{\mathbf{v}^\top\mathbf{v}}{2}\right)\right)$. Since equality is achieved at $\mathbf{u} = \mathbf{v}$, the gradients must be equal there as well, in other words $\left.\frac{\partial g(\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{v}} = \left.\frac{\partial q(\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{v}}$. This determines that $\mathbf{b} = \exp\left(\frac{\mathbf{v}^\top\mathbf{v}}{2}\right)/(\alpha + \exp\left(\frac{\mathbf{v}^\top\mathbf{v}}{2}\right))\mathbf{v}$. Otherwise, the functions cross at $\mathbf{u} = \mathbf{v}$ which violates the bound. Inserting these values for $\mathbf{b}$ and c into the quadratic form for $q(\mathbf{u})$ reveals that A must be chosen such that $\frac{1}{2}(\mathbf{u}-\mathbf{v})^\top A(\mathbf{u}-\mathbf{v})$ is greater than or equal to

$$\log\left(\frac{\alpha + \exp(\mathbf{u}^\top\mathbf{u}/2)}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\right) - \frac{\exp(\mathbf{v}^\top\mathbf{v}/2)}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\mathbf{v}^\top(\mathbf{u}-\mathbf{v}).$$

Consider the choice for A suggested by the theorem to prove that it satisfies this requirement

$$A = I + \frac{\tanh(\frac{1}{2}\log(\alpha\exp(-\mathbf{v}^\top\mathbf{v}/2)))}{2\log(\alpha\exp(-\mathbf{v}^\top\mathbf{v}/2))}\mathbf{v}\mathbf{v}^\top.$$

Multiply A appropriately to obtain the desired expression

$$\frac{1}{2}(\mathbf{u}-\mathbf{v})^\top A(\mathbf{u}-\mathbf{v}) = \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top\left(I + \frac{\tanh(\frac{1}{2}\log(\varphi))}{2\log(\varphi)}\mathbf{v}\mathbf{v}^\top\right)(\mathbf{u}-\mathbf{v}),$$

where, for brevity, we define the scalar $\varphi = \alpha\exp(-\mathbf{v}^\top\mathbf{v}/2)$. Rewrite the right hand side as

$$\begin{aligned} \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top A(\mathbf{u}-\mathbf{v}) &= \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top\left(I + \frac{\tanh(\frac{1}{2}\log(\varphi))}{2\log(\varphi)}\mathbf{v}\mathbf{v}^\top\right)(\mathbf{u}-\mathbf{v}) - \frac{\tanh(\frac{1}{2}\log\varphi)}{2}\mathbf{v}^\top(\mathbf{u}-\mathbf{v}) \\ &\quad + \frac{\varphi-1}{2}\frac{1}{\varphi+1}\mathbf{v}^\top(\mathbf{u}-\mathbf{v}) \end{aligned}$$

while noting that $\tanh(\frac{1}{2}\log\varphi) = (\varphi-1)/(\varphi+1)$. The right hand side further simplifies into

$$\frac{1}{2}(\mathbf{u}-\mathbf{v})^\top A(\mathbf{u}-\mathbf{v}) \quad = \quad \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top(\mathbf{u}-\mathbf{v}) + \frac{\tanh(\frac{1}{2}\log\varphi)}{4\log\varphi}(\chi^2 - (\log\varphi)^2) + \frac{\varphi-1}{2}z$$

where, for brevity, we have defined the following

$$\begin{aligned} z &= \frac{1}{\varphi+1}\mathbf{v}^\top(\mathbf{u}-\mathbf{v}), \\ \chi &= \mathbf{v}^\top(\mathbf{u}-\mathbf{v}) - \log\varphi \;=\; (\varphi+1)z - \log\varphi. \end{aligned}$$

Recall the following inequality (Jaakkola and Jordan, 2000) which holds for any choice of $\xi \in \mathbb{R}$:

$$\log\left(\exp\left(-\frac{\xi}{2}\right) + \exp\left(\frac{\xi}{2}\right)\right) + \frac{\tanh(\frac{\xi}{2})}{4\xi}(\chi^2 - \xi^2) \quad \geq \quad \log\left(\exp\left(-\frac{\chi}{2}\right) + \exp\left(\frac{\chi}{2}\right)\right).$$

Choose $\xi = \log\varphi$ (or, equivalently, $\xi = -\log\varphi$) and rewrite the bound as

$$\frac{\tanh(\frac{1}{2}\log\varphi)}{4\log\varphi}(\chi^2 - (\log\varphi)^2) \quad \geq \quad \log\left(\exp\left(-\frac{\chi}{2}\right) + \exp\left(\frac{\chi}{2}\right)\right) - \log(\varphi^{\frac{1}{2}} + \varphi^{-\frac{1}{2}}).$$

Applying this bound in the formula involving the A matrix and rearranging yields

$$\begin{aligned} \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top A(\mathbf{u}-\mathbf{v}) &\geq \frac{\|\mathbf{u}-\mathbf{v}\|^2}{2} + \log\left(\exp\left(-\frac{\chi}{2}\right) + \exp\left(\frac{\chi}{2}\right)\right) - \log(\varphi^{\frac{1}{2}} + \varphi^{-\frac{1}{2}}) + \frac{\varphi-1}{2}z \\ &= \frac{\|\mathbf{u}-\mathbf{v}\|^2}{2} + \log\left(\exp(-z+\log\varphi) + \exp(\varphi z)\right) - \log(\varphi+1) \\ &= \frac{\|\mathbf{u}-\mathbf{v}\|^2}{2} + \log\left(\frac{\varphi}{\varphi+1}\exp(-z) + \frac{1}{\varphi+1}\exp(\varphi z)\right) \\ &= \frac{\|\mathbf{u}-\mathbf{v}\|^2}{2} + \log\left(\frac{\varphi}{\varphi+1}\exp\left(-\frac{\mathbf{v}^\top(\mathbf{u}-\mathbf{v})}{\varphi+1}\right) + \frac{1}{\varphi+1}\exp\left(\frac{\varphi\mathbf{v}^\top(\mathbf{u}-\mathbf{v})}{\varphi+1}\right)\right) \\ &= \frac{\|\mathbf{u}-\mathbf{v}\|^2}{2} + \log\left(\frac{\varphi + \exp(\mathbf{v}^\top(\mathbf{u}-\mathbf{v}))}{\varphi+1}\right) - \frac{\mathbf{v}^\top(\mathbf{u}-\mathbf{v})}{\varphi+1} \\ &= \frac{\|\mathbf{u}-\mathbf{v}\|^2}{2} + \log\left(\frac{\alpha + \exp(-\mathbf{v}^\top\mathbf{v}/2 + \mathbf{v}^\top\mathbf{u})}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\right) - \frac{\exp(\mathbf{v}^\top\mathbf{v}/2)}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\mathbf{v}^\top(\mathbf{u}-\mathbf{v}) \\ &\geq \frac{\|\mathbf{u}-\mathbf{v}\|^2}{2} + \log\left(\frac{\alpha\exp(-\frac{1}{2}\|\mathbf{u}-\mathbf{v}\|^2) + \exp(-\mathbf{v}^\top\mathbf{v}/2 + \mathbf{v}^\top\mathbf{u})}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\right) \\ &\quad - \frac{\exp(\mathbf{v}^\top\mathbf{v}/2)}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\mathbf{v}^\top(\mathbf{u}-\mathbf{v}). \end{aligned}$$

In the last line, we use the fact that $1 \geq \exp(-\frac{1}{2}\|\mathbf{u}-\mathbf{v}\|^2)$. Absorbing the $\frac{1}{2}\|\mathbf{u}-\mathbf{v}\|^2$ term into the logarithm multiplicatively gives the desired inequality

$$\frac{1}{2}(\mathbf{u}-\mathbf{v})^\top A(\mathbf{u}-\mathbf{v}) \quad \geq \quad \log\left(\frac{\alpha + \exp(\mathbf{u}^\top\mathbf{u}/2)}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\right) - \frac{\exp(\mathbf{v}^\top\mathbf{v}/2)}{\alpha + \exp(\mathbf{v}^\top\mathbf{v}/2)}\mathbf{v}^\top(\mathbf{u}-\mathbf{v}).$$

■

Appendix B. Convergence of Sequential Quadratic Programming

Theorem 2 *Algorithm 1 finds a* $\tilde{\boldsymbol{\lambda}} \in \Lambda$ *achieving* $J(\tilde{\boldsymbol{\lambda}}) \geq (1-\varepsilon)\max_{\boldsymbol{\lambda}\in\Lambda} J(\boldsymbol{\lambda})$ *where*

$$J(\boldsymbol{\lambda}) = D\log(\alpha+1) - \sum_{d=1}^{D} \log\left(\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}^\top H_d \boldsymbol{\lambda}\right)\right) + \boldsymbol{\lambda}^\top \mathbf{1}$$
$$\text{s.t. } \boldsymbol{\lambda} \in \Lambda = \begin{cases} 0 \leq \lambda_{m,t} \leq C, & t = 1,\ldots,T_m,\ m = 1,\ldots,M \\ \sum_{t=1}^{T_m} y_{m,t}\lambda_{m,t} = 0, & m = 1,\ldots,M \end{cases}$$

in at most $\left\lceil \frac{\log(1/\varepsilon)}{\log\left(\min\left(1+\frac{1}{\alpha},2\right)\right)} \right\rceil$ *iterations for any* $\alpha \geq 0$ *and* $\varepsilon \in (0,1)$.

Proof Sequential quadratic programing is used to approximate $\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}\in\Lambda} J(\boldsymbol{\lambda})$. Given a current setting $\boldsymbol{\lambda}_i$ at iteration i, Theorem 1 obtains a variational quadratic bound on $J(\boldsymbol{\lambda})$ as:

$$\begin{aligned} L_i(\boldsymbol{\lambda}) \;=\; & D\log(\alpha+1) - \sum_{d=1}^{D} \log\left(\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)\right) - \sum_{d=1}^{D} \frac{\exp(\frac{\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i}{2})}{\alpha + \exp(\frac{\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i}{2})} \boldsymbol{\lambda}_i^\top H_d (\boldsymbol{\lambda} - \boldsymbol{\lambda}_i) \\ & -\frac{1}{2}(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)^\top \left(\sum_{d=1}^{D} \frac{\tanh(\frac{1}{2}\log(\alpha\exp(-\frac{\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i}{2})))}{2\log(\alpha\exp(-\frac{\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i}{2})} H_d \boldsymbol{\lambda}_i \boldsymbol{\lambda}_i^\top H_d + H_d\right)(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i) + \boldsymbol{\lambda}^\top \mathbf{1}. \end{aligned}$$

The bound satisfies $L_i(\boldsymbol{\lambda}) \leq J(\boldsymbol{\lambda})$ for all $\boldsymbol{\lambda}$ and equality is achieved when $\boldsymbol{\lambda} = \boldsymbol{\lambda}_i$. Next, we will find an upper bound $J(\boldsymbol{\lambda}) \leq U_i(\boldsymbol{\lambda})$. We first form a component of $U(\boldsymbol{\lambda})$ called $U^d(\boldsymbol{\lambda})$ that upper bounds the following component of the objective function

$$J^d(\boldsymbol{\lambda}) \;=\; -\log\left(\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}^\top H_d \boldsymbol{\lambda}\right)\right).$$

Apply Jensen's inequality for any choice of the scalar variational parameter $\zeta_d \in [0,1]$ to get

$$J^d(\boldsymbol{\lambda}) \;\leq\; -\zeta_d \log\left(\frac{\alpha}{\zeta_d}\right) - (1-\zeta_d)\log\left(\frac{\exp\left(\frac{1}{2}\boldsymbol{\lambda}^\top H_d \boldsymbol{\lambda}\right)}{(1-\zeta_d)}\right).$$

Setting the variational parameter as $\zeta_d = \alpha\left(\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)\right)^{-1}$ produces the bound $J^d(\boldsymbol{\lambda}) \leq$

$$-\log\left(\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)\right) + \frac{\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)}{\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)} \frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i - \frac{\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)}{\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)} \frac{1}{2}\boldsymbol{\lambda}^\top H_d \boldsymbol{\lambda}.$$

Repeating the above for $d = 1,\ldots,D$ terms produces the overall variational upper bound

$$\begin{aligned} U_i(\boldsymbol{\lambda}) \;=\; & D\log(\alpha+1) - \sum_{d=1}^{D} \log\left(\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)\right) + \sum_{d=1}^{D} \frac{\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)}{\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)} \frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i \\ & - \sum_{d=1}^{D} \frac{\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)}{\alpha + \exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i\right)} \frac{1}{2}\boldsymbol{\lambda}^\top H_d \boldsymbol{\lambda} + \boldsymbol{\lambda}^\top \mathbf{1}. \end{aligned}$$

Clearly, $J(\boldsymbol{\lambda}) \leq U_i(\boldsymbol{\lambda})$ and equality is achieved when $\boldsymbol{\lambda} = \boldsymbol{\lambda}_i$. Thus, we have an upper quadratic bound and a lower quadratic bound which sandwich the objective as $L_i(\boldsymbol{\lambda}) \leq J(\boldsymbol{\lambda}) \leq U_i(\boldsymbol{\lambda})$. Both

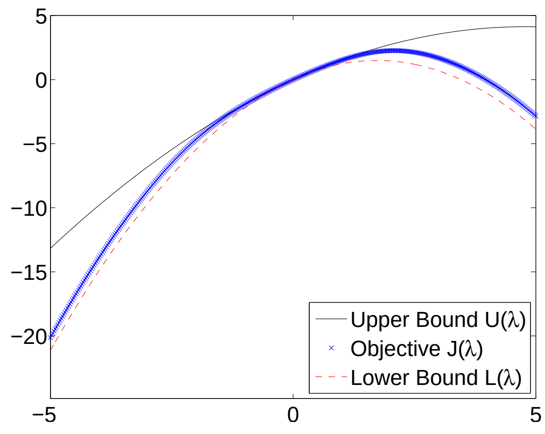


Figure 5: Upper and lower quadratic bounds on the objective function.

bounds are tight at $\boldsymbol{\lambda}_i$, in other words, $L_i(\boldsymbol{\lambda}_i) = J(\boldsymbol{\lambda}_i) = U_i(\boldsymbol{\lambda}_i)$. Figure 5 depicts the bounds. The algorithm initializes $\boldsymbol{\lambda}_0 = \mathbf{0}$ and updates via $\boldsymbol{\lambda}_{i+1} = \arg\max_{\boldsymbol{\lambda}\in\Lambda} L_i(\boldsymbol{\lambda})$ for each iteration i. Apply Lemma 3 which provides a value of $\kappa = \max(\alpha+1,2)$ such that the following holds

$$\sup_{\boldsymbol{\lambda}\in\Lambda} L_i(\boldsymbol{\lambda}) - L_i(\boldsymbol{\lambda}_i) \quad \geq \quad \frac{1}{\kappa}\sup_{\boldsymbol{\lambda}\in\Lambda}\left(U_i(\boldsymbol{\lambda}) - U_i(\boldsymbol{\lambda}_i)\right).$$

Since $L_i(\boldsymbol{\lambda}_i) = J(\boldsymbol{\lambda}_i) = U_i(\boldsymbol{\lambda}_i)$, $J(\boldsymbol{\lambda}_{i+1}) \geq \sup_{\boldsymbol{\lambda}\in\Lambda} L_i(\boldsymbol{\lambda})$ and $\sup_{\boldsymbol{\lambda}\in\Lambda} U_i(\boldsymbol{\lambda}) \geq J(\boldsymbol{\lambda}^*)$, we have

$$J(\boldsymbol{\lambda}_{i+1}) - J(\boldsymbol{\lambda}_i) \quad \geq \quad \frac{1}{\kappa}\left(J(\boldsymbol{\lambda}^*) - J(\boldsymbol{\lambda}_i)\right).$$

Rearrange the inequality as follows

$$J(\boldsymbol{\lambda}_{i+1}) - J(\boldsymbol{\lambda}^*) \quad \geq \quad \left(1-\frac{1}{\kappa}\right)\left(J(\boldsymbol{\lambda}_i) - J(\boldsymbol{\lambda}^*)\right).$$

Iterate the above inequality starting at $i = 0$ to obtain

$$J(\boldsymbol{\lambda}_i) - J(\boldsymbol{\lambda}^*) \quad \geq \quad \left(1-\frac{1}{\kappa}\right)^i\left(J(\boldsymbol{\lambda}_0) - J(\boldsymbol{\lambda}^*)\right).$$

Since the initialization used was $J(\boldsymbol{\lambda}_0) = J(\mathbf{0}) = 0$, the above simplifies as

$$J(\boldsymbol{\lambda}_i) \quad \geq \quad \left(1-\left(1-\frac{1}{\kappa}\right)^i\right)J(\boldsymbol{\lambda}^*).$$

Therefore, a solution that is within a relative multiplicative factor of ε implies that

$$\begin{aligned}\varepsilon &= \left(1-\frac{1}{\kappa}\right)^i = \left(1-\frac{1}{\max(\alpha+1,2)}\right)^i \\ \log(1/\varepsilon) &= i\log\left(\min\left(1+\frac{1}{\alpha},2\right)\right).\end{aligned}$$

Therefore, the number of iterations i required is at most $\left\lceil\frac{\log(1/\varepsilon)}{\log\left(\min\left(1+\frac{1}{\alpha},2\right)\right)}\right\rceil$. ■

Lemma 3 *The functions*

$$L_i(\boldsymbol{\lambda}) = D\log(\alpha+1) - \sum_{d=1}^{D}\log\left(\alpha+\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i\right)\right) - \sum_{d=1}^{D}\frac{\exp(\frac{\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i}{2})}{\alpha+\exp(\frac{\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i}{2})}\boldsymbol{\lambda}_i{}^\top H_d(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)$$
$$-\frac{1}{2}(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)^\top\left(\sum_{d=1}^{D}\mathcal{G}_d H_d\boldsymbol{\lambda}_i\boldsymbol{\lambda}_i{}^\top H_d + H_d\right)(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)+\boldsymbol{\lambda}^\top\mathbf{1},$$
$$U_i(\boldsymbol{\lambda}) = D\log(\alpha+1) - \sum_{d=1}^{D}\log\left(\alpha+\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i\right)\right) + \sum_{d=1}^{D}\frac{\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i\right)}{\alpha+\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i\right)}\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i$$
$$-\sum_{d=1}^{D}\frac{\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i\right)}{\alpha+\exp\left(\frac{1}{2}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i\right)}\frac{1}{2}\boldsymbol{\lambda}^\top H_d\boldsymbol{\lambda}+\boldsymbol{\lambda}^\top\mathbf{1}$$

for $\mathcal{G}_d = \frac{1}{2}\tanh(\frac{1}{2}\log(\alpha\exp(-\frac{\tilde{\boldsymbol{\lambda}}^\top H_d\tilde{\boldsymbol{\lambda}}}{2})))/\log(\alpha\exp(-\frac{\tilde{\boldsymbol{\lambda}}^\top H_d\tilde{\boldsymbol{\lambda}}}{2})$ *and* $H_d \succeq \mathbf{0}$ *for* $d=1,\ldots,D$ *satisfy*

$$\sup_{\boldsymbol{\lambda}\in\Lambda}(L_i(\boldsymbol{\lambda})-L_i(\boldsymbol{\lambda}_i)) \geq \frac{1}{\max(\alpha+1,2)}\sup_{\boldsymbol{\lambda}\in\Lambda}(U_i(\boldsymbol{\lambda})-U_i(\boldsymbol{\lambda}_i))$$

where

$$\Lambda = \begin{cases} 0\leq\lambda_{m,t}\leq C, & t=1,\ldots,T_m,\ m=1,\ldots,M \\ \sum_{t=1}^{T_m} y_{m,t}\lambda_{m,t}=0, & m=1,\ldots,M. \end{cases}$$

Proof Rewrite the functions as follows

$$L_i(\boldsymbol{\lambda})-L_i(\boldsymbol{\lambda}_i) = -\frac{1}{2}(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)^\top\Phi(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)-(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)^\top\boldsymbol{\mu}$$
$$U_i(\boldsymbol{\lambda})-U_i(\boldsymbol{\lambda}_i) = -\frac{1}{2}(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)^\top\Psi(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)-(\boldsymbol{\lambda}-\boldsymbol{\lambda}_i)^\top\boldsymbol{\mu}$$

where

$$\Phi = \sum_{d=1}^{D}\left(\mathcal{G}_d H_d\boldsymbol{\lambda}_i\boldsymbol{\lambda}_i{}^\top+I\right)H_d$$
$$\Psi = \sum_{d=1}^{D}\frac{\exp(\frac{1}{2}\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i)}{\alpha+\exp(\frac{1}{2}\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i)}H_d$$
$$\boldsymbol{\mu} = \sum_{d=1}^{D}\frac{\exp(\frac{\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i}{2})}{\alpha+\exp(\frac{\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i}{2})}\boldsymbol{\lambda}_i{}^\top H_d-\mathbf{1}.$$

Since $\mathrm{tr}(A)I\succeq A$ for matrices $A\succeq\mathbf{0}$, the following holds in the Loewner ordering sense

$$\Phi \preceq \sum_{d=1}^{D}\left(\mathcal{G}_d\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i+1\right)H_d.$$

Rewrite this bound as $\Phi\preceq\sum_d\phi_d H_d$ and rewrite $\Psi=\sum_d\psi_d H_d$. Consider the ratio

$$\frac{\phi_d}{\psi_d} = \frac{\alpha+\exp(\frac{1}{2}\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i)}{\exp(\frac{1}{2}\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i)}\left(\frac{\tanh(\frac{1}{2}\log(\alpha\exp(-\frac{\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i}{2})))}{2\log(\alpha\exp(-\frac{\boldsymbol{\lambda}_i{}^\top H_d\boldsymbol{\lambda}_i}{2})}\boldsymbol{\lambda}_i^\top H_d\boldsymbol{\lambda}_i+1\right).$$

Define $\zeta_d = \frac{1}{2}\boldsymbol{\lambda}_i^\top H_d \boldsymbol{\lambda}_i$ and rewrite the ratio as

$$\frac{\phi_d}{\psi_d} = \frac{\alpha + \exp(\zeta_d)}{\exp(\zeta_d)} \left(\frac{\tanh(\frac{1}{2}\log(\alpha \exp(-\zeta_d)))}{\log(\alpha \exp(-\zeta_d))} \zeta_d + 1 \right).$$

It is easy to verify that this ratio is maximized when $\zeta_d \to \infty$ if $\alpha \leq 1$ and when $\zeta_d = 0$ when $\alpha > 1$. This reveals that the ratio is bounded as $\frac{\phi_d}{\psi_d} \leq \kappa$ where $\kappa = \max(\alpha+1, 2)$. Therefore, we can rewrite

$$\Phi \preceq \sum_{d=1}^{D} \phi_d H_d \preceq \sum_{d=1}^{D} \kappa \psi_d H_d = \kappa \Psi.$$

Recall the primal maximization problems of interest: $\mathbf{P}_L = \sup_{\boldsymbol{\lambda} \in \Lambda} L_i(\boldsymbol{\lambda}) - L_i(\boldsymbol{\lambda}_i)$ and $\mathbf{P}_U = \sup_{\boldsymbol{\lambda} \in \Lambda} U_i(\boldsymbol{\lambda}) - U_i(\boldsymbol{\lambda}_i)$. The constraints $\boldsymbol{\lambda} \in \Lambda$ can be summarized by linear inequalities $A\boldsymbol{\lambda} \leq \mathbf{b}$ for some A and $\mathbf{b}$. Apply the change of variables $\mathbf{z} = \boldsymbol{\lambda} - \boldsymbol{\lambda}_i$. The constraint $A(\mathbf{z} + \boldsymbol{\lambda}_i) \leq \mathbf{b}$ simplifies into $A\mathbf{z} \leq \tilde{\mathbf{b}}$ where $\tilde{\mathbf{b}} = \mathbf{b} - A\boldsymbol{\lambda}_i$. Since $\boldsymbol{\lambda}_i \in \Lambda$ is a feasible solution (which is true by construction), it is easy to show that $\tilde{\mathbf{b}} \geq \mathbf{0}$. We obtain the following equivalent primal optimization problems

$$\mathbf{P}_L = \sup_{A\mathbf{z} \leq \tilde{\mathbf{b}}} -\frac{1}{2}\mathbf{z}^\top \Phi \mathbf{z} - \mathbf{z}^\top \boldsymbol{\mu}, \quad \mathbf{P}_Z = \sup_{A\mathbf{z} \leq \tilde{\mathbf{b}}} -\frac{\kappa}{2}\mathbf{z}^\top \Psi \mathbf{z} - \mathbf{z}^\top \boldsymbol{\mu}, \quad \mathbf{P}_U = \sup_{A\mathbf{z} \leq \tilde{\mathbf{b}}} -\frac{1}{2}\mathbf{z}^\top \Psi \mathbf{z} - \mathbf{z}^\top \boldsymbol{\mu}.$$

The respective dual problems to the above are

$$\begin{aligned}
\mathbf{D}_L &= \inf_{\mathbf{y} \geq \mathbf{0}} \frac{1}{2}\mathbf{y}^\top A \Phi^{-1} A^\top \mathbf{y} + \mathbf{y}^\top A \Phi^{-1} \boldsymbol{\mu} + \mathbf{y}^\top \tilde{\mathbf{b}} + \frac{1}{2}\boldsymbol{\mu}^\top \Phi^{-1} \boldsymbol{\mu} \\
\mathbf{D}_Z &= \inf_{\mathbf{y} \geq \mathbf{0}} \frac{1}{\kappa}\frac{1}{2}\mathbf{y}^\top A \Psi^{-1} A^\top \mathbf{y} + \frac{1}{\kappa}\mathbf{y}^\top A \Psi^{-1} \boldsymbol{\mu} + \mathbf{y}^\top \tilde{\mathbf{b}} + \frac{1}{\kappa}\frac{1}{2}\boldsymbol{\mu}^\top \Psi^{-1} \boldsymbol{\mu} \\
\mathbf{D}_U &= \inf_{\mathbf{y} \geq \mathbf{0}} \frac{1}{2}\mathbf{y}^\top A \Psi^{-1} A^\top \mathbf{y} + \mathbf{y}^\top A \Psi^{-1} \boldsymbol{\mu} + \mathbf{y}^\top \tilde{\mathbf{b}} + \frac{1}{2}\boldsymbol{\mu}^\top \Psi^{-1} \boldsymbol{\mu}.
\end{aligned}$$

Due to strong duality, $\mathbf{P}_L = \mathbf{D}_L$, $\mathbf{P}_Z = \mathbf{D}_Z$ and $\mathbf{P}_U = \mathbf{D}_U$. Apply the bound $\Phi \preceq \kappa\Psi$ as follows

$$\begin{aligned}
\mathbf{P}_L &= \sup_{A\mathbf{z} \geq \tilde{\mathbf{b}}} -\frac{1}{2}\mathbf{z}^\top \Phi \mathbf{z} - \mathbf{z}^\top \boldsymbol{\mu} \\
&\geq \sup_{A\mathbf{z} \geq \tilde{\mathbf{b}}} -\frac{\kappa}{2}\mathbf{z}^\top \Psi \mathbf{z} - \mathbf{z}^\top \boldsymbol{\mu} = \mathbf{P}_Z = \mathbf{D}_Z \\
&= \inf_{\mathbf{y} \geq \mathbf{0}} \frac{1}{\kappa}\frac{1}{2}\mathbf{y}^\top A \Psi^{-1} A^\top \mathbf{y} + \frac{1}{\kappa}\mathbf{y}^\top A \Psi^{-1} \boldsymbol{\mu} + \mathbf{y}^\top \tilde{\mathbf{b}} + \frac{1}{\kappa}\frac{1}{2}\boldsymbol{\mu}^\top \Psi^{-1} \boldsymbol{\mu} \\
&= \inf_{\mathbf{y} \geq \mathbf{0}} \frac{1}{\kappa}\frac{1}{2}\mathbf{y}^\top A \Psi^{-1} A^\top \mathbf{y} + \frac{1}{\kappa}\mathbf{y}^\top A \Psi^{-1} \boldsymbol{\mu} + \frac{1}{\kappa}\mathbf{y}^\top \tilde{\mathbf{b}} + \frac{\kappa - 1}{\kappa}\mathbf{y}^\top \tilde{\mathbf{b}} + \frac{1}{\kappa}\frac{1}{2}\boldsymbol{\mu}^\top \Psi^{-1} \boldsymbol{\mu} \\
&\geq \inf_{\mathbf{y} \geq \mathbf{0}} \frac{1}{\kappa}\frac{1}{2}\mathbf{y}^\top A \Psi^{-1} A^\top \mathbf{y} + \frac{1}{\kappa}\mathbf{y}^\top A \Psi^{-1} \boldsymbol{\mu} + \frac{1}{\kappa}\mathbf{y}^\top \tilde{\mathbf{b}} + \frac{1}{\kappa}\frac{1}{2}\boldsymbol{\mu}^\top \Psi^{-1} \boldsymbol{\mu} = \frac{1}{\kappa}\mathbf{D}_U = \frac{1}{\kappa}\mathbf{P}_U.
\end{aligned}$$

In the last line, we have dropped the term $\frac{\kappa-1}{\kappa}\mathbf{y}^\top \tilde{\mathbf{b}}$ since it is positive (recall that $\mathbf{y} \geq \mathbf{0}$ and $\tilde{\mathbf{b}} \geq \mathbf{0}$). Thus, $\mathbf{P}_L \geq \frac{1}{\kappa}\mathbf{P}_U$ which yields the desired inequality

$$\sup_{\boldsymbol{\lambda} \in \Lambda} L_i(\boldsymbol{\lambda}) - L_i(\boldsymbol{\lambda}_i) \geq \frac{1}{\max(\alpha+1, 2)} \sup_{\boldsymbol{\lambda} \in \Lambda} U_i(\boldsymbol{\lambda}) - U_i(\boldsymbol{\lambda}_i).$$

■

Appendix C. Relation to Other Sparse Regression Methods

This section considers MED regression with a squared error loss. This will show a connection between the MED regression framework and standard regression methods such as least squares or Ridge regression, ℓ_1 regularized regression methods such as the Lasso (Tibshirani, 1996) and intermediates such as the Elastic Net (Zou and Hastie, 2005). In particular, the regularizer introduced by feature selection and kernel selection in the MED framework will be shown to resemble the Elastic Net and the Lasso and Ridge regression for appropriate choices of α. In this article, we define the ℓ_1 norm of a vector $\mathbf{w} \in \mathbb{R}^d$ as $\|\mathbf{w}\|_1 = \sum_{d=1}^D |\mathbf{w}(d)|$, and the ℓ_2 norm as $\|\mathbf{w}\|_2^2 = \sum_{d=1}^D |\mathbf{w}(d)|^2$.

Consider the ℓ_2-regularized least squares problem with input-output pairs $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$ where $\mathbf{x}_t \in \mathbb{R}^D$ and $y_t \in \mathbb{R}$. The squared error in predicting y_t from $\mathbf{x}_t$ is minimized while also minimizing the ℓ_2 norm of the classifier. Equivalently, this can be posed as the minimization of the ℓ_2 norm of the classifier subject to a hard constraint on the total squared error obtained on the training data. We wish to estimate a regression function of the form $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$ whose parameters $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ are given by the following constrained minimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \text{ s.t. } \sum_{t=1}^T \|\mathbf{w}^\top \mathbf{x}_t + b - y_t\|^2 \leq \Upsilon$$

for some $\Upsilon \in \mathbb{R}^+$. The dual problem for the above can be obtained by noting that the solution must be of the form $\mathbf{w}^* = \sum_{t=1}^T \lambda_t \mathbf{x}_t$ by standard reproducing kernel Hilbert space arguments (Schölkopf and Smola, 2001). We can rewrite the optimization problem as follows:

$$\min_{\boldsymbol{\lambda}} \frac{1}{2}\sum_{t,\tau} \lambda_t \lambda_\tau \mathbf{x}_t^\top \mathbf{x}_\tau \text{ s.t. } \sum_{t=1}^T \left\| \sum_{\tau=1}^T \lambda_\tau \mathbf{x}_t^\top \mathbf{x}_\tau + \frac{1}{T}\sum_{u=1}^T y_u - \frac{1}{T}\sum_{u=1}^T \sum_{\tau=1}^T \lambda_\tau \mathbf{x}_u^\top \mathbf{x}_\tau - y_t \right\|^2 \leq \Upsilon \qquad (3)$$

after minimization over b has been performed. Recall that the prediction function can also be written in terms of $\lambda_1, \ldots, \lambda_T$ as a function of a query datum $\mathbf{x}$ as follows:

$$\hat{y} = \sum_{t=1}^T \lambda_t \mathbf{x}_t^\top \mathbf{x} + \frac{1}{T}\sum_{t=1}^T y_t - \frac{1}{T}\sum_{t=1}^T \sum_{\tau=1}^T \lambda_\tau \mathbf{x}_t^\top \mathbf{x}_\tau.$$

It is possible to now consider the same manipulation that MED with feature selection produces by integrating over switches with a Bernoulli prior. This yields the following feature selection convex program that is a simple variant of least squares regression:

$$\begin{cases} \min_{\boldsymbol{\lambda}} \sum_{d=1}^D \log(\alpha + \exp(\frac{1}{2}\sum_t \sum_\tau \lambda_t \lambda_\tau \mathbf{x}_t(d) \mathbf{x}_\tau(d))) - D\log(\alpha+1) \\ \text{s.t. } \sum_{t=1}^T \left\| \sum_{\tau=1}^T \lambda_\tau \mathbf{x}_t^\top \mathbf{x}_\tau + \frac{1}{T}\sum_{u=1}^T y_t - \frac{1}{T}\sum_{u=1}^T \sum_{\tau=1}^T \lambda_\tau \mathbf{x}_u^\top \mathbf{x}_\tau - y_t \right\|^2 \leq \Upsilon. \end{cases}$$

Similarly, the prediction rule is as follows in the feature selection variant:

$$\hat{y} = \sum_{t=1}^T \lambda_t \sum_{d=1}^D \hat{\mathbf{s}}(d) \mathbf{x}_t(d) \mathbf{x}(d) + \frac{1}{T}\sum_{t=1}^T y_t - \frac{1}{T}\sum_{t=1}^T \sum_{\tau=1}^T \lambda_\tau \sum_{d=1}^D \hat{\mathbf{s}}(d) \mathbf{x}_t(d) \mathbf{x}_\tau(d) \qquad (4)$$

where $\hat{\mathbf{s}}(d)$ is given by:

$$\hat{\mathbf{s}}(d) = \frac{1}{1 + \alpha \exp(-\frac{1}{2}\sum_{t=1}^T \sum_{\tau=1}^T \lambda_t \lambda_\tau \mathbf{x}_t(d) \mathbf{x}_\tau(d))}.$$

If we let $\alpha = 0$, it is straightforward to see that we recover the standard least squares setup in Equation 3. However, this dual problem in the MED formulation is clearly inducing a different regularization on the classifier. We next investigate what primal regularizer corresponds to this change and write it in terms of the original classification parameters $\mathbf{w}$. This will show a connection to the ℓ_1 regularization popularized by the Lasso method.

First, note that the prediction $\hat{y}$ in Equation 4 can be written in terms of a primal parameter $\mathbf{w}$ as $\hat{y} = \mathbf{w}^\top \mathbf{x} + b$ if we define the parameter element-wise as:

$$\mathbf{w}(d) = \frac{\sum_t \lambda_t \mathbf{x}_t(d)}{1 + \alpha \exp(-\frac{1}{2}\sum_{t=1}^T \sum_{\tau=1}^T \lambda_t \lambda_\tau \mathbf{x}_t(d) \mathbf{x}_\tau(d))}.$$

Instead of an ℓ_2 norm, the MED program corresponds to minimizing the following regularizer:

$$\ell_{MED} = \sum_{d=1}^D \log\left(\frac{\alpha}{\alpha+1} + \frac{1}{\alpha+1}\exp(\frac{1}{2}\sum_t \sum_\tau \lambda_t \lambda_\tau \mathbf{x}_t(d) \mathbf{x}_\tau(d))\right).$$

The above can be written in terms of $\mathbf{w}$ as follows:

$$\ell_{MED}(\mathbf{w}) = \sum_{d=1}^D h(\mathbf{w}_d)$$

where the function $h()$ is defined implicitly by the following equation

$$\mathbf{w}(d)^2 = \frac{2\log(\alpha) + 2\log(\exp(h) - 1)}{(1 - 1/(\exp(h) - 1))^2}.$$

Near the origin, this function behaves like an ℓ_1 norm and, further away, behaves like an ℓ_2 norm. In Figure 6(a) we plot the function $h(\mathbf{w}(d))$ for various values of $\mathbf{w}(d)$ scaled appropriately so that $h(1) = 1$. For small α, the induced penalty on the regression parameters resembles an ℓ_2 norm. As α increases, a behavior resembling an ℓ_1 norm emerges. In intermediate settings, the MED regularizer interpolates between these two behaviors in a manner reminiscent of the so-called Elastic Net (Zou and Hastie, 2005) which uses a conic combination of ℓ_1 and ℓ_2 regularization. In addition, two-dimensional contour plots are shown comparing ℓ_{MED} to ℓ_1 and ℓ_2 regularization in Figure 6(b). While ℓ_{MED} is not identical to the Elastic Net regularization, the similarity warrants further exploration and may be useful in group Lasso and multitask settings (Turlach et al., 2005).

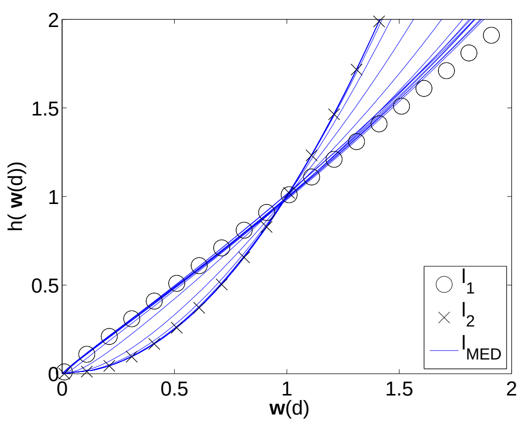
References

R.K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

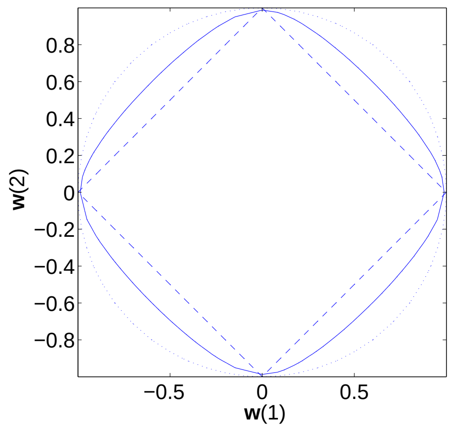
A. Argyriou, T. Evgeniou, and M. Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, December 2008.

F. Bach, G.R.G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality and the SMO algorithm. In *Proceedings of the International Conference on Machine Learning*, 2004.

F.R. Bach. Consistency of the group Lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9:1179–1225, 2008.



(a) One dimensional plot



(b) Two dimensional contour plot

Figure 6: Various penalties on the classifier weights. In (a), the penalties are shown as one dimensional functions of the form $h(\mathbf{w}(d))$ as α varies from $\alpha = 0$ (which mimics an ℓ_2 norm) to large α large (which resembles an ℓ_1 norm). In (b), a two-dimensional contour plot is provided showing the shape of the ℓ_1 penalty as a dashed line, the shape of the ℓ_2 penalty as a dotted line and the shape of the ℓ_{MED} penalty with $\alpha = 2$ as a solid line.

J. Baxter. Learning internal representations. In *Proceedings of the International ACM Workshop on Computational Learning Theory*, 1995.

J. Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12: 149–198, 2000.

S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. In *Proceedings of the Conference on Learning Theory and Kernel Machines*, 2003.

L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, 2008.

S. Boyd and L. Vanderberghe. *Convex Optimization*. Cambridge University Press, 2004.

R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

N. Cristianini, J. Kandola, A. Elisseeff, and J. Shawe-Taylor. On kernel target alignment. In *Advances in Neural Information Processing Systems*, volume 14, 2001.

O. Dekel, Y. Singer, and P. Long. Online multitask learning. In *Proceedings of the Conference on Learning Theory*, 2006.

T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

M. Dudik, M.E. Blei, and R.E. Schapire. Hierarchical maximum entropy density estimation. In *Proceedings of the International Conference on Machine learning*, 2007.

T. Evgeniou, C.A. Micchelli, and M. Pontil. Learning multiple tasks with kernels. *Journal of Machine Learning Research*, 6:615–637, 2005.

T. Heskes. Solving a huge number of similar tasks: A combination of multi-task learning and a hierarchical Bayesian approach. In *Proceedings of the International Conference on Machine Learning*, 1998.

T. Heskes. Empirical Bayes for learning to learn. In *Proceedings of the International Conference on Machine Learning*, 2004.

T. Jaakkola and M. Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10:25–37, 2000.

T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *Advances in Neural Information Processing Systems*, 1999.

T. Jebara. *Machine Learning: Discriminative and Generative*. Kluwer Academic, 2003.

T. Jebara. Multi-task feature and kernel selection for SVMs. In *Proceedings of the International Conference on Machine Learning*, 2004.

T. Jebara and T. Jaakkola. Feature selection and dualities in maximum entropy discrimination. In *Uncertainty in Artifical Intelligence*, 2000.

T. Joachims. Training linear SVMs in linear time. In *Proceedings of the ACM International Conference On Knowledge Discovery and Data Mining*, pages 217–226, 2006.

K. Koh, S.J. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine Learning Research*, 8:1519–1555, July 2007.

G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. Jordan. Learning the kernel matrix with semi-definite programming. In *Proceedings of the International Conference on Machine Learning*, 2002.

J. Langford and J. Shawe-Taylor. PAC-Bayes and margins. In *Advances in Neural Information Processing Systems*, pages 423–430, 2002.

P. Long and X. Wu. Mistake bounds for maximum entropy discrimination. In *Advances in Neural Information Processing Systems*, 2004.

A. Maurer. Bounds for linear multi-task learning. *Journal of Machine Learning Research*, 7:117–139, December 2006.

A. Maurer. Transfer bounds for linear feature learning. *Machine Learning*, 75(3):327–350, June 2009.

D. McAllester. PAC-Bayesian model averaging. In *Proceedings of the Workshop on Computational Learning Theory*, 1999.

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

B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2001.

B. Schölkopf, A. J. Smola, and K.R. Muller. *Advances in Kernel Methods - Support Vector Learning*, chapter Kernel principal component analysis, pages 327–352. MIT Press, 1999.

S. Shalev-Shwartz and N. Srebro. SVM optimization: Inverse dependence on training set size. In *Proceedings of the International Conference on Machine Learning*, 2008.

S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *Proceedings of the International Conference on Machine Learning*, 2007.

B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, 2004.

S. Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems*, 1995.

S. Thrun and L.Y. Pratt. *Learning to Learn*. Kluwer Academic, 1997.

R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.

J.A. Tropp. Just relax: Convex programming methods for identifying sparse signals. *IEEE Transactions on Information Theory*, 51(3):1030–1051, 2006.

B.A. Turlach, W.N. Venables, and S.J. Wright. Simultaneous variable selection. *Technometrics*, pages 349–363, 2005.

M. Wainwright, P. Ravikumar, and J. Lafferty. High-dimensional graphical model selection using l1-regularized logistic regression. In *Advances in Neural Information Processing Systems*, 2007.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In *Advances in Neural Information Processing Systems*, 2000.

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

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society B*, 68:49–67, 2006.

J. Zhu, E.P. Xing, and B. Zhang. Laplace maximum margin Markov networks. In *Proceedings of the International Conference on Machine Learning*, 2008.

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, 67(2):301–320, 2005.