BMC Bioinformatics

Research article

Enhanced protein fold recognition through a novel data integration approach

Yiming Ying*[1], Kaizhu Huang[2] and Colin Campbell[1]

Address: [1]Department of Engineering Mathematics, University of Bristol, Bristol, BS8 1TR, UK and [2]National Laboratory of Pattern Recognition, Institute of Automation, The Chinese Academy of Sciences, 100190 Beijing, PR China

Email: Yiming Ying* - mathying@gmail.com; Kaizhu Huang - kzhuang@nlpr.ia.ac.cn; Colin Campbell - C.Campbell@bris.ac.uk

* Corresponding author

Published: 26 August 2009

BMC Bioinformatics 2009, **10**:267 doi:10.1186/1471-2105-10-267

Received: 16 April 2009
Accepted: 26 August 2009

This article is available from: http://www.biomedcentral.com/1471-2105/10/267

Abstract

Background: Protein fold recognition is a key step in protein three-dimensional (3D) structure discovery. There are multiple fold discriminatory data sources which use physicochemical and structural properties as well as further data sources derived from local sequence alignments. This raises the issue of finding the most efficient method for combining these different informative data sources and exploring their relative significance for protein fold classification. Kernel methods have been extensively used for biological data analysis. They can incorporate separate fold discriminatory features into kernel matrices which encode the similarity between samples in their respective data sources.

Results: In this paper we consider the problem of integrating multiple data sources using a kernel-based approach. We propose a novel information-theoretic approach based on a Kullback-Leibler (KL) divergence between the output kernel matrix and the input kernel matrix so as to integrate heterogeneous data sources. One of the most appealing properties of this approach is that it can easily cope with multi-class classification and multi-task learning by an appropriate choice of the output kernel matrix. Based on the position of the output and input kernel matrices in the KL-divergence objective, there are two formulations which we respectively refer to as *MKLdiv-dc* and *MKLdiv-conv*. We propose to efficiently solve MKLdiv-dc by a difference of convex (DC) programming method and MKLdiv-conv by a projected gradient descent algorithm. The effectiveness of the proposed approaches is evaluated on a benchmark dataset for protein fold recognition and a yeast protein function prediction problem.

Conclusion: Our proposed methods MKLdiv-dc and MKLdiv-conv are able to achieve state-of-the-art performance on the SCOP PDB-40D benchmark dataset for protein fold prediction and provide useful insights into the relative significance of informative data sources. In particular, MKLdiv-dc further improves the fold discrimination accuracy to 75.19% which is a more than 5% improvement over competitive Bayesian probabilistic and SVM margin-based kernel learning methods. Furthermore, we report a competitive performance on the yeast protein function prediction problem.

Background

A huge number of protein coding sequences have been generated by genome sequencing projects. In contrast, there is a much slower increase in the number of known three-dimensional (3D) protein structures. Determination of a protein's 3D structure is a formidable challenge if there is no sequence similarity to proteins of known structure and thus protein structure prediction remains a core problem within computational biology.

Computational prediction of protein structure has achieved significant successes [1,2]. Focusing on the fold prediction problem of immediate interest to this paper, one computational method known as the *taxonomic* approach [3,4], presumes the number of folds is restricted and focuses on structural predictions in the context of a particular classification of 3D folds. Proteins are in a common fold if they share the same major secondary structures in the same arrangement and the same topological connections [5,6]. In the taxonomic method for protein fold classification, there are several fold discriminatory *data sources* or groups of attributes available such as amino acid composition, predicted secondary structure, and selected structural and physicochemical properties of the constituent amino acids. Previous methods for integrating these heterogeneous data sources include simply merging them together or combining trained classifiers over individual data sources [3,4,7,8]. However, how to integrate fold discriminatory data sources systematically and efficiently, without resorting to *ad hoc* ensemble learning, still remains a challenging problem.

Kernel methods [9,10] have been successfully used for data fusion in biological applications. *Kernel matrices* encode the similarity between data objects within a given input space and these data objects can include graphs and sequence strings in addition to real-valued or integer data. Thus the problem of data integration is transformed into the problem of learning the most appropriate combination of candidate kernel matrices, representing these heterogeneous data sources. The typical framework is to learn a linear combination of candidate kernels. This is often termed *multiple kernel learning* (MKL) in Machine Learning, and *non-parametric group lasso* in Statistics. Recent trends in kernel learning are usually based on the margin maximization criterion used by Support Vector Machines (SVMs) or variants [11]. The popularity of SVM margin-based kernel learning stems from its efficient optimization formulations [11-14] and sound theoretical foundation [11,15,16]. Other data integration methods include the COSSO estimate for additive models [17], kernel discriminant analysis [18], multi-label multiple kernel learning [19,20] and Bayesian probabilistic models [21,22]. These methods, in general, can combine multiple data sources to enhance biological inference [21,23] and provide insights into the significance of the different data sources used.

Following a different approach, in this paper we propose an alternative criterion for kernel matrix learning and data integration, which we will call *MKLdiv*. Specifically, we propose an information-theoretic approach to learn a linear combination of kernel matrices, encoding information from different data sources, through the use of a Kullback-Leibler divergence [24-28] between two zero-mean Gaussian distributions defined by the input matrix and output matrix. The potential advantage of this approach is that, by choosing different output matrices, the method can be easily extended to different learning tasks, such as multi-class classification and multi-task learning. These are common tasks in biological data analysis.

To illustrate the method, we will focus on learning a linear combination of candidate kernel matrices (heterogeneous data sources) using the KL-divergence criterion with a main application to the protein fold prediction problem. There are two different formulations based on the relative position of the input kernel matrix and the output kernel matrix in the KL-divergence objective. For the first formulation, although this approach involves a matrix determinant term which is not convex in general, we elegantly reformulate the learning task as a difference of convex problem, which can be efficiently solved by a sequence of convex optimizations. Hence we refer to it as *MKLdiv-dc*. The second KL-divergence formulation for kernel integration, called *MKLdiv-conv*, is convex and can be solved by a projected gradient descent algorithm. Experimental results show that these formulations lead to state-of-the-art prediction performance. In particular, MKLdiv-dc outperforms the best reported performance on the important task of protein fold recognition, for the benchmark dataset used.

Methods

In the following we first revisit kernel learning approaches based on SVMs [11] and kernel discriminant analysis [18]. Then, we introduce our novel information-theoretic approach for data integration based on a KL-divergence criterion. Finally we discuss how to solve the optimization task efficiently. For brevity, we use the conventional notation $\mathbb{N}_n = \{1, 2, ..., n\}$ for any $n \in \mathbb{N}$.

Background and Related Work

Kernel methods are extensively used for biological data analysis. A symmetric function $K : X \times X \to \mathbb{R}$ is called a *kernel function* if it is positive semi-definite, by which we mean that, for any $n \in \mathbb{N}$ and $\{x_i \in X : i \in \mathbb{N}_n\}$, the *Gram matrix*

$(K(x_i, x_j))_{i,j\in\mathbb{N}_n}$ is positive semi-definite. According to [29], its corresponding reproducing kernel Hilbert space (RKHS), usually denoted by $\mathcal{H}_K$, can be defined to be the completion of the linear span of the set of functions $\{K_x(\cdot) := K(x, \cdot): x \in X\}$ with inner product satisfying, for any $x \in X$ and $g \in \mathcal{H}_K$, the *reproducing property* $\langle K_x, g\rangle_K = g(x)$. By Mercer's theorem, there exists a high dimensional (possible infinite) Hilbert feature space with inner product $\langle\cdot,\cdot\rangle$ and a feature map ϕ: $X \to \mathcal{F}$ such that $K(x, t) = \langle\phi(x), \phi(t)\rangle_{\mathcal{F}}$, $\forall\, x, t \in X$. Intuitively, the kernel function K implicitly maps the data space X into a high dimensional space $\mathcal{F}$, see [9,10] for more details.

Within the context of protein fold recognition, we have m different *fold discriminatory data sources* where *samples* across each data source can be represented by $\mathbf{x}^\ell = \{x_i^\ell : i \in \mathbb{N}_n\}$ for $\ell \in \mathbb{N}_m$ and the outputs are denoted by $\mathbf{y} = \{y_i : i \in \mathbb{N}_n\}$. For kernel methods, for any $\ell \in \mathbb{N}_m$, each ℓ-th data source can be encoded into a candidate kernel matrix denoted by $\mathbf{K}_\ell = (K_\ell(x_i^\ell, x_j^\ell))_{ij}$. Depending on the different data sources used, the candidate kernel function K_ℓ should be specified *a priori* by the practitioner. The composite kernel matrix is given by $\mathbf{K}_\lambda = \sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell$ where $\{\lambda_\ell: \ell \in \mathbb{N}_m\}$ are real-valued kernel weights and typically they are restricted to be non-negative. In this context, the problem of data integration is consequently reduced to the problem of learning a convex combination of candidate kernel matrices: more precisely learning the kernel weights λ. Different optimization criteria over the candidate kernels arise from the particular kernel learning algorithm used. Cristianini et al. [30] proposed a kernel learning approach which uses the cosine of the angle between the two bi-dimensional vectors $\mathbf{K}_\lambda$ and $\mathbf{K}_\mathbf{y}$ representing the Gram matrices. This is achieved by maximizing the *kernel alignment*:

$$\frac{\left\langle \mathbf{K}_\lambda, \mathbf{K}_\gamma \right\rangle}{\sqrt{\left\langle \mathbf{K}_\lambda, \mathbf{K}_\lambda \right\rangle \left\langle \mathbf{K}_\gamma, \mathbf{K}_\gamma \right\rangle}}.$$

The above kernel learning formulation can be solved by a semi-definite programming (SDP) approach (see Section 4.7 of [11]). However, an SDP formulation is computationally intensive.

Another widely used criterion for kernel learning is based on the margin concept in SVMs and variants. Denoting the simplex set as $\Delta = \{\lambda=(\lambda_1,\lambda_2,...,\lambda_m): \sum_{\ell\in\mathbb{N}_m} \lambda_\ell = 1, \lambda_\ell \geq 0\}$, Lanckriet et al [11] proposed the following formulation for kernel learning:

$$\min_{\lambda\in\Delta} \Omega(\mathbf{K}_\lambda) = \min_{\lambda\in\Delta} \max\{\alpha^\top \mathbf{1}_n - \frac{1}{2}\alpha^\top \mathrm{diag}(\mathbf{t})\mathbf{K}_\lambda \mathrm{diag}(\mathbf{t})\alpha : 0 \leq \alpha \leq C, \text{ and } \alpha^\top t = 0\}, \quad (1)$$

where $\mathbf{1}_n$ is a column vector of all ones, C is a trade-off parameter, and $\mathbf{t} = (t_1, t_2, ..., t_n)$ denotes the binary outputs with $t_i \in \{1, -1\}$ being the class label for i-th instance. This task was reformulated as a quadratically constrained quadratic programming (QCQP) problem and later improved by Sonnenburg et al. [14] who reformulated it as a semi-infinite linear programming (SILP) task. Moreover, it was pointed out in [12,13,17,31] that this is equivalent to the following sparse L^1-regularization formulation:

$$\min_{f_\ell\in\mathcal{H}_{K_\ell},\ell\in\mathbb{N}_m} C\sum_{i\in\mathbb{N}_n}(1 - t_i\sum_{\ell\in\mathbb{N}_n} f_\ell(x_i^\ell))_+ + \frac{1}{2}\left(\sum_{\ell\in\mathbb{N}_n} \|f_\ell\|_{K_\ell}\right)^2. \quad (2)$$

The L^1-regularization term $\sum_{\ell\in\mathbb{N}_m} \|f_\ell\|_{K_\ell}$ encourages the sparsity [32] of RKHS-norm terms, and thus indicates the relative importance of data sources. It was shown in [13] that the standard L^2-regularization $\sum_{\ell\in\mathbb{N}_m} ||f_\ell||^2_{K_\ell}$ is equivalent to the use of uniformly weighted kernel weights λ, i.e. $\lambda_\ell = \frac{1}{m}$ for any $\ell \in \mathbb{N}_m$. Recently, Ye et al. [18] proposed an appealing kernel learning approach based on regularized kernel discriminant analysis. This can similarly be shown to be equivalent to a sparse L^1-regularization formulation with a least square loss, see Appendix 1 for more details.

Information-theoretic Data Integration

In this paper we adopt a novel information-theoretic approach to learn the kernel combinatorial weights. The main idea is to quantify the similarity between $\mathbf{K}_\lambda$ and $\mathbf{K}_\mathbf{y}$ through a Kullback-Leibler (KL) divergence or relative entropy term [24-28]. This approach is based on noting that these kernel matrices encode the similarity of data objects within their respective input and label data spaces. Furthermore, there is a simple bijection between the set of distance measures in these data spaces and the set of zero-mean multivariate Gaussian distributions [25]. Using this

bijection, the difference between two distance measures, parameterized by $\mathbf{K}_\lambda$ and $\mathbf{K}_y$, can be quantified by the relative entropy or Kullback-Leibler (KL) divergence between the corresponding multivariate Gaussians. Matching kernel matrices $\mathbf{K}_\lambda$ and $\mathbf{K}_y$ can therefore be realized by minimizing a KL divergence between these distributions and we will exploit this approach below in the context of multiple kernel learning.

Kernel matrices are generally positive semi-definite and thus can be regarded as the covariance matrices of Gaussian distributions. As described in [24], the Kullback-Leibler (KL) divergence (relative entropy) between a Gaussian distribution $\mathcal{N}(0, \mathbf{K}_y)$ with the output covariance matrix $\mathbf{K}_y$ and a Gaussian distribution $\mathcal{N}(0, \mathbf{K}_x)$ with the input kernel covariance matrix $\mathbf{K}_x$ is:

$$\mathrm{KL}(\mathcal{N}(0,\mathbf{K}_y)\,||\mathcal{N}(0,\mathbf{K}_x)) := \frac{1}{2}\mathrm{Tr}(\mathbf{K}_y\mathbf{K}_x^{-1}) + \frac{1}{2}\log|\mathbf{K}_x| - \frac{1}{2}\log|\mathbf{K}_y| - \frac{n}{2}. \quad (3)$$

where, for any square matrix $\mathbf{B}$, the notation $\mathrm{Tr}(\mathbf{B})$ denotes its trace. The *a priori* choice of the output matrix $\mathbf{K}_y$ will be discussed later. Though KL ($\mathcal{N}(0, \mathbf{K}_y)||\mathcal{N}(0, \mathbf{K}_x)$) is non-convex w.r.t. $\mathbf{K}_x$, it has a unique minimum at $\mathbf{K}_x = \mathbf{K}_y$ if $\mathbf{K}_y$ is positive definite, suggesting that minimizing the above KL-divergence encourages $\mathbf{K}_x$ to approach $\mathbf{K}_y$. If the input kernel matrix $\mathbf{K}_x$ is represented by a linear combination of m candidate kernel matrices, i.e. $\mathbf{K}_x = \mathbf{K}_\lambda = \sum_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell$, the above KL-divergence based kernel learning is reduced to the following formulation:

$$\begin{aligned}&\arg\min_{\lambda\in\Delta}\mathrm{KL}(\mathcal{N}(0,\mathbf{K}_y)\,||\mathcal{N}(0,\mathbf{K}_\lambda))\\ &= \arg\min_{\lambda\in\Delta}\mathrm{Tr}(\mathbf{K}_y(\sum\nolimits_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell + \sigma\mathbf{I}_n)^{-1}) + \log\left|\sum\nolimits_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell + \sigma\mathbf{I}_n\right|,\end{aligned} \quad (4)$$

where $\mathbf{I}_n$ denotes the $n \times n$ identity matrix and $\sigma > 0$ is a supplemented small parameter to avoid the singularity of $\mathbf{K}_\lambda$.

Since the KL-divergence is not symmetric with respect to $\mathbf{K}_y$ and $\mathbf{K}_\lambda$, another alternative approach to matching kernel matrices is given by

$$\begin{aligned}&\arg\min_{\lambda\in\Delta}\mathrm{KL}(\mathcal{N}(0,\mathbf{K}_\lambda)\,||\mathcal{N}(0,\mathbf{K}_y))\\ &= \arg\min_{\lambda\in\Delta}\sum\nolimits_{\ell\in\mathbb{N}_m}\lambda_\ell\mathrm{Tr}((\mathbf{K}_y + \sigma\mathbf{I}_n)^{-1}\mathbf{K}_\ell) - \log\left|\sum\nolimits_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell + \sigma\mathbf{I}_n\right|,\end{aligned} \quad (5)$$

where parameter $\sigma > 0$ is added to avoid the singularity of $\mathbf{K}_y$. If there is no positive semi-definiteness restriction over $\mathbf{K}\ell$, this formulation is a well-known convex *maximum-determinant problem* [33] which is a more general formulation than semi-definite programming (SDP), its implementation is computationally intensive, and thus cannot be extended to large-scale problems according to [33]. However, formulation (5) has a special structure here: λ_ℓ is non-negative and all candidate kernel matrices are positive semi-definite. Hence, we can solve this problem by a simple projected gradient descent method, see below for more details.

The KL-divergence criterion for kernel integration was also successfully used in [27,28] which formulated the problem of supervised network inference as a kernel matrix completion problem. In terms of information geometry [34], formulation (4) corresponds to finding the *m*-projection of $\mathbf{K}_y$ over an *e*-flat submanifold. The convex problem (5) can be regarded as finding the *e*-projection of $\mathbf{K}_y$ over a *m*-flat submanifold. In [26], formulation (4) was developed for learning an optimal linear combination of diffusion kernels for biological networks. A gradient-based method was employed in [26] to learn a proper linear combination of diffusion kernels. This optimization method largely relies on the special property of all candidate diffusion kernel matrices enjoying the same eigenvectors and the gradient-based learning method could be a problem if we deal with general kernel matrices. In the next section, we propose to solve the general kernel learning formulation (4) using a difference of convex optimization method.

The formulation (4) also has a close relation with *Gaussian Process regression* [35]. A Gaussian process f can be fully specified by giving the covariance matrix for any finite set of zero-mean random variables $\mathbf{f} = \{f(x_i): i \in \mathbb{N}_m\}$. The relation between the inputs $\mathbf{x} = \{x_i : i \in \mathbb{N}_n\}$ and outputs $\mathbf{y} = \{y_i : i \in \mathbb{N}_m\}$ is realized by the latent variable $\mathbf{f}$ as follows:

$$\mathbf{y} \mid \mathbf{f}, \mathbf{x} \sim \mathcal{N}(\mathbf{y} \mid \mathbf{f}, \sigma\mathbf{I}_n)$$

where $\mathbf{I}_n$ denotes the $n \times n$ identity matrix and the latent random variable $\mathbf{f} = (f(x_1, \ldots, f(x_n)))$ is distributed as a *Gaussian process prior*. The Gaussian process prior can be fully specified by a kernel K with a random covariance matrix $\mathbf{K} = (K(x_i, x_j))_{i,j\in\mathbb{N}_n}$ associated with random samples $\mathbf{x} = \{x_i: i \in \mathbb{N}_n\}$. Specifically, it can be written as $\mathbf{f}|\mathbf{x} \sim \mathcal{N}(\mathbf{f}|\, 0, \mathbf{K}_\lambda)$. We assume a uniform distribution over λ, i.e.

a Dirichlet prior distribution $\lambda \sim \prod_{\ell=1}^{m} \lambda_\ell^{\alpha 0^{-1}}$ with $\alpha_0 = 1$. If we let $\mathbf{K}_y = \mathbf{y}\mathbf{y}^T$ in the objective function of formulation (4), then one can easily check that, up to a constant term, the objective function in formulation (4) is the negative of the log likelihood of Gaussian process regression, and maximizing the log likelihood is equivalent to the minimization problem (4).

Optimization Formulation

We now turn our attention to optimization approaches for the KL-divergence based kernel learning formulations (4) and (5). In particular, we show that formulation (5) can be approached by a projected gradient descent method and (4) can be solved by a difference of convex algorithm (DCA) [36] which, for linear constraint conditions, reduces to the special case of a concave convex procedure (CCCP) [37]. To this end, let

$$g(\lambda) := -\log\left|\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n\right| \tag{6}$$

and

$$f(\lambda) := \mathrm{Tr}(\mathbf{K}_y(\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}). \tag{7}$$

Theorem 1 *Let functions g and f be defined by (6) and (7). Then, both f and g are convex with respect to* $\lambda \in \Delta$. *Moreover, problem (5) is convex and problem (4) is a difference of convex problem, i.e.*

$$\min_{\lambda\in\Delta} \mathcal{L}(\lambda) := f(\lambda) - g(\lambda). \tag{8}$$

Proof It suffices to prove the convexity of *f* and *g*. To this end, from [38] we observe that functions – log |C| and $\mathrm{Tr}(\mathbf{K}_y\mathbf{C}^{-1})$ are convex with respect to positive semi-definite matrices **C**. Hence, *f* and *g* are convex with respect to $\lambda \in \Delta$. This completes the proof of the theorem.

For simplicity we refer to the KL-divergence kernel learning formulation (4) as *MKLdiv-dc* since it is a difference of convex problem and refer to formulation (5) as *MKLdiv-conv* since it is a convex problem.

Projected Gradient Descent Method for MKLdiv-conv

We propose a projected gradient descent (PGD) method to solve problem (5). The idea of this method is to alternately implement a gradient descent and then a projection to the feasible domain, see e.g. [39]. Recall the derivative of the log determinant,(see e.g. the matrix cookbook http://matrixcookbook.com/

$$\frac{\partial g(\lambda)}{\partial \lambda_j} = -\mathrm{Tr}\left((\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}\mathbf{K}_j\right). \tag{9}$$

With a little abuse of notation, we also denote by $\mathcal{L}$ the objective function of problem (5). Consequently, its gradient is given by

$$\frac{\partial \mathcal{L}(\lambda)}{\partial \lambda_j} = \mathrm{Tr}((\mathbf{K}_y+\sigma \mathbf{I}_n)^{-1}\mathbf{K}_j)\text{-}\mathrm{Tr}\left((\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}\mathbf{K}_j\right). \tag{10}$$

Then, at iteration step *t* the gradient descent step is realized by

$$\beta^{(t)} = \lambda^{(t)} - \eta \nabla \mathcal{L}(\lambda^{(t)}),$$

where $\eta > 0$ is a prescribed step size. The projection of β to the feasible domain Δ can be written as the following quadratic programming problem

$$\lambda^{(t+1)} = \arg\min_{\lambda\in\Delta} \left\| \beta^{(t)} - \lambda \right\|^2. \tag{11}$$

The theoretical convergence rate of the projected gradient descent method is generally of complexity $O\left(\frac{L}{\sqrt{t}}\right)$ where *t* is the iteration number and *L* is the Lipschitz constant of the gradient function defined by (10), see e.g. [39]. Here, the Lipschitz constant *L* is bounded by the largest eigenvalue of the Hessian $\mathcal{H}(\mathcal{L}) = ((\mathcal{H}(\mathcal{L}))_{ij\in\mathbb{N}_m})$ of the objective function defined, for any $i, j \in \mathbb{N}_m$, by

$$(\mathcal{H}(\mathcal{L}))_{ij} := \frac{\partial \mathcal{L}(\lambda)}{\partial \lambda_i \lambda_j} = \mathrm{Tr}((\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}\mathbf{K}_i(\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}\mathbf{K}_j).$$

Since $\mathcal{H}$ is convex, the Hessian $(\mathcal{L})$ is positive semi-definite and thus

$$\begin{aligned} L &\le \sup_{\lambda\in\Delta} \sum_{j\in\mathbb{N}_m} \mathrm{Tr}((\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}\mathbf{K}_\ell(\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}\mathbf{K}_j) \\ &= \sup_{\lambda\in\Delta} \sum_{j\in\mathbb{N}_m} ||(\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}\mathbf{K}_j||_{\mathrm{Fro}}^2 \\ &\le \sup_{\lambda\in\Delta} \sum_{j\in\mathbb{N}_m} ||(\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)^{-1}||_{\mathrm{Fro}}^2 ||\mathbf{K}_j||_{\mathrm{Fro}}^2 \le n \sum_{j\in\mathbb{N}_m} ||\mathbf{K}_j||_{\mathrm{Fro}}^2 / \sigma^2, \end{aligned} \tag{12}$$

where $||\cdot||_{\mathrm{Fro}}$ denotes the Frobenious norm of a matrix. Hence, the projected gradient descent algorithm could take longer time to become convergent if the value of σ is very small.

Difference of Convex Algorithm for MKLdiv-dc

By Theorem 1, problem (4) is a difference of convex problem. We propose to solve this problem by a concave convex procedure (CCCP) [36,37]. This procedure iteratively solves the following convex problem:

$$\lambda^{(t+1)} = \arg\min_{\lambda\in\Delta} f(\lambda) - g(\lambda^{(t)}) - \nabla g(\lambda^{(t)})(\lambda - \lambda^{(t)}), \tag{13}$$

where, for any $j \in \mathbb{N}_m$, the derivative of the log determinant is given by equation (9). Before we continue the main discussion, let us first note an interesting property of CCCP. By the definition of $\lambda^{(t+1)}$, we know that

$$\begin{aligned}\mathcal{L}(\lambda^{(t)}) = f(\lambda^{(t)}) - g(\lambda^{(t)}) &= f(\lambda^{(t)}) - g(\lambda^{(t)}) - \nabla g(\lambda^{(t)})(\lambda^{(t)} - \lambda^{(t)}) \\ &\geq \min_{\lambda\in\Delta} f(\lambda) - g(\lambda^{(t)}) - \nabla g(\lambda^{(t)})(\lambda - \lambda^{(t)}) \\ &= f(\lambda^{(t+1)}) - g(\lambda^{(t)}) - \nabla g(\lambda^{(t)})(\lambda^{(t+1)} - \lambda^{(t)}).\end{aligned}$$

Since g is convex, we have that

$$-g(\lambda^{(t)}) - \nabla g(\lambda^{(t)})(\lambda^{(t+1)} - \lambda^{(t)}) \geq -g(\lambda^{(t+1)}).$$

Consequently,

$$\mathcal{L}(\lambda^{(t)}) = f(\lambda^{(t)}) - g(\lambda^{(t)}) \geq f(\lambda^{(t+1)}) - g(\lambda^{(t+1)}) = \mathcal{L}(\lambda^{(t+1)}), \tag{14}$$

which means that the objective value ($\lambda^{(t)}$) monotonically decreases with each iteration. Consequently, we can use the relative change of the objective function as a stopping criterion. Local convergence of the DCA algorithm is proven in [36] (Lemma 3.6, Theorem 3.7). Tao and An [36] state that the DCA often converges to the global solution. Overall, the DC programming approach to MKLdiv-dc can be summarized as follows.

- Given a stopping threshold ε
- Initialize $\lambda^{(1)}$, e.g. $\lambda_\ell^{(1)} = \frac{1}{m}$ for any $\ell \in \mathbb{N}_m$
- Given the solution $\lambda^{(t)}$ at step t, for step $t + 1$, first compute á$g(\lambda^{(t)})$ by equation (9). Then, compute solution $\lambda^{(t+1)}$ of convex subproblem (13).
- Stop until the relative change $\frac{\mathcal{L}(\lambda^{(t)}) - \mathcal{L}(\lambda^{(t+1)})}{\mathcal{L}(\lambda^{(t+1)})} \leq \varepsilon$

  where ε is a stopping threshold

SILP Formulation for the Convex Subproblem (13)

We now turn to the solution of the convex subproblem (13). To see this, first decompose the output matrix $\mathbf{K}_y$ into the form $\mathbf{K}_y = AA^\top$, e.g. by eigen-decomposition. Here, A is an $n \times r$ matrix with $r = rank(A)$ which always exists since $\mathbf{K}_y$ is positive semi-definite. Hence, by introducing an auxiliary matrix $\alpha \in \mathbb{R}^{n\times r}$, we observe, for any positive definite matrix $\mathbf{C}$, that

$$\max_\alpha 2\mathrm{Tr}(\alpha^\top A) - \mathrm{Tr}(\alpha^\top \mathbf{C}\alpha) = \mathrm{Tr}(A^\top \mathbf{C}^{-1} A) = \mathrm{Tr}(AA^\top \mathbf{C}^{-1}).$$

Applying the above equality with $\mathbf{C} = \sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n$, up to a constant, equation (13) is equivalent to the augmented problem:

$$\min_{\lambda\in\Delta} \max_\alpha 2\mathrm{Tr}(\alpha^\top A) - \mathrm{Tr}(\alpha^\top (\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)\alpha) - \nabla g(\lambda^{(t)})\lambda.$$

Equivalently, by the min-max theorem (see e.g. [38])

$$\max_{\lambda\in\Delta} \min_\alpha - 2\mathrm{Tr}(\alpha^\top A) + \mathrm{Tr}(\alpha^\top (\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)\alpha) + \nabla g(\lambda^{(t)})\lambda. \tag{15}$$

To solve the subproblem (15), we can formulate it as a quadratically constrained quadratic programming (QCQP) problem as in [11]. Here we formulate the problem in (15) as a semi-infinite linear programming (SILP) problem [14,40] since SILP usually has better scalability compared to QCQP. To see this, let

$S_\ell(\alpha) = \mathrm{Tr}(\alpha\alpha^\top K_\ell) + \frac{\partial g(\lambda^{(t)})}{\partial \lambda_\ell}$, and

$S_0(\alpha) = -2\mathrm{Tr}(\alpha^\top A) + \sigma \mathrm{Tr}(\alpha^\top \alpha)$. Then, letting $\gamma = \min_\alpha - 2\mathrm{Tr}(\alpha^\top A) + \mathrm{Tr}(\alpha^\top (\sum_{\ell\in\mathbb{N}_m} \lambda_\ell \mathbf{K}_\ell + \sigma \mathbf{I}_n)\alpha) + \nabla g(\lambda^{(t)})\lambda$, we can rewrite (15) as a SILP problem:

$$\begin{aligned} \max_{\gamma,\lambda} \quad & \gamma \\ \text{s.t.} \quad & \sum_{\ell=1}^m \lambda_\ell = 1, \lambda_\ell \geq 0 \\ & \gamma - \sum_{\ell=1}^m \lambda_\ell S_\ell(\alpha) \leq S_0(\alpha), \ \forall \alpha \end{aligned} \tag{16}$$

In (16), there are an infinite number of constraints (indexed by a), indicative of a *semi-infinite linear programming* (SILP) problem. The SILP task can be solved by an iterative *column generation* algorithm (or exchange method) which is guaranteed to converge to a global optimum. A brief description of the column generation method is illustrated in Appedix 2.

Alternatively we could apply the projected gradient descent (PGD) method in the above subsection directly to the convex subproblem (13). However, the gradient function of its objective function involves the matrix

$(\sum_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell+\sigma\mathbf{I}_n)^{-1}$. In analogy to the argument of inequality (12), the Lipschitz constant of the gradient of the objective function in (13) is very large when the value of σ is very small, and thus the projected gradient descent algorithm could take longer to become convergent. Hence, this could make the overall DC programming unacceptable slow. In contrast, in the SILP formulation (16) we introduce the auxiliary variables α to avoid the matrix $(\sum_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell+\sigma\mathbf{I}_n)^{-1}$. In addition, the gradient descent algorithm generally needs to determine the step size η according to the value of σ, see also discussion in the experimental section.

Prior Choice of the Output Kernel Matrix

The choice of the output kernel matrix $\mathbf{K}_y$ will depend on the problem considered. We first consider a multi-class classification for the specific task of protein fold recognition. In this case, we preprocess the output labels using a one-against-all strategy. In particular, for a C-class classification we recast the outputs $\mathbf{y} = \{y_i : i \in \mathbb{N}_n\}$ as $(y_{i1}, ..., y_{iC})$ such that $y_{ip} = 1$ if x_i is in class p and otherwise -1. Hence the outputs are represented by an $n \times C$ *indicator matrix* $\mathbf{Y} = (y_{ip})_{i,p}$ whose p-th column vector is denoted by $\mathbf{y}_p$. Then, taking $\mathbf{K}_y = \mathbf{Y}\mathbf{Y}^\top$, formulation (4) can be extended to the joint optimization problem

$$\min_{\lambda\in\Delta}\mathcal{L}(\lambda) := \sum_{p\in\mathbb{N}_C}\mathbf{y}_p^\top\Big(\sum_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell+\sigma\mathbf{I}_n\Big)^{-1}\mathbf{y}_p+\log\left|\sum_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell+\sigma\mathbf{I}_n\right|. \tag{17}$$

and formulation (5) can be written as

$$\min_{\lambda\in\Delta}\mathcal{L}(\lambda) := \sum_{p\in\mathbb{N}_C}\mathbf{y}_p^\top\Big(\sum_{\ell\in\mathbb{N}_m}\lambda_\ell\mathrm{Tr}\big((\mathbf{K}_y+\sigma\mathbf{I}_n)^{-1}\mathbf{K}_y\big)\Big)\mathbf{y}_p-\log\left|\sum_{\ell\in\mathbb{N}_m}\lambda_\ell\mathbf{K}_\ell+\sigma\mathbf{I}_n\right|. \tag{18}$$

For the protein fold recognition and yeast protein function prediction projects discussed below, we choose $\mathbf{K}_y = \mathbf{Y}\mathbf{Y}^\top$ as stated.

In general, though, $\mathbf{K}_y$ might encode a known structural relationship between labels. For example, in supervised gene or protein network inference (see e.g. [41,42]) the output information corresponds to an adjacency (square) matrix A where $A_{ij} = 1$ means there is an interaction between gene or protein pair (e_i, e_j) of an organism, otherwise $A_{ij} = 0$. In this case, the output kernel matrix $\mathbf{K}_y$ can potentially be chosen as the graph Laplacian defined as $L = \mathrm{diag}(A\mathbf{1}) - A$, where $\mathbf{1}$ is the vector of all ones. It can also be formulated as a diffusion kernel [43] defined by $e^{\beta L} = \mathbf{I} + \beta L + \frac{\beta^2}{2}L^2 + \frac{\beta^3}{3!}L^3 + \ldots$, where hyper-parameter $\beta > 0$. Other potential choices of $\mathbf{K}_y$ can be found in [19,20] for multi-labeled datasets.

Results and Discussion

We mainly evaluate MKLdiv methods (**MKLdiv-dc** and **MKLdiv-conv**) on protein fold recognition, and then consider an extension to the problem of yeast protein function prediction. In these tasks we first compute the kernel weights by MKLdiv and then feed these into a one-against-all multi-class SVM to make predictions. The trade-off parameter in the multi-class SVM is adjusted by 3-fold cross validation over the training dataset. For all experiments with MKLdiv-dc, we choose $\sigma = 10^{-5}$ and for MKLdiv-conv, we tune $\sigma = \{10^{-5}, ..., 10^{-1}\}$ using cross validation. In both methods, we use a stopping criterion of $\varepsilon = 10^{-5}$ and initialize the kernel weight λ by setting $\lambda_\ell = \frac{1}{m}$ for any $\ell \in \mathbb{N}_m$ where m is the number of candidate kernel matrices.

Synthetic Data

We first validated the proposed MKLdiv algorithms on a simple three-class dataset illustrated in subfigure (a) of Figure 1. As in [11], we use a Gaussian kernel with unit variance, a polynomial kernel of order two and a linear kernel. In this case we demonstrate the effect of our approaches on combining kernel matrices derived from a single data source. Subfigures (e) and (f) of Figure 1 illustrate the kernel weights learned by MKLdiv-dc and MKLdiv-conv. In particular, MKLdiv-dc successfully picked up the Gaussian kernel as the most dominant kernel, which is more reasonable than MKLdiv-conv. Subfigures (b) and (c) of Figure 1 show the relative change of objective function values versus iteration, i.e. $((\lambda^{(t-1)}) - (\lambda^{(t)}))/(\lambda^{(t)})$, of MKLdiv-dc and MKLdiv-conv. We can see that the DC algorithm of MKLdiv-dc converges quickly to a local minimum while the projected gradient descent algorithm converges a little slower to a global minimum. However, MKLdiv-dc needs more time per iteration than MKLdiv-conv since MKLdiv-dc needs to solve the subproblem (13) at each iteration. As mentioned before, the subproblem (13) can be solved by either semi-infinite linear programming (SILP) or a projected gradient descent (PGD) method. To see their convergence, in subfigure (d) of Fig-

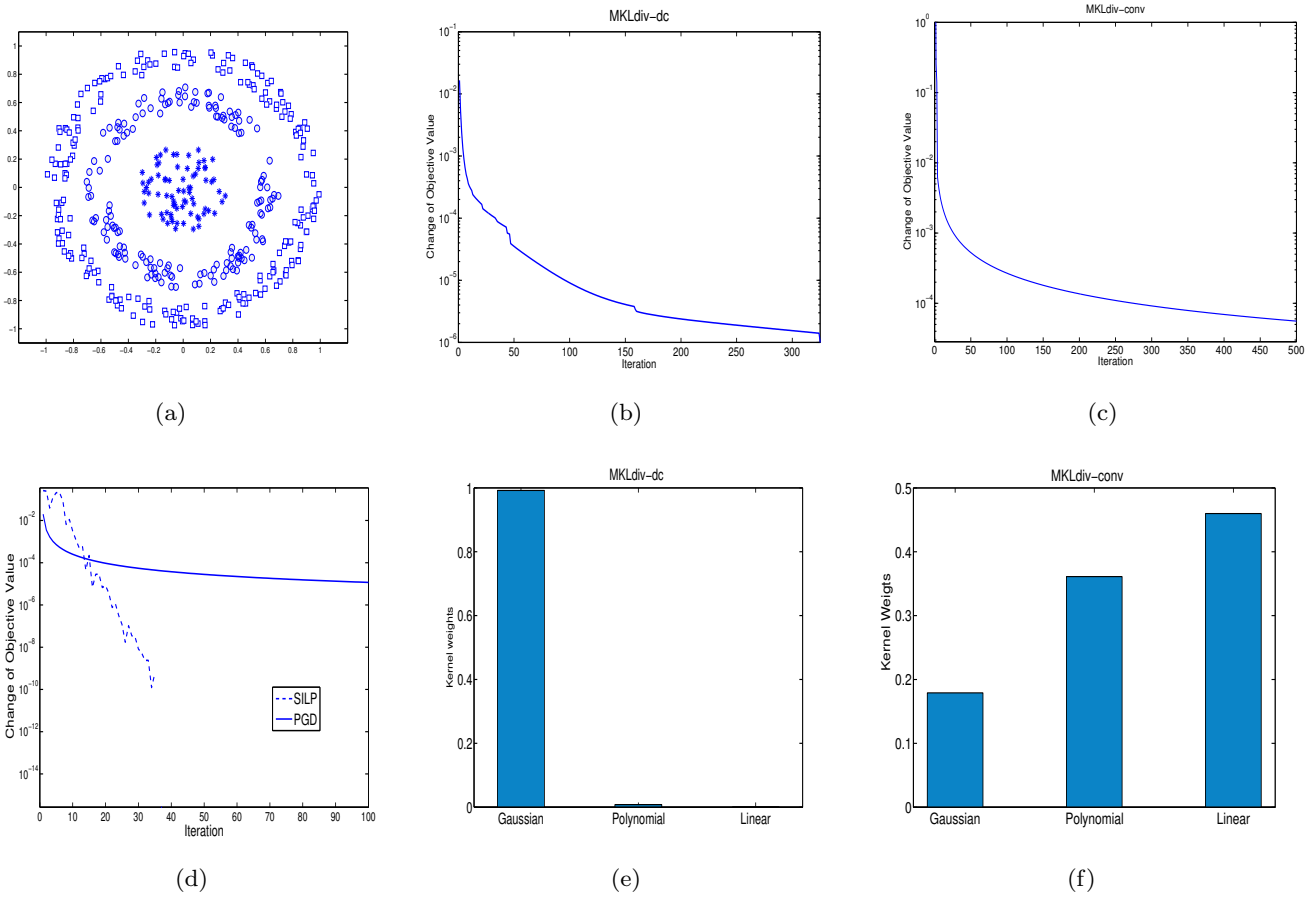


Figure 1

Synthetic data verification. Synthetic data verification of MKLdiv: (a) depiction of the three-circle dataset; (b) relative change of objective values of MKLdiv-dc versus iteration number of CCCP; (c) relative change of objective values of MKLdiv-conv versus iteration number of projected gradient descent (PGD) method; (d) relative change of objective values of subproblem (13) by SILP (dish-line) and PGD methods; (e) kernel weights learned by MKLdiv-dc; (f) kernel weights learned by MKLdiv-conv.

ure 1 we plot the relative changes of the objective function in subproblem (13) when $\lambda_\ell^{(t)} = 1/m$ for $\ell \in \mathbb{N}_m$. We can see from subfigure (d) that the PGD approach converges faster in the beginning but stalls at a higher precision and the SILP method converges faster at higher precision.

Protein Fold Recognition

Next we evaluated MKLdiv on a well-known protein fold prediction dataset [3]. This benchmark dataset (based on SCOP PDB-40D) has 27 SCOP fold classes with 311 proteins for training and 383 for testing. This dataset was originally proposed by Ding and Dubchak [3] and it has 313 samples for training and 385 for testing. There is less than 35% sequence identity between any two proteins in the training and test set. We follow Shen and Chou [4] who proposed to exclude two proteins from the training and test datasets due to a lack of sequence information. We compare our MKLdiv methods with kernel learning based on one-against-all multiclass SVM using the SimpleMKL software package [44], kernel learning for regularized discriminant analysis (MKL-RKDA) [18]http://www.public.asu.edu/~jye02/Software/DKL/ and a probabilistic Bayesian model for kernel learning (VBKC) [21]. The trade-off parameters in SimpleMKL and MKL-RKDA were also adjusted by 3-fold cross validation on the training set.

Description of the Fold Discriminatory Data Sources

As listed in Table 1, there are a total of 12 different fold discriminatory data sources available: Amino Acid Composition (C), Predicted Secondary Structure (S), Hydrophobicity (H), Polarity (P), van der Waals volume (V), Polarizability (Z), PseAA λ = 1 (L1), PseAA λ = 4 (L4), PseAA λ = 14 (L14), PseAA λ = 30 (L30), SW with

Table 1: Performance with individual and all data sources

Data sources	MKLdiv-dc	MKLdiv-conv	SimpleMKL	VBKC	MKL-RKDA
Amino acid composition (C)	51.69	51.69	51.83	51.2 ± 0.5	45.43
Predicted secondary structure (S)	40.99	40.99	40.73	38.1 ± 0.3	38.64
Hypdrophobicity (H)	36.55	36.55	36.55	32.5 ± 0.4	34.20
Polarity (P)	35.50	35.50	35.50	32.2 ± 0.3	30.54
van der Walls volume (V)	37.07	37.07	37.85	32.8 ± 0.3	30.54
Polarizability (Z)	37.33	37.33	36.81	33.2 ± 0.4	30.28
PseAA λ = 1 (L1)	44.64	44.64	45.16	41.5 ± 0.5	36.55
PseAA λ = 4 (L4)	44.90	44.90	44.90	41.5 ± 0.4	38.12
PseAA λ = 14 (L14)	43.34	43.34	43.34	38 ± 0.2	40.99
PseAA λ = 30 (L30)	31.59	31.59	31.59	32 ± 0.2	36.03
SW with BLOSUM62 (SW1)	62.92	62.92	62.40	59.8 ± 1.9	61.87
SW with PAM50 (SW2)	63.96	63.96	63.44	49 ± 0.7	64.49
All data sources	**73.36**	**71.01**	66.57	**68.1 ± 1.2**	**68.40**
Uniform weighted	68.40	68.40	**68.14**	-	66.06

The results of VBKC are cited from [21]. The results not employed there are denoted by '-'. The best result for each kernel learning method is marked in bold.

BLOSUM62 (SW1) and SW with PAM50 (SW2). The first six data sources were originally from [3]. Four data sources using different dimensions of pseudo-amino acid composition (PseAA) were introduced in [4] to replace the amino-acid composition. The last two data sources used in [21] are derived from a pairwise kernel [45] for local sequence alignment based on Smith-Waterman scores.

As in [21], we employ linear kernels (Smith-Waterman scores) for SW1 and SW2 and second order polynomial kernels for the other data sources. Ding and Dubchak [3] conducted an extensive study on the use of various multi-class variants of standard SVMs and neural network classifiers. For these authors the best test set accuracy (TSA) was 56%, and the most informative among their six data sources (CSHPVZ) were amino-acid composition (C), the predicted secondary structure (S) and hydrophobicity (H). Shen and Chou [4] introduced four additional PSeAA data sources to replace the amino acid composition (C) and raised test performance to 62.1%. The latter authors used an *ad hoc* ensemble learning approach involving a combination of multi-class k nearest neighbor classifiers individually trained on each data source. Recently, test performance was greatly improved by Damoulas and Girolami [21] using a Bayesian multi-class multi-kernel algorithm. They reported a best test accuracy of 70% on a single run.

Performance with Individual and All Data Sources

We ran MKLdiv-dc, MKLdiv-conv, SimpleMKL and MKL-RKDA on the overall set of 12 data sources, also evaluating performance on a uniformly weighted (averaged) composite kernel in addition to individual performance on each separate data source. In Table 1 we report the test set accuracy on each individual data source. The performance of MKLdiv-dc and MKLdiv-conv inclusive of all data sources achieves a test set accuracy of 73.36% and 71.01% respectively, consistently outperforming all individual performances and the uniformly weighted composite kernel (68.40%). Moreover, individual performance for MKLdiv-dc, SimpleMKL and MKL-RKDA indicates that the most informative data sources are local sequence alignments (SW1 and SW2) and the amino acid composition (C). The performance with individual data sources for MKLdiv-dc, MKLdiv-conv, and SimpleMKL are almost the same since, for a fixed kernel, they use the same one-against-all multi-class SVM.

From Table 1, performances of MKLdiv-dc and MKLdiv-conv with all the available data sources achieve test set accuracies of 73.36% and 71.01%, both of which outperform the state-of-art performance 70% on a single run reported in [21] and other kernel learning methods including SimpleMKL (66.57%) and MKL-RKDA (68.40%). The performance of the uniformly weighted kernel is 68.14% which is better than the performance 66.57% of SimpleMKL. This indicates that sparse L^1-regularization does not necessarily yield better performance. The kernel weights λ of MKLdiv-dc, SimpleMKL, and MKL-RKDA are shown in subfigures (b), (e) and (g) of Figure 2 which indicates that Amino Acid Composition (C), predicted secondary structure (S), Hypdrophobicity (H), and the last two data sources SW1 and SW2 are the most informative data sources, and the remaining data sources of H, P, V, and PseAA are less informative. As depicted in the subfigure (b) of Figure 2, MKLdiv-dc and MKLdiv-conv include some less informative data sources such as P, Z, L1, L4, L14, L30 etc., with small (but not

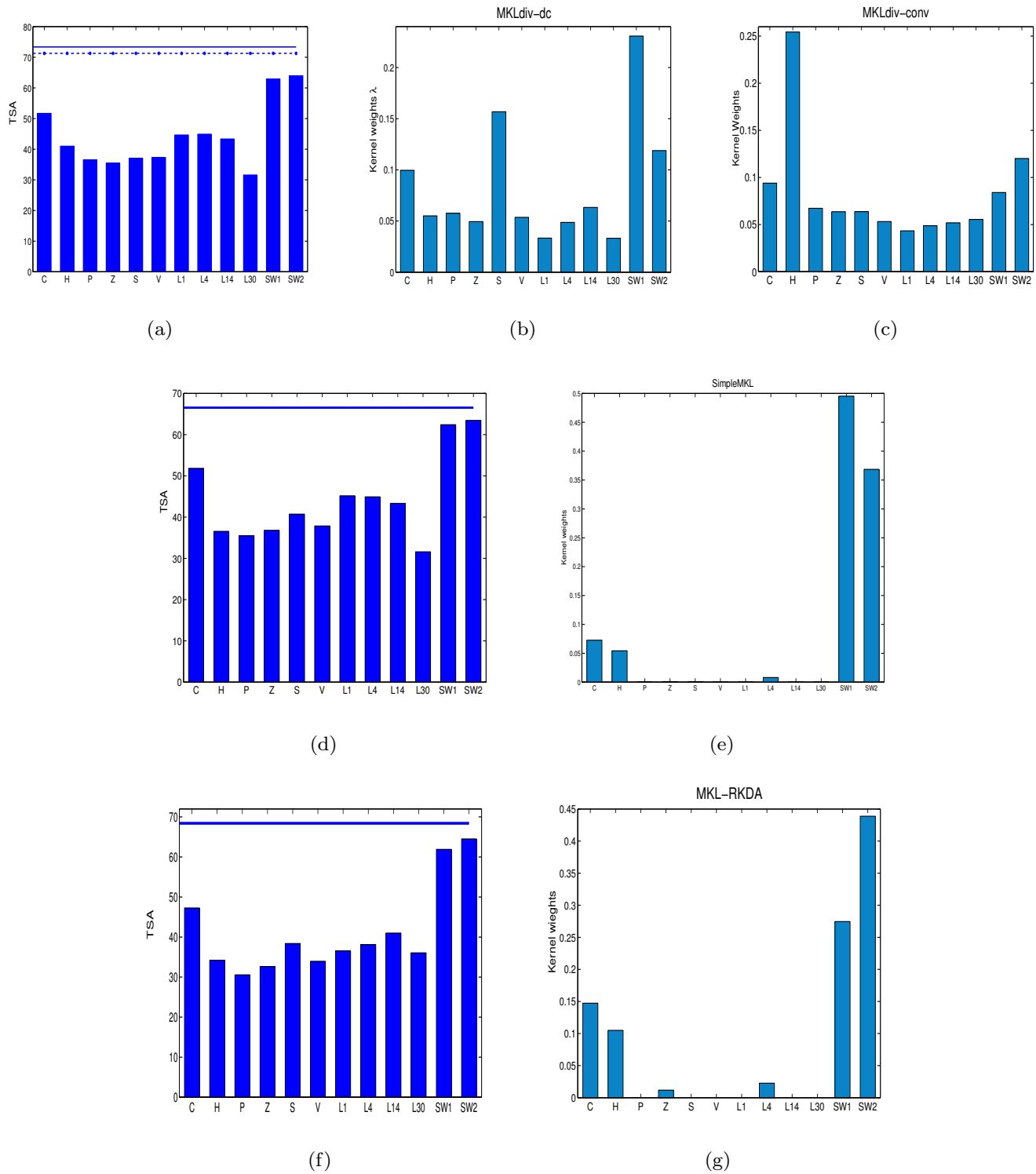


Figure 2

Performance with all data sources on protein fold recognition. Test set accuracy of individual (bars) and all data sources (horizontal lines) on the protein fold recognition dataset: (a) MKLdiv-dc and MKLdiv-conv, where the solid line is the performance of MKLdiv-dc and the star-dashed line is the performance of MKLdiv-conv; (d) SimpleMKL; (f) MKL-RKDA. Kernel weights: (b) MKLdiv-dc, (c) MKLdiv-conv, (e) SimpleMKL and (g) MKL-RKDA.

zero) kernel weights. In contrast, as shown in (e) and (g) of Figure 2, SimpleMKL and MKL-RKDA completely discard these less informative data sources. However, as shown in (d) and (f) of Figure 2, SimpleMKL and MKL-RKDA achieve poorer performance, less than 70%, while MKLdiv-dc achieves 73.36% and MKLdiv-conv achieves 71.01%. This suggests that MKLdiv-dc provides a more reasonable balance over the entire set of data sources. This observation also suggests that achieving a sparsity among kernel weights does not necessarily guarantee good generalization performance since some available data sources may be weakly informative but may still carry some useful additional information.

Performance with Sequential Addition of Data Sources

As mentioned above, the kernel weights learned by MKLdiv on all the data sources can provide useful insights into the significance of informative data sources. Hence, we further investigated the effect of sequentially adding data sources based on information from learned kernel weights in Tables 2 and 3. Without loss of generality, we take the kernel weights learned by MKLdiv-dc as an example.

We first report in Table 2 the effect of sequentially adding the sources in the order which was used in [3] and [21] and MKLdiv-dc and MKLdiv-conv consistently outperform the competitive kernel learning methods VBKC,

Table 2: Effects of sequentially adding data sources

Data sources	MKLdiv-dc	MKLdiv-conv	VBKC	SimpleMKL	MKL-RKDA
C	51.69	51.69	51.2 ± 0.5	51.69	47.25
CS	56.39 (20.23 s)	55.35 (0.32 s)	55.7 ± 0.5 (-)	55.61 (14.67 s)	48.30 (0.15 s)
CSH	57.70 (50.35 s)	58.22 (2.44 s)	57.7 ± 0.6 (-)	56.91 (10.40 s)	55.61 (0.12 s)
CSHP	58.48 (39.02 s)	53.52 (72.14 s)	57.9 ± 0.9 (-)	57.96 (17.84 s)	56.65 (0.18 s)
CSHPV	60.05 (75.05 s)	53.26 (86.39 s)	58.1 ± 0.8 (-)	57.96 (15.05 s)	55.87 (0.17 s)
CSHPVZ	59.26 (135.08 s)	53.52 (99.64 s)	58.6 ± 1.1 (-)	59.00 (20.02 s)	57.70 (0.20 s)
CSHPVZL1	60.05 (221.75 s)	52.74 (122.74 s)	60.0 ± 0.8 (-)	61.35 (27.38 s)	57.70 (0.21 s)
CSHPVZL1L4	62.14 (315.70 s)	52.74 (129.08 s)	60.8 ± 1.1 (-)	61.61 (151.38 s)	58.22 (0.25 s)
CSHPVZL1L4L14	62.14 (450.57 s)	61.09 (57.09 s)	61.5 ± 1.2 (-)	60.05 (42.81 s)	59.53 (0.25 s)
CSHPVZL1L4L14L30	62.14 (612.72 s)	62.14 (67.29 s)	62.2 ± 1.3 (-)	62.40 (64.74 s)	55.61 (0.25 s)
CSHPVZL1L4L14L30SW1	71.80 (620.16 s)	**71.54** (17.97 s)	66.4 ± 0.8 (-)	65.79 (78.94 s)	66.84 (0.31 s)
CSHPVZL1L4L14L30SW1SW2	**73.36** (805.11 s)	71.01 (84.21 s)	**68.1 ± 1.2** (-)	**66.57** (196.42 s)	**68.40** (0.31 s)
SHPVZL1L4L14L30	60.57 (438.89 s)	61.09 (67.92 s)	61.1 ± 1.4 (-)	59.00 (44.79 s)	54.56 (0.25 s)

The result of Bayesian kernel learning model (VBKC) is cited from [21]. The results not employed there are denoted by '-'. The term inside the parenthesis is the CPU running time (seconds). The best test set accuracy of each kernel learning method is marked in bold.

Table 3: Effects of sequentially adding data sources (continued)

Data sources	MKLdiv-dc	MKLdiv-conv	SimpleMKL	MKL-RKDA
SW1	62.92	62.92	62.40	61.87
SW1S	65.27 (24.72 s)	66.31 (10.49 s)	64.22 (40.60 s)	64.75 (0.12 s)
SW1SW2S	67.10 (48.79 s)	66.05 (4.65 s)	64.75 (61.71 s)	64.49 (0.15 s)
SW1SW2CS	73.36 (40.65 s)	72.32 (23.43 s)	65.01 (62.81 s)	67.62 (0.17 s)
SW1SW2CSH	74.67 (72.19 s)	72.32 (8.69 s)	66.31 (75.11 s)	67.88 (0.15 s)
SW1SW2CSHP	74.93 (123.98 s)	**74.41** (11.63 s)	66.31 (74.85 s)	**69.71** (0.18 s)
SW1SW2CSHPZ	**75.19** (189.91 s)	73.36 (15.00 s)	**68.92** (109.09 s)	66.05 (0.20 s)
SW1SW2CSHPZV	74.41 (278.47 s)	**74.41** (17.47 s)	66.31 (117.14 s)	69.19 (0.25 s)
SW1SW2CSHPZVL1	73.10 (404.82 s)	73.32 (49.41 s)	66.84 (101.01 s)	68.66 (0.25 s)
SW1SW2CSHPZVL1L4	72.84 (576.29 s)	72.06 (57.83 s)	67.10 (107.88 s)	67.62 (0.25 s)
SW1SW2CSHPZVL1L4L14	72.58 (748.72 s)	72.36 (19.43 s)	66.84 (163.85 s)	69.19 (0.28 s)
SW1SW2CSHPZVL1L4L14L30	73.36 (811.54 s)	71.01 (83.93 s)	66.57 (197.57 s)	68.40 (0.31 s)

Test set accuracy of sequentially adding fold discriminatory data sources (continued) according to the ranking of kernel weights obtained by MKLdiv-dc over all data sources. The results of the Bayesian kernel learning method were not employed in [21], hence we do not list in the table. The term inside the parenthesis is the CPU running time (seconds). The best test set accuracy of each kernel learning method is marked in bold.

SimpleMKL, MKL-RKDA and the best performing SVM combination methodology stated in [3]. As suggested by the kernel weights of MKLdiv-dc in the subfigure (b) of Figure 2, the sequence alignment based data source SW1 is most informative, then S, then SW2 and so on. Hence, in Table 3 we further report the effect of sequentially adding data sources in this rank order. As shown in Table 3, there is a significant improvement over SW1SW2 in MKLdiv-dc when we sequentially add the data sources of amino acid composition (C) and predicted secondary structure (S). The performance of MKLdiv-dc keeps increasing until we include CSHPZ, giving the best performance of 75.19%. Although according to [4], the PseAA data sources are believed to contain more information than the conventional amino acid composition. The same behaviour appears for MKLdiv-conv. However, the MKLdiv-dc performance degenerates if we continue to add PseAA composition data sources and the same behaviour appears for MKLdiv-conv. Similar observations were made by [21] which suggests that PseAA measurements may carry non-complementary information with the conventional amino acid compositions.

With regard to the best performance of MKLdiv-dc with the feature set SW1SW2CSHPZ, we display the corresponding kernel weights in Figure 3. We can see in Figure 3 that SimpleMKL and MKL-RKDA almost eliminate the informative feature set HPZ while MKLdiv-dc and MKLdiv-conv include them into the composite kernel. The sparse L^1-regularization of SimpleMKL and MKL-RKDA accounts for the sparse weights of SimpleMKL and MKL-RKDA.

Comparison of Running Time

To investigate the run-time efficiency of MKLdiv on protein fold recognition dataset, we list their CPU time in

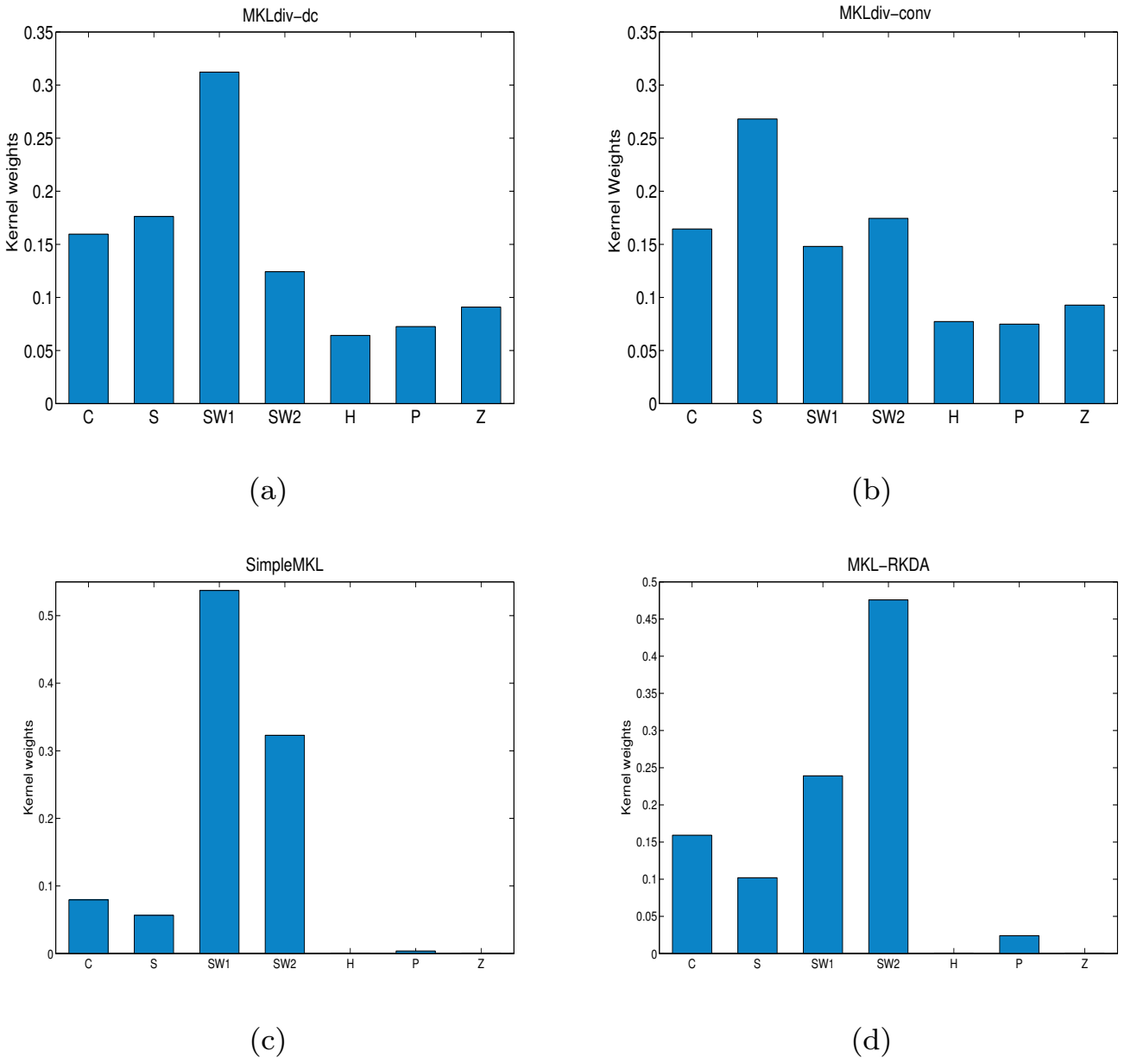


Figure 3
Kernel weights on dominant data sources for protein fold recognition. Kernel weights on the dominant data sources SW1SW2CSHPZ which yields the best prediction on the protein fold recognition dataset: (a) MKLdiv-dc, (b) MKLdiv-conv, (c) SimpleMKL and (d) MKL-RKDA.

Tables 2 and 3. The running time (in seconds) is the term inside the parenthesis. The SILP approach for MKL-RKDA is very efficient while SimpleMKL takes a bit longer. The reason could be that MKL-RKDA essentially used the least-square loss for multi-class classification in contrast to the one-against-all SVM used in SimpleMKL. Generally, more time is required to run the interior method for one-against-all SVM than directly computing the solution of the least-square regression. The projected gradient descent method for MKLdiv-conv is also slower than MKL-RKDA. It is to be expected that MKLdiv-conv converges faster than MKLdiv-dc since the DC algorithm for MKLdiv-dc is non-convex and it needs to solve the subproblem (13) in each iteration of CCCP. Nevertheless, the price we paid in run-

ning time for MKLdiv-dc is worthwhile given its significantly better performance on the protein fold prediction problem.

Sensitivity against Parameter σ

The initial purpose of introducing σ is to avoid the singularity of the input kernel matrix or the output kernel matrix. However, in practice we found that, in the convex formulation MKLdiv-conv, values of σ have a great influence on performance for protein fold recognition. Hence, when we ran MKLdiv-conv, we always did cross validation over the training set to select the parameter σ. To see how sensitive the test set accuracy is with respect to σ, in Figure 4 we depicted the test set accuracy versus values of σ. In Figure 4 we can observe that the test set accuracy of MKLdiv-dc is relatively stable for small values of σ's. However, this is not the case for MKLdiv-conv and generally suggests that the parameter σ has a great impact on performance of MKLdiv-conv. This could be because the output kernel matrix $\mathbf{K}_y = \mathbf{Y}\mathbf{Y}^\top$ is of low rank (rank one in binary classification) and thus adding a small matrix $\sigma\mathbf{I}_n$ in the formulation MKLdiv-conv could dramatically change the information of the output kernel matrix. In contrast, we can reasonably assume the input kernel matrices are non-singular or not of low rank and the effect of adding a small matrix $\sigma\mathbf{I}_n$ in the formulation MKLdiv-dc can be ignored.

Extension of Investigation to Yeast Protein Classification

We next extend our investigation of MKLdiv-dc and MKLdiv-conv on a yeast membrane protein classification problem [23]. This binary classification task has 2316 examples derived from the MIPS comprehensive Yeast Genome Database (CYGD) (see [46]). There are eight kernel matrices http://noble.gs.washington.edu/proj/sdp-svm/. The first three kernels (K_{SW}, K_{B}, and K_{Pfam}) are respectively designed to measure the similarity of protein sequences using BLAST, Smith-Waterman pairwise sequence comparison algorithms and a generalization of pairwise comparison method derived from hidden Markov models. The fourth sequence-based kernel matrix (K_{FFT}) incorporates information about hydrophobicity which is known to be useful in identifying membrane proteins, computed by Fast Fourier Transform. The fifth and sixth kernel matrices (K_{LI}, K_{D}) are respectively derived from linear and diffusion kernels based on protein-protein interaction information. The seventh kernel matrix (K_{E}) is a Gaussian kernel encoding gene expression data. Finally, we added a noise kernel matrix K_{Ran} generated by first generating random numbers and then using a linear kernel.

The performance of MKLdiv-dc and MKLdiv-conv is evaluated by 10 random partitions of the data into a training and test set in a proportion of 4: 1. We report the receiver

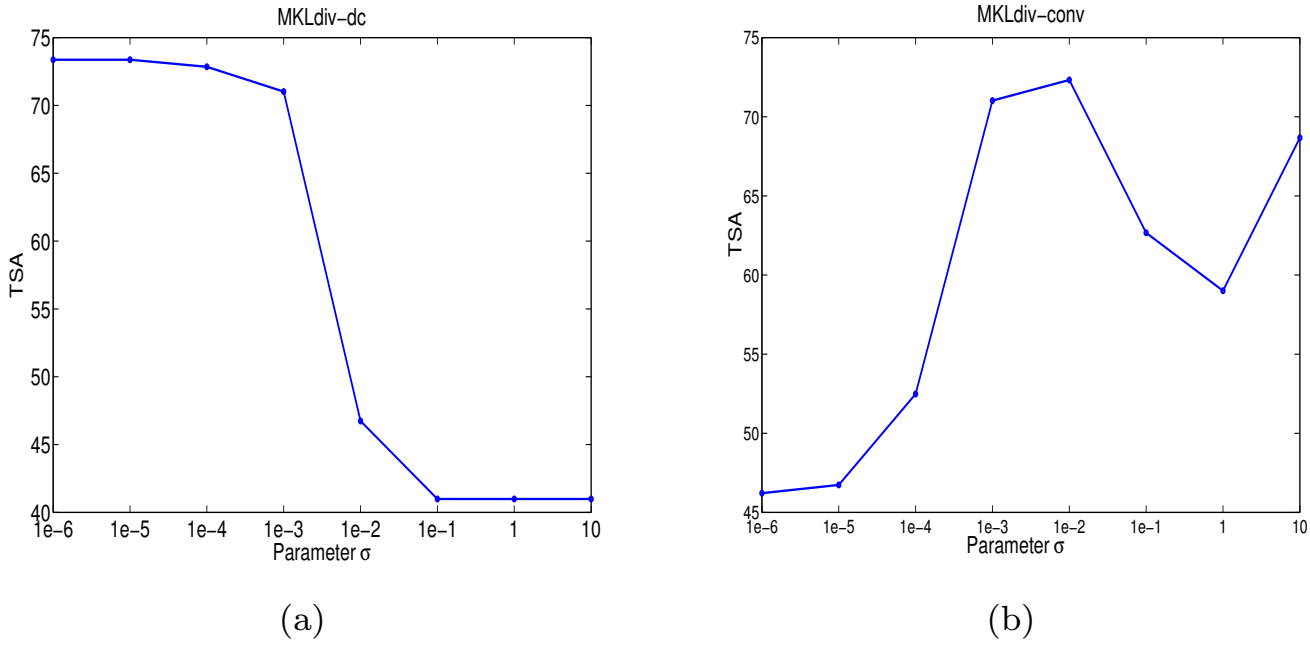
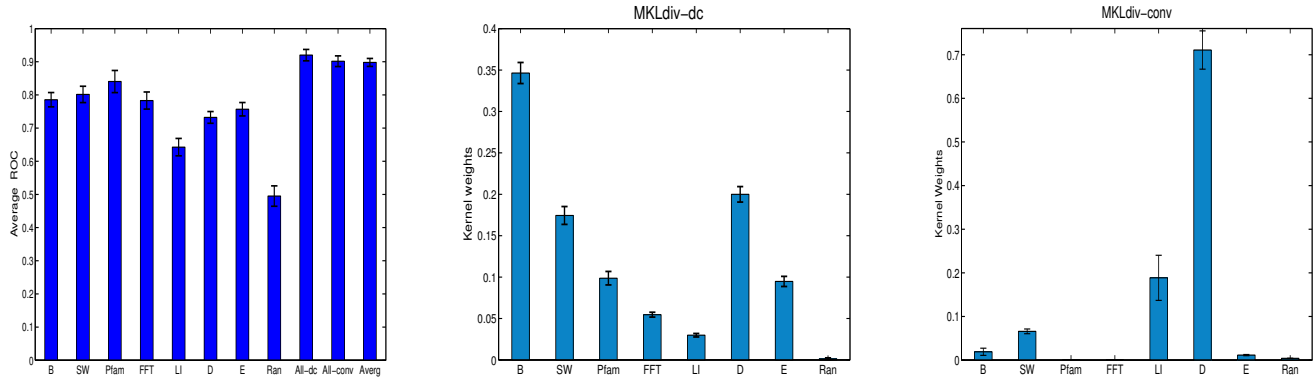


Figure 4
Sensitivity against parameter σ for protein fold recognition. Test set accuracy versus different values of σ on the protein fold recognition dataset: (a) MKLdiv-dc and (b) MKLdiv-conv.



(a) (b) (c)

Figure 5

Performance of MKLdiv on yeast membrane protein. Performance on the yeast membrane protein function dataset: (a) average ROC score for individual data sources, using MKLdiv-dc and MKLdiv-conv, where the third bar to last (All-dc) is MKLdiv-dc, the second bar to last (All-conv) is MKLdiv-conv and the last bar (Averg) is the performance using uniformly weighted kernels. Kernel weights: (b) MKLdiv-dc and (c) MKLdiv-conv.

operating characteristic (ROC) score, which measures the overall quality of the ranking induced by the classifier, rather than the quality of a single point in that ranking. The first subfigure of Figure 5 shows the performance with individual kernels and the performance of MKLdiv-dc (the third to last bar), MKLdiv-conv (the next to last bar), and the uniformly weighted kernel (last bar). Specifically, MKLdiv-dc yields a ROC score of 0.9189 ± 0.0171 which is competitive with the result in [23]. MKLdiv-conv, however, achieved a ROC score of 0.9016 ± 0.0161 which is worse than MKLdiv-dc. The performance of MKLdiv-dc is also slightly better than the performance of the uniformly weighted kernel 0.9084 ± 0.0177 excluding the noise kernel and 0.8979 ± 0.0120 including the noise kernel. We also plot the kernel weights on (b) and (c) of Figure 5. As expected, in MKLdiv-dc the BLAST kernel (K_B) derived from the protein sequence similarity comparison is very informative which is consistent with [23]. The derived kernel weights also show that the interaction-based diffusion kernel is more informative than the expression kernel, which is consistent with [23]. Also, it is interesting to note that MKLdiv-dc shows that the noise kernel (K_{Ran}) is least informative. This is indicated by its individual ROC score: a ROC score around 0.5 corresponds to random ranking. The kernel weights of MKLdiv-conv indicate that the diffusion kernel (D) is the most important data source, and also suggest that Pfam and FFT are almost non-informative regardless of their good individual performances. For the kernel weights, MKLdiv-dc is more reasonable than MKLdiv-conv since MKLdiv-dc is more consistent with the individual data source's performance and MKLdiv-dc outperforms MKLdiv-conv using all data sources.

Conclusion

In this paper we developed a novel information-theoretic approach to learning a linear combination of kernel matrices based on the KL-divergence [24-28], especially focused on the protein fold recognition problem. Based on the different position of the input kernel matrix and the output kernel matrix in the KL-divergence objective, there are two formulations. The first one is a difference of convex (DC) problem termed MKLdiv-dc and the second formulation is a convex formulation called MKLdiv-conv. The sparse formulation for kernel learning based on discriminant analysis [18] was also established. Our proposed methods are able to achieve state-of-the-art performance on the SCOP PDB-40D benchmark dataset for protein fold recognition problem. In particular, MKLdiv-dc further improves the fold discrimination accuracy to 75.19% which is a more than 5% improvement over a competitive Bayesian probabilistic approach [21], SVM margin-based kernel learning methods [11], and the kernel learning based on discriminant analysis [18]. We further extended the investigation to the problem of yeast protein function prediction.

Generally, it is difficult to determine which criterion is better for multiple kernel combination since this problem is highly data-dependent. For the information-theoretic approaches MKLdiv-dc and MKLdiv-conv, although MKLdiv-dc is not convex and its DC algorithm tends to find a local minima, in practice we would recommend MKLdiv-dc for the following reasons. Firstly, as mentioned above MKLdiv-dc has a close relation with the kernel matrix completion problem using information geometry [27,28] and the maximization of the log likelihood of Gaussian

Process regression [35], which partly explains the success of MKLdiv-dc. Secondly, we empirically observed that MKLdiv-dc outperforms MKLdiv-conv in protein fold recognition and yeast protein function prediction. Finally, as we showed in Figure 4, the performance of MKLdiv-conv is quite sensitive to the parameter σ and the choice of σ remains a challenging problem. MKLdiv-dc is relatively stable with respect to small values of σ and we can fix σ to be a very small number e.g. $\sigma = 10^{-5}$. In future, we are planning to empirically compare performance with other existing kernel integration formulations on various datasets, and discuss convergence properties of the DC algorithm for MKLdiv-dc based on the theoretical results of [36].

Authors' contributions

YY and CC conceived the project. YY proposed and implemented the method, drafted the manuscript. KH joined the project and participated in the design of the study. All authors read and improved the manuscript.

Appendix

Appendix 1 – Column generation method for SILP

Here we briefly describe the column generation method (see e.g. [40]) for SILP (16) to solve the subproblem (15), i.e.

$$\begin{aligned} &\max_{\gamma,\lambda} \quad \gamma \\ &\text{s.t.} \quad \sum_{\ell=1}^{m} \lambda_\ell = 1, \lambda_\ell \geq 0 \\ &\qquad \gamma - \sum_{\ell=1}^{m} \lambda_\ell S_\ell(\alpha) \leq S_0(\alpha), \ \forall \alpha \end{aligned} \tag{19}$$

where $S_\ell(\alpha) = \mathrm{Tr}(\alpha\alpha^\top K_\ell) + \frac{\partial g(\lambda^{(t)})}{\partial \lambda_\ell}$, and $S_0(\alpha)$ = -2Tr($\alpha^\top A$) + σTr($\alpha^\top \alpha$). The basic idea is to compute the optimum (λ, γ) by linear programming for a restricted subset of constraints, and update the constraint subset based on the obtained suboptimal (λ, γ). More precisely, given restricted constraints $\{\alpha_p : p = 1, ..., P\}$, first we find the intermediate solution (λ, γ) by the following linear programming optimization with P linear constraints

$$\begin{aligned} &\max_{\gamma,\lambda} \quad \gamma \\ &\text{s.t.} \quad \sum_{\ell} \lambda_\ell = 1, \ 0 \leq \lambda \leq 1 \\ &\qquad \gamma - \sum_{\ell} \lambda_\ell S_\ell(\alpha_p) \leq S_0(\alpha_p), \ \forall p = 1, \ldots, P. \end{aligned} \tag{20}$$

This problem is often called the *restricted master problem*. Then, we find the next constraint with the maximum violation for the given intermediate solution (λ, γ), i.e.

$$\min_{\alpha} \sum_{\ell \in \mathbb{N}_d}^{d} \lambda_\ell S_\ell(\alpha) + S_0(\alpha). \tag{21}$$

If the optimizer α^* of the above equation satisfies $\sum_\ell \lambda_\ell S_\ell(\alpha^*) + S_0(\alpha) \geq \gamma$ then the current intermediate solution (λ, γ) is optimal for the optimization (19). Otherwise α^* should be added to the restriction set. We repeat the above iteration until convergence which is guaranteed to be globally optimal, see e.g. [14,40]. In a similar fashion to the convergence criterion in [14], the algorithm stops when

$$\left| 1 - \frac{\sum_\ell \lambda_\ell^{(t-1)} S_\ell(\alpha^{(t)}) + S_0(\alpha^{(t)})}{\gamma^{(t-1)}} \right| \leq \varepsilon.$$

For instance, the threshold ε is usually chosen to be 5×10^{-4}.

Appendix 2 – Sparse formulation of kernel learning based on discriminant analysis

In this appendix we show that kernel learning for regularized discriminant analysis [18] is closely related to sparse regularization. To see this, consider the following algorithm

$$\begin{aligned} &\min_{\mathrm{f,b}} \quad \sum_{i \in \mathbb{N}_n} (y_i - \sum_{\ell \in \mathbb{N}_m} f_\ell(x_i^\ell) - b)^2 + \frac{1}{2} (\sum_{\ell \in \mathbb{N}_m} || f ||_{\mathcal{H}_\ell})^2 \\ &\text{s.t.} \quad f_\ell \in \mathcal{H}_\ell, \ \ell \in \mathbb{N}_m \end{aligned}$$

Using the fact [31] that $\min \{\sum_{\ell \in \mathbb{N}_m} || f_\ell ||_{K_\ell}^2 / \lambda_\ell : \lambda \in \Delta\} = \left(\sum_\ell || f_\ell ||_{K_\ell} \right)^2$, the above equation is identical to

$$\begin{aligned} &\min_{\mathrm{f},\lambda,\mathrm{b}} \quad \mu \sum_{i \in \mathbb{N}_n} (y_i - \sum_{\ell \in \mathbb{N}_m} f_\ell(x_i^\ell) - b)^2 + \frac{1}{2} \sum_{\ell \in \mathbb{N}_m} \frac{||f_{\ell c}||_{\mathcal{H}_\ell}^2}{\lambda_\ell} \\ &\text{s.t.} \quad \lambda \in \Delta, f_\ell \in \mathcal{H}_\ell, \ \forall \ell \in \mathbb{N}_m. \end{aligned} \tag{22}$$

The equivalence between the above algorithm and RKDA kernel learning becomes clear if we formulate its dual problem as follows:

Theorem 2 *Let* $\mathbf{K}_\lambda = (\sum_{\ell\in\mathbb{N}_m} \lambda_\ell K_\ell(x_i^\ell, x_j^\ell))_{ij\in\mathbb{N}_n}$, $\mathbf{I}_n$ *be the identity matrix and* $\mathbf{1}_n$ *be an n-dimensional column vector of all ones. Define* $P = \mathbf{I}_n - \frac{\mathbf{1}_n\mathbf{1}_n^\top}{n}, \tilde{\mathbf{K}}_\lambda = P\mathbf{K}_\lambda P$, *and* $\overline{\gamma}_i = \gamma_i - \sum_{\ell\in\mathbb{N}_n} \gamma_j$ *for any* $i \in \mathbb{N}_n$. *Then, the dual problem of algorithm (22) can be written as*

$$\min_{\lambda\in\Delta} \max_{\alpha} \sum_i \alpha_i\overline{\gamma}_i - \frac{1}{4}\sum_i \alpha_i^2 - \frac{1}{4\gamma}\sum_{i,j} \alpha_i\alpha_j\tilde{K}_\lambda(x_i, x_j),$$

where $\gamma = \frac{1}{2\mu}$.

Proof: Taking the minimization of b first, algorithm (22) yields $b = \frac{1}{n}\sum_{i\in\mathbb{N}_n}(\gamma_i - \sum_{\ell\in\mathbb{N}_m} f_\ell(x_i^\ell))$. Then, algorithm (22) can be further rewritten as

$$\begin{aligned} \min_{\mathrm{f},\lambda} \quad & \mu\sum_{i\in\mathbb{N}_n}(\overline{\gamma}_i - \sum_{\ell\in\mathbb{N}_m}\overline{f}_\ell(x_i^\ell))^2 + \frac{1}{2}\sum_{\ell\in\mathbb{N}_m}\frac{||f_\ell||^2_{\mathcal{H}_\ell}}{\lambda_\ell} \\ \text{s.t.} \quad & \lambda\in\Delta, f_\ell\in\mathcal{H}_\ell,\ \forall\ell\in\mathbb{N}_m. \end{aligned} \tag{23}$$

Here, for any ℓ and i, $\overline{f}_\ell(x_i^\ell) = f_\ell(x_i^\ell) - \frac{1}{n}\sum_{j\in\mathbb{N}_n} f_\ell(x_j^\ell)$ which can be further represented by $\overline{f}_\ell(x_i^\ell) = \left\langle K_\ell(x_i^\ell,\cdot) - \frac{1}{n}\sum_{j\in\mathbb{N}_n} K_\ell(x_j^\ell,\cdot), f_\ell\right\rangle_{K_\ell}$. Then, letting $\xi_i = \overline{\gamma}_i - \sum_{\ell\in\mathbb{N}_m}\overline{f}_\ell(x_i)$ for any i and solving the standard Lagrangian formulation of (23) with Lagrangian variables α yields

$$\min_{\lambda\in\Delta}\max_{\alpha} \sum_i \alpha_i\overline{\gamma}_i - \frac{1}{4\mu}\sum_i \alpha_i^2 - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j\tilde{K}_\lambda(x_i, x_j).$$

Now, replacing α_i by $\mu\alpha_i$ and letting $\mu = \frac{1}{2\gamma}$ completes the argument. □

Let n_- and n_+ denote the number of samples in class +1 and −1. If we redefine the class indicator output $\mathbf{y}$, for any $i \in \mathbb{N}_n$ by $\gamma_i = \frac{n}{2n_-n_+}$ if x_i is in class +1 otherwise - $\frac{n}{2n_-n_+}$, then the class indicator output $\overline{\mathbf{y}}$ reduces to the vector a defined in [18] for binary classification, i.e.

$$\overline{\gamma}_i = a_i = \begin{cases} \frac{1}{n_+}, & \text{if } x_i \text{ is in class } +1 \\ -\frac{1}{n_-}, & \text{otherwise.} \end{cases}$$

Now we turn our attention to multiclass classification. To this end, consider

$$\begin{aligned} \min_{\mathrm{f,b}} \quad & \mu\sum_{i\in\mathbb{N}_n}\sum_{c\in\mathbb{N}_C}(\gamma_{ic} - \sum_{\ell\in\mathbb{N}_m} f_{\ell c}(x_i^\ell) - b_c)^2 + \frac{1}{2}\sum_{\ell\in\mathbb{N}_m}(\sum_{c\in\mathbb{N}_C}||f_{\ell c}||^2_{\mathcal{H}_\ell})^{\frac{1}{2}} \\ \text{s.t.} \quad & f_{\ell c}\in\mathcal{H}_\ell,\ \forall c\in\mathbb{N}_C, \ell\in\mathbb{N}_m \end{aligned}$$

Using the above argument for binary classification it is easy to check its dual problem is as follows

$$\min_{\lambda\in\Delta}\max_{\alpha} \sum_{i,c}\alpha_{ic}\overline{\gamma}_{ic} - \frac{1}{4}\sum_{i,c}\alpha_{ic}^2 - \frac{1}{4\gamma}\sum_{i,j,c}\alpha_{ic}\alpha_{jc}\tilde{K}_\lambda(x_i, x_j) \tag{24}$$

where $\overline{\gamma}_{ic} = \gamma_{ic} - \sum_{j\in\mathbb{N}_n}\gamma_{jc}$. Let n_c denote the number of samples in class c. If we redefine the class indicator matrix $\mathbf{Y}$, for any $i \in \mathbb{N}_n$ and $c \in \mathbb{N}_C$ by $\gamma_{ic} = \frac{1}{2}\sqrt{\frac{n}{n_c}}$ if $\gamma_i = c$, otherwise $-\frac{1}{2}\sqrt{\frac{n}{n_c}}$, then the class indicator matrix $\overline{\mathbf{Y}}$ reduces to the matrix H defined in [18] for multi-class classification, i.e.

$$\overline{\gamma}_{ic} = h_i(j) = \begin{cases} \sqrt{\frac{n}{n_c}} - \sqrt{\frac{n_c}{n}}, & \text{if } \gamma_i = c \\ -\sqrt{\frac{n_c}{n}}, & \text{otherwise.} \end{cases}$$

Now we can see that the dual problem of algorithm (24) is exactly the same as the formulation (see equation (36) in [18]) of RKDA kernel learning.

Acknowledgements

We would like to thank the referees for their constructive comments and suggestions which greatly improve the paper. We also thank Prof. Mark Girolami, Dr. Theodoros Damoulas and Dr. Simon Rogers for stimulating discussions. This work is supported by EPSRC grant EP/E027296/1.

References

1. Baker D, Sali A: **Protein structure prediction and structural genomics.** *Science* **294:**93-96.
2. Jones DT, *et al.*: **A new approach to protein fold recognition.** *Nature* 1992, **358:**86-89.
3. Ding C, Dubchak I: **Multi-class protein fold recognition using support vector machines and neural networks.** *Bioinformatics* 2001, **17:**349-358.

4. Shen HB, Chou KC: **Ensemble classifier for protein fold pattern recognition.** *Bioinformatics* 2006, **22:**1717-1722.
5. Andreeva A, *et al.*: **SCOP database in 2004: refinements integrate strucuture and sequence family data.** *Nucleic Acids Res* 2004, **32:**226-229.
6. Lo Conte L, *et al.*: **SCOP: a structural classification of protein database.** *Nucleic Acids Res* 2000, **28:**257-259.
7. Chou K, Zhang C: **Prediction of protein structural classes.** *Crit Revi Biochem Mol Biol* 1995, **30:**275-349.
8. Dubchak I, *et al.*: **Prediction of protein folding class using global description of amino acid sequence.** *Proc Natl Acad Sci* 1995, **92:**8700-8704.
9. Schölkopf B, Smola AJ: *Learning with Kernels* The MIT Press, Cambridge, MA, USA; 2002.
10. Shawe-Taylor J, Cristianini N: *Kernel Methods for Pattern Analysis* Cambridge university press; 2004.
11. Lanckriet GRG, Cristianini N, Bartlett PL, Ghaoui LE, Jordan ME: **Learning the kernel matrix with semidefinite programming.** *J of Machine Learning Research* 2004, **5:**27-72.
12. Bach F, Lanckriet GRG, Jordan MI: **Multiple kernel learning, conic duality and the SMO algorithm.** *Proceedings of the Twenty-first International Conference on Machine Learning (ICML)* 2004.
13. Rakotomamonjy A, Bach F, Canu S, Grandvalet Y: **More efficiency in multiple kernel learning.** *Proceedings of the 24th International Con- ference on Machine Learning (ICML)* 2007.
14. Sonnenburg S, Rätsch G, Schäfer C, Schölkopf B: **Large scale multiple kernel learning.** *J of Machine Learning Research* 2006, **7:**1531-1565.
15. Bach F: **Consistency of the group Lasso and multiple kernel learning.** *J of Machine Learning Research* 2008, **9:**1179-1225.
16. Ying Y, Zhou DX: **Learnability of Gaussians with flexible variances.** *J of Machine Learning Research* 2007, **8:**249-276.
17. Lin Y, Zhang H: **Component selection and smoothing in multivariate nonparametric regression.** *Annals of Statistics* 2006, **34:**2272-2297.
18. Ye J, Ji S, Chen J: **Multi-class discriminant kernel learning via convex programming.** *J of Machine Learning Research* 2008, **9:**719-758.
19. Ye J, *et al.*: **Heterogeneous data fusion for Alzheimer's disease study.** *The Fourteenth ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (SIGKDD)* 2008.
20. Ji S, Sun L, Jin R, Ye J: **Multi-label multiple kernel learning.** *The Twenty-Second Annual Conference on Neural Information Processing Systems (NIPS)* 2008.
21. Damoulas T, Girolami M: **Probabilistic multi-class multi-kernel learning: On protein fold recognition and remote homology detection.** *Bioinformatics* 2008, **24(10):**1264-1270.
22. Girolami M, Zhong M: **Data integration for classification problems employing gaussian process priors.** In *Advances in Neural Information Processing Systems* Cambridge, MA: MIT Press; 2007.
23. Lanckriet GRG, De Bie T, Cristianini N, Jordan MI, Noble WS: **A statistical framework for genomic data fusion.** *Bioinformatics* 2004, **20(16):**2626-2635.
24. Lawrence ND, Sanguinetti G: **Matching kernels through Kullback-Leibler divergence minimization.** In *Technical Report CS-04–12* Department of Computer Science, University of Sheffield; 2004.
25. Davis JV, Kulis B, Jain P, Sra S, Dhillon IS: **Information-theoretic metric learning.** *Proceedings of the 24th International Conference on Machine Learning* 2007.
26. Sun L, Ji S, Ye J: **Adaptive diffusion kernel learning from biological networks for protein function prediction.** *BMC Bioinformatics* 2008, **9:**162.
27. Tsuda K, Akaho S, Asai K: **The em algorithm for kernel matrix completion with auxiliary data.** *Journal of Machine Learning Research* 2003, **4:**67-81.
28. Kato T, Tsuda K, Asai K: **Selective integration of multiple biological data for supervised network inference.** *Bioinformatics* 2005, **21:**2488-2495.
29. Aronszajn N: **Theory of reproducing kernels.** *Trans Amer Math Soc* 1950, **68:**337-404.
30. Cristianini N, Shawe-Taylor J, Elisseeff A: **On kernel-target alignment.** In *Advances in Neural Information Processing Systems 14* Cambridge, MA: MIT Press; 2002.
31. Micchelli CA, Pontil M: **Learning the kernel function via regularization.** *J of Machine Learning Research* 2005, **6:**1099-1125.
32. Hastie T, Tibshirani R, Friedman J: *Elements of Statistical Learning: Data Mining, Inference, and Prediction* New York, Springer; 2001.
33. Vandenberghe L, Boyd S, Wu S: **Determinant maximization with linear matrix inequality constraints.** *SIAM J Matrix Analysis and Applications* 1998, **19:**499-533.
34. Amari S: **Information geometry of the EM and em algorithms for neural networks.** *Neural Networks* 1995, **8:**1379-1408.
35. Rasmussen CE, Williams C: *Gaussian Processes for Machine Learning* the MIT Press; 2006.
36. Tao PD, An LTH: **A D.C. optimization algorithm for solving the trust region subproblem.** *SIAM J Optim* 1998, **8:**476-505.
37. Yuille AL, Rangarajan A: **The concave convex procedure.** *Neural Computation* 2003, **15:**915-936.
38. Borwein JM, Lewis AS: *Convex Analysis and Nonlinear Optimization: Theory and Examples* CMS Books in Mathematics, Springer-Verlag, New York; 2000.
39. Nesterov Y: *Introductory Lectures on Convex Optimization: A Basic Course* Springer; 2003.
40. Hettich R, Kortanek KO: **Semi-infinite programming: theory, methods, and applications.** *SIAM Review* 1993, **3:**380-429.
41. Bleakley K, Biau G, Vert JP: **Supervised reconstruction of biological networks with local models.** *Bioinformatics* 2007, **23:**57-65.
42. Yamanishi Y, Vert J-P, Kanehisa M: **Protein network inference from multiple genomic data: a supervised approach.** *Bioinformatics* 2004, **20:**363-370.
43. Kondor RI, Lafferty JD: **Diffusion kernels on graphs and other discrete structures.** *Proceedings of the Nineteenth International Conference on Machine Learning* 2002.
44. **The SimpleMKL Toolbox** [http://asi.insa-rouen.fr/enseignants/~arakotom/code/mklindex.html]
45. Liao L, Noble WS: **Combining pairwise sequence similarity and support vector machine for detecting remote protein evolutionary and structural relationships.** *J Comput Biol* 2003, **6:**857-868.
46. Mewes HW, *et al.*: **MIPS: a database for genomes and protein sequences.** *Nucleic Acids Res* 2000, **28:**37-40.