REVIEW **Communicated by Ole Winther**

Online Learning with (Multiple) Kernels: A Review

Tom Diethe
t.diethe@cs.ucl.ac.uk
Mark Girolami
girolami@stats.ucl.ac.uk
Department of Statistical Science, Centre for Computational Statistics and Machine Learning, University College London, WC1E 6BT, U.K.

This review examines kernel methods for online learning, in particular, multiclass classification. We examine margin-based approaches, stemming from Rosenblatt's original perceptron algorithm, as well as nonparametric probabilistic approaches that are based on the popular gaussian process framework. We also examine approaches to online learning that use combinations of kernels—online multiple kernel learning. We present empirical validation of a wide range of methods on a protein fold recognition data set, where different biological feature types are available, and two object recognition data sets, Caltech101 and Caltech256, where multiple feature spaces are available in terms of different image feature extraction methods.

1 Introduction

In its canonical form, the term *online learning* refers to the paradigm where the "learner" (i.e. algorithm) receives data instances and labels in a sequential fashion (Freund & Schapire, 1997). The goal of the learner is to predict the label $y_t \in \mathcal{Y}$ of the instance $\mathbf{x}_t \in \mathcal{X}$ at time t, after which the true label is revealed. This differs from batch learning, where a (training) data set S consisting of m pairs of examples $\{\mathbf{x}_i, y_i\}$ is given, and incremental learning, where multiple batches $S_t, t = 1, .., T$ are given. Typically in batch and incremental learning, it is assumed that the data are generated independent and identically *(iid)* from an underlying data-generating distribution $\mathcal{D} = \{\mathcal{X} \times \mathcal{Y}\}$. In this scenario, the goal of the learner in incremental learning is to build a model that closely approximates the model that would be learned if all the batches of the data were given at once, with computational advantages of efficient incremental updates and processing only on the batch at any one time. Of course, if the size of each batch is 1, this would reduce to the online learning setting, with the assumption that the data are generated i.i.d. However, often online learning algorithms (and the theory associated with them) does not assume that the data are generated i.i.d.

(i.e., the underlying data distribution may be shifting or switching), so this reduction of incremental to online learning is to a restricted case only.

The perceptron algorithm (Rosenblatt, 1958) is an online algorithm that iteratively builds a hyperplane to separate data into two classes. It proceeds by adjusting the hyperplane whenever a mistake is made and has been shown to have favorable convergence properties if the data are linearly separable (Novikoff, 1962). In the case where the data are noisy, the perceptron may never converge, but this can be alleviated by fixing the model after a period of time or through the use of a regularization term in the cost function. If the goal is to find the hyperplane that best separates the data, it stands to reason that casting this goal as an optimization problem is a sensible thing to do. The support vector machine (SVM) (Vapnik, 1995) does exactly this: it seeks to find the hyperplane that maximizes the margin between two classes of data. It has since been shown that a minor modification of the perceptron (the margin perceptron; see Duda and Hart, 1973) would be equivalent to an unregularized SVM, with the only remaining difference appearing to be the training method: stochastic gradient descent (SGD) for the perceptron and quadratic programming for the SVM (Collobert & Bengio, 2004). In fact, since the SVM can be solved using SGD as well, this link is probably even deeper (Bordes, Ertekin, Weston, & Bottou, 2005). Since the perceptron has long stood as a benchmark in online learning, it should come as no surprise that the SVM stands as the benchmark in the batch classification setting, certainly in terms of kernel methods (see section 1.2).

These distinctions will become clearer when we discuss specific algorithms, as some algorithms are designed from the batch or incremental perspective, sometimes even requiring that several (pseudo-online) passes over the (batch) data set are required, and others are "true" online. We will refer to the former as "online batch" and the latter as "true online." This often leads to confusion as often both are referred to as online learning. In fact, it is also possible to distinguish two separate cases in the online-batch setting (approaches based on Monte Carlo sampling techniques): only one pass over the training data, which is known as stochastic approximation (SA), and multiple passes over the training data, which is known as sample average approximation (SAA) (Nemirovski, Juditsky, Lan, & Shapiro, 2009). Neither is related to the true-online setting, but when the data are i.i.d., they can be used to obtain classifiers with generalization error bounds. In SA, the aim is to approximate a problem of the form

$$\min_{\mathbf{w}} \mathcal{R}(\mathbf{w}) + \mathbb{E}[\mathcal{L}(\mathbf{w}, \mathcal{X}, \mathcal{Y})], \tag{1.1}$$

where $\mathcal{R}$ is the regularization function, $\mathbf{w}$ is the solution vector, and $\mathcal{L}$ is the loss function. In SAA, the above problem is approximated with

$$\min_{\mathbf{w}} \mathcal{R}(\mathbf{w}) + \frac{1}{T} \sum_{i=1}^{T} \mathcal{L}(\mathbf{w}_i, \mathbf{x}_i, y_i), \tag{1.2}$$

which can be minimized, such as with stochastic methods. The solutions of the algorithms will differ greatly, even if their instantiations are similar (Nemirovski et al., 2009). A full analysis of these methods is outside the scope of this review.

1.1 Online-to-Batch Conversions. Note that in general, it is always possible to transform an online algorithm into a batch algorithm. From a theoretical perspective, if it is known that the samples are i.i.d., it is possible to transform a regret bound for an online algorithm into a convergence rate or generalization bound for the equivalent batch problem. However the opposite is not true; algorithms that are designed specifically to work with i.i.d. samples with a proven generalization error bound in principle cannot be used in the pure online setting and do not have a regret bound guarantee. Examples of this are the algorthms based on stochastic gradient descent (SGD) analyzed by Zhang (2004). These have finite-sample convergence rate bounds but cannot be used in the online setting without projections (see Zinkevich, 2003).

When an online algorithm is converted to a batch algorithm, the resulting batch algorithm often closely resembles some form of SGD algorithm. This online-to-batch conversion can be carried out in different ways. It may seem like the simplest method is to take the last solution. However, if the algorithm has not yet converged, the last solution may be arbitrarily bad. For convex loss functions, the simplest approach that mitigates this problem is to take the average of all of the solutions generated at each time step, (see Cesa-Bianchi, Conconi, & Gentile, 2004, for details). For nonconvex losses, a complex procedure is also described in Cesa-Bianchi et al. (2004), which enables one to obtain generalization error bounds from regret bounds, such as for the 0–1 loss function of the perceptron algorithm.

We will attempt to disambiguate from a wide set of algorithms that have been proposed, and for a representative selection of these, we evaluate their performance in terms of their true online performance, as well as their online-batch performance, that is, how well the model trained on a data set in an online fashion then performs on a separate test set. We will not consider the incremental setting in this study except where it is necessary for the algorithm to run in a reasonable time frame.

For the purposes of this study, for one set of experiments we use the online-to-batch conversion mechanism described in Cesa-Bianchi et al. (2004)—averaging the hyperplanes generated at each step in the learning phase.

1.2 Kernel Methods in Online Learning. We restrict this survey to so-called kernel methods (Shawe-Taylor & Cristianini, 2004), a more rigorous definition of which will be given in section 2. Briefly, a kernel function κ is a function that for all $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ satisfies $\kappa(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$, where $\phi : \mathcal{X} \mapsto \mathcal{H}$ is a mapping from $\mathcal{X}$ to an (inner product) Hilbert space $\mathcal{H}$. This allows inner products between nonlinear mappings (when $\phi(\cdot)$ is a nonlinear

function), as long as the inner product $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ can be evaluated efficiently. In many cases, this inner product or kernel function can be evaluated much more efficiently than the feature vector itself, which can even be infinite-dimensional in principle. For a given kernel function, the associated feature space is not necessarily unique. A commonly used kernel function for which this is the case is the radial basis function (RBF) kernel, defined as

$$\kappa_\gamma(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{\gamma}\right). \tag{1.3}$$

The RBF kernel projects the data into a (nonunique) infinite-dimensional Hilbert space. One drawback, as with most other kernel methods, is that storing large kernel matrices is computationally prohibitive: it requires $\mathcal{O}(m^2)$ data storage.[1] When kernel methods are used in online learning, this data storage requirement is dropped at the expense of extra kernel computations. An online algorithm will have to calculate the kernel function between the new data point and any points that are stored from previous rounds that may be used in calculating the decision function or updating of the statistical model at time t. The set of points stored by the algorithm will be called the active set and denoted by $\mathcal{A}$ henceforth. In many cases, it is more practical to store the original data points rather than the kernel evaluations, at the expense of the computational cost per iteration. The use of an active set means that the memory requirements at time t are only $\mathcal{O}(|\mathcal{A}_t|)$ (or $\mathcal{O}(|\mathcal{A}_t|^2)$ if the kernel evaluations are stored) where typically $|\mathcal{A}_t| \ll t$, and in many cases can be fixed a priori. We will see that how this active set is created and maintained, together with the statistical model being generated, are the two key features that differentiate between the algorithms we investigate here.

1.3 Online Multiple Kernel Learning. Online learning and kernel learning are two active research topics in machine learning. Although each has been studied extensively, only recently have efforts in addressing the intersecting research been made. Some early work that did not specifically refer to the multiple kernel learning (MKL) problem examined the situation where a set of "experts" is employed, and relative loss bounds were developed using weighted combinations of these experts (Herbster & Warmuth, 1998).[2] Of course, each expert could be the same algorithm (such as a kernel perceptron) using a different feature space, in which case this resembles an MKL approach (and as we will see particularly resembles the approach to

[1] For online learning, $m = T$. For batch learning, m is then the number of training points.

[2] See Gönen and Alpaydin (2011) for a recent review.

online MKL of Jin, Hoi, & Yang, 2010). However the goal of the framework was to model situations in which the data-generating distribution of the examples changes (or switches), and different experts are best for certain segments of the sequence of examples. The algorithm has been shown to perform well on data organized in temporal blocks (Herbster & Warmuth, 2001; Herbster, 2001). Here we restrict ourselves to algorithms that attempt to find a stable combination of the feature spaces. Two notable approaches to online multiple kernel learning (OMKL) have been made, both of which try to learn a kernel-based prediction function from a pool of predefined kernels (or kernel functions) in the online learning framework (Jin et al., 2010; Luo, Orabona, Fornoni, Caputo, & Cesa-Bianchi, 2010). The problem of OMKL is generally more challenging than typical online learning because both the kernel classifiers and the (linear) combination of these must be learned simultaneously. The specific approaches outlined by Jin et al. (2010) and Luo et al. (2010) are discussed in section 3.3.

1.4 Probabilistic Online Learning. Another approach to the online learning problem is the probabilistic or Bayesian approach. Traditional Bayesian inference requires the definition of prior and likelihood distributions, and then computation of the posterior conditioned on the data and the prior. In the online setting, the true posterior distribution is replaced with a simpler parametric distribution, and following on from this, one can define an online algorithm by alternating between updates of the approximate posterior when a new example arrives, and an optimal projection into the parametric family (Opper, 1998). Predictions are made by averaging over the approximate posterior. It was also suggested that minimizing the difference between the batch and the approximate posterior optimizes the performance of the Bayes online algorithm (Winther & Solla, 1998). These methods were demonstrated in neural network learning (Opper, 1998) and linear perceptron learning with binary or continuous weight priors (Winther & Solla, 1998).

The SVM is perhaps the state of the art in kernel-based classification of batch data sets. In the (nonparametric) Bayesian framework, gaussian process classifiers (GPC) (Rasmussen & Williams, 2005) produce solutions that often generalize as well as or better than the SVM solutions (Kuss, Rasmussen, & Herbrich, 2005), while providing additional information, namely, confidence of predictions, in return for added computational complexity. The gaussian process is used as a prior probability distribution over functions, where a multivariate gaussian whose covariance matrix parameter is the Gram matrix of the set of m data points with some desired kernel function. There have been efforts to create sparse efficient versions of the GPC, such as the informative vector machine (IVM) (Lawrence, Seeger, & Herbrich, 2002), where a reduced set of points (support vectors) is used for final classification. This leads to the possibility that the estimation of the model parameters can be done in an online fashion, with the support

vectors also being chosen online (again forming an active set again). There have been several attempts to do this, notably sparse online gaussian processes (SOGP) (Csató & Opper, 2002), the virtual vector machine (VVM) (Minka, Xiang, & Qi, 2009), and predictive active set selection methods for gaussian Processes (PASS-GP) (Henao & Winther, 2010). These methods are discussed further in section 3.5.

1.5 Outline. The rest of the review is set out as follows. In section 2, we formally introduce the notation that will be used throughout the review. In section 3, we describe each of the algorithms that we will be examining. These will be presented in (roughly) chronological order and divided into several subsections: those that come from a decision-theoretic perspective (section 3.1), perceptron based (section 3.2), margin-based perspective (section 3.2.4), and those that come from a probabilistic (Bayesian) perspective (section 3.5). In section 4 we provide extensive empirical testing of the proposed methods, and in section 5 we conclude with a discussion.

2 Preliminaries

In the binary online classification framework with a single feature space, we assume that at time t, we are given an example $\mathbf{x}_t \in \mathbb{R}^n$, and the goal is to make a prediction $\hat{y}_t \in \{-1, 1\}$, after which the true label $y_t \in \{-1, 1\}$ is revealed.

Define $\mathbb{I}(z)$ as the indicator function that returns 1 if $z > 0$ and 0 otherwise, $\text{sgn}(z)$ as the function that returns -1 if $z < 0$ and 1 otherwise, $\mathbf{1}$ as a vector of all ones, $\mathbf{I} \in \mathbb{R}^{m \times m}$ as the m-dimensional identity matrix, and $\mathbf{z}_{\mathcal{I}}$ as the vector indexed by the index set $\mathcal{I}$. $\mathbf{z}'$ and $\mathbf{A}'$ denote the transposes of the vector $\mathbf{z}$ and matrix $\mathbf{A}$, respectively.

We aim to learn a linear function $f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$, which due to the representer theorem can be rewritten in the form $f(\mathbf{x}) = \sum_i \alpha_i \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle = \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x})$, where κ is the kernel function described in the following definition:

Definition 1. *A kernel is a function κ that for all $\boldsymbol{x}, \boldsymbol{z} \in \mathcal{X}$ satisfies*

$$\kappa(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle,$$

where ϕ is a mapping from $\mathcal{X}$ to an (inner product) Hilbert space $\mathcal{H}$,

$$\phi : \mathcal{X} \mapsto \mathcal{H}.$$

For a kernel function defined in this way, its reproducing property can be stated as $f(\mathbf{x}_i) = \langle f, \kappa(\mathbf{x}_i, \cdot) \rangle_{\mathcal{H}}$ for the reproducing kernel κ for every function $f(\mathbf{x}_i)$ belonging to $\mathcal{H}$.

In the multiple kernel learning setting, we are given a finite number of kernel functions (a kernel family) $\kappa_\beta = \{\kappa_1, \ldots, \kappa_k\}$, which can be combined using a convex combination as follows:[3]

$$\kappa_\beta(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{k} \beta_j \kappa_j(\mathbf{x}, \mathbf{z}), \quad \text{with } \beta_j \geq 0, \ \sum_{j=1}^{k} \beta_j = 1, \tag{2.1}$$

where $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$.

The explicit feature mapping induced by each kernel function is defined as $\phi_j : \mathbf{x} \mapsto \phi_j(\mathbf{x}) \in \mathcal{H}_j$, $j = 1, \ldots, k$, where $\mathcal{H}_j$ is the Hilbert space of the jth feature space. We will denote the set of weight vectors for each feature map as $\bar{\mathbf{w}} = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ and the kernel coefficients as $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_k)'$.

2.1 Batch Multiple Kernel Learning. The primal optimization of multiple kernel learning (MKL) (which we refer to as "ℓ_1 MKL"), as defined in Rakotomamonjy, Bach, Canu, and Grandvalet (2008), can be written as

$$\begin{aligned}
\min \ & \frac{1}{2} \sum_{j=1}^{k} \left(\frac{1}{\beta_j} \|\mathbf{w}_j\|_2 \right) + C \sum_{i=1}^{m} \xi_i \\
\text{w.r.t.} \ & \bar{\mathbf{w}} \in \mathbb{R}^{n \times k}, \boldsymbol{\beta} \in \mathbb{R}^k, \boldsymbol{\xi} \in \mathbb{R}^m, b \in \mathbb{R}, \\
\text{s.t.} \ & y_i \left(\sum_{j=1}^{k} \langle \mathbf{w}_j, \phi_j(\mathbf{x}_i) \rangle + b \right) \geq \mathbf{1} - \xi_i, \quad i = 1, \ldots, m, \\
& \xi_i \geq 0, \quad i = 1, \ldots, m, \\
& \sum_{j=1}^{k} \beta_j = 1, \quad \beta_j \geq 0, \quad j = 1, \ldots, k,
\end{aligned} \tag{2.2}$$

with the following decision rule,

$$\begin{aligned}
f(\mathbf{x}) &= \operatorname{sgn}\left(\sum_{j=1}^{k} \beta_j^* \langle \mathbf{w}_j^* \phi_j(\mathbf{x}) \rangle + b^* \right) \\
&= \operatorname{sgn}\left(\sum_{j=1}^{k} \beta_j^* \sum_{i=1}^{m} \alpha_i^* \kappa_j(\mathbf{x}, \mathbf{x}_i) + b^* \right) \\
&= \operatorname{sgn}\left(\sum_{i=1}^{m} \alpha_i^* \kappa_\beta(\mathbf{x}, \mathbf{x}_i) + b^* \right),
\end{aligned} \tag{2.3}$$

[3] There exist other combinations such as linear combinations (as convex but without the nonnegativity constraint), multiplicative combinations, and nonlinear combinations. Here we restrict ourselves to convex combinations only.

where κ_β is defined as in equation 2.1. The ℓ_1 norm penalty on the kernel coefficients leads to a sparse subselection of kernels in the kernel combination. This is often seen as desirable for computational and interpretability reasons, but a sparse subselection of kernels may not always be desirable. The formulation above can be generalized to the case of generic ℓ_p norms, $p \geq 1$ (see, for example, Kloft, Brefeld, Sonnenburg, & Zien, 2011).

It will also be useful to define the $\ell_{2,p}$-group norm $\|\bar{\mathbf{w}}\|_{2,p}$ of $\bar{\mathbf{w}}$ as

$$\|\bar{\mathbf{w}}\|_{2,p} \doteq \|(\|\mathbf{w}_1\|_2, \|\mathbf{w}_2\|_2, \ldots, \|\mathbf{w}_k\|_2)\|_p$$

2.2 Gaussian Processes for (Online) Classification. Here we briefly review the application of gaussian processes for classification and how they are used in an online setting.

A gaussian process is defined as a gaussian distribution over latent functions. Given the consistency property of gaussian distributions, whereby marginals are also gaussian, pointwise evaluations at the data points $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_T)]^T$ are jointly gaussian. By specifying a covariance function $\mathbf{C}(\mathbf{x}, \mathbf{x}')$ and a mean function $h(\mathbf{x})$, we see that $\mathbf{f} \sim \mathcal{N}(\mathbf{h}, \mathbf{K})$ where $\mathbf{h} = [h(\mathbf{x}_1, \ldots, \mathbf{x}_T)]$ and $\mathbf{K}$ is the covariance matrix, which is equivalent to the Gram matrix in kernel methods. Since the data points appear in the expression only through the covariance matrix, and hence through inner products, any nonlinear mapping that produces valid covariances (kernels) can be used, as before. For classification, it is assumed that the labels are observed independently, and a probit likelihood function $g(y_t|f(\mathbf{x}_t)) = Q(f_t y_t)$ where $Q(\cdot)$ is the gaussian cumulative density function.[4] The (intractable) posterior is then

$$p(\mathbf{f}|\mathbf{X}, \mathbf{y}) = \frac{1}{Z} p(\mathbf{f}|\mathbf{X}) \prod_{t=1}^{T} g(y_t|f(\mathbf{x}_t)),$$

where the normalizing constant $Z = p(\mathbf{y}|\mathbf{X})$ is the marginal likelihood. Currently the most accurate deterministic approximation to this is through the use of expectation propagation (EP) (Rasmussen & Williams, 2005). In EP, the likelihood is approximated by an unnormalized gaussian to give

$$\begin{aligned} p(\mathbf{f}|\mathbf{X}, \mathbf{y}) &= \frac{1}{Z_{EP}} p(\mathbf{f}|\mathbf{X}) \prod_{t=1}^{T} \frac{1}{Z_t} \tilde{g}(y_t|f(\mathbf{x}_t)), \\ &= \frac{1}{Z_{EP}} p(\mathbf{f}|\mathbf{X}) \mathcal{N}(\mathbf{f}|\tilde{\mathbf{h}}, \tilde{\mathbf{C}}), \\ &= \mathcal{N}(\mathbf{f}|\mathbf{h}, \mathbf{C}), \end{aligned}$$

[4] A logit likelihood function can also be used.

where Z_{EP} and Z_t are normalization coefficients, $g(y_t|\tilde{f}(\mathbf{x}_t))$ and $\mathcal{N}(\mathbf{f}|\tilde{\mathbf{h}}, \tilde{\mathbf{C}})$ are the gaussian approximations to $g(y_t|f(\mathbf{x}_t))$ at each site $\mathbf{x}_t$. In order to obtain the full (approximate) posterior $q(\mathbf{f}|\mathbf{X}, \mathbf{y})$, one would start by using the prior, $q(\mathbf{f}|\mathbf{X}, \mathbf{y}) = p(\mathbf{f}|\mathbf{X})$, and update each site approximation $\tilde{g}(\cdot)$ sequentially. In order to do this, the so-called cavity distribution $\tilde{q}(\mathbf{f}|\mathbf{X}, \mathbf{y}_t)$ is used—the current posterior with the point $\mathbf{x}_t$ removed. In the "online" methods of Csató and Opper (2002), Lawrence et al. (2002), and Henao and Winther (2010), the full posterior approximation is achieved only after (at least one) full pass through the data set. If time is unbounded, then this method is not possible. The alternative is to process the data in minibatches, where the EP step is run each time a batch of points is included, with the predictions of those points being made on the basis of the model at the previous time step. Of course, if the batch size is set to 1, this would be a true online setting, but the EP updates are generally too expensive to compute at that frequency. There is a further problem: unless some kind of active set is used, all of the data points up to time t would be required, which would lead to an unbounded increase in computational complexity. Csató and Opper (2002), Lawrence et al. (2002), and Henao and Winther (2010) all use an "active set"; they store a restricted set of data points used for EP updates. The methods are differentiated by how examples are included and excluded from the active set, which we discuss in section 3.5. Another approach, known as the virtual vector machine Minka et al. (2009), which we also discuss, is to use "virtual vectors"—vectors that are created by merging existing data points—as a proxy for the full data set.

2.3 Multiclass Classification. Most linear classifiers, whether online or offline, can be naturally generalized to multiclass classification. Here, the input $\mathbf{x}$ and the output y are drawn from the label set $\mathcal{Y}$. The classification function (or latent function in the GP setting) $f(\mathbf{x}, y)$ maps each possible input-output pair to a finite-dimensional real-valued feature vector. The resulting score is used to choose among many possible outputs:

$$\hat{y}_t = \arg\max_{y \in \mathcal{Y}} f(\mathbf{x}_t, y).$$

For example, in the case of the perceptron algorithm, the function f would simply be $f(\mathbf{x}_t, y = y_j) = \mathbf{w}_j' \mathbf{x}_j$. The notion of the kernel-defined feature space can be extended to be defined jointly on the space $\mathcal{X} \times \mathcal{Y}$, in which case $f(\mathbf{x}_t, y = y_j) = \mathbf{w}_j' \phi_j(\mathbf{x}_j, y_j)$ (see Fink & Singer, 2006). In the multiclass setup, with multiple feature spaces, for each class y_j, $j = 1, \ldots, C$, we define

$$\phi_j(\mathbf{x}_j, y_j) = (\mathbf{0}, \ldots, \mathbf{0}, \underbrace{\psi_j(\mathbf{x}_j)}_{y_j}, \mathbf{0}, \ldots, \mathbf{0}),$$

where $\psi_j(\cdot)$ is a label-independent feature map. Similarly, $\mathbf{w}$ consists of C blocks, and hence by construction $\mathbf{w}'\phi_j(\mathbf{x}_j, y_j) = \mathbf{w}_j'\psi_j(\mathbf{x}_j)$.

3 Online Learning with Kernels

In this section, we describe a range of the different approaches to online learning that have been proposed, with particular attention to those that have been extended using kernel methods.

3.1 Decision-Theoretic Approaches. The Hedge algorithm (Freund & Schapire, 1997; Vovk, 1998) is an algorithm for prediction with expert advice, which comes from a decision-theoretic generalization of online learning. Freund and Schapire (1997, theorem 2) prove that for any sequence of outcomes,

$$L_T \leq \frac{\ln 1/\beta}{1-\beta} L_T(k) + \frac{\ln K}{1-\beta},$$

for all T and k, where $\beta \in [0, 1]$ is a parameter, L_T is the loss suffered by the algorithm over the first T trials and $L_T(k) = \sum_{t=1}^{T} \omega_k^t$ is the loss (or expected loss) suffered by the kth expert over the first T trials. The inequality above was improved by Vovk (1998) using the strong aggregating algorithm,

$$L_T \leq c(\beta) L_T(k) + \frac{c(\beta)}{\ln(1/\beta)} \ln K,$$

where $c(\beta) = (\ln \frac{1}{\beta})/(K \ln \frac{K}{K+\beta-1})$. The same method can be applied if the experts and the algorithm provide the probability distribution over the outcome space and they suffer the expected loss of a decision randomly selected according to this distribution. In this case, the algorithm predicts simply the weighted average of the experts predictions. The theoretical bound for the loss of the Hedge algorithm is

$$L_T \leq L_T(k) + \sqrt{2L \ln K} + \ln K.$$

The constant L is a prior upper bound on the loss of the best strategy and in the worst case is $T\widetilde{L}$, where $\widetilde{L}$ is the bound for the loss used. The weights are updated by the rule $w_k := w_k \beta^{\lambda_k}$, where $\beta = \frac{1}{1+\sqrt{2 \ln K/L}}$, and then they are normalized. The bounds for some particular loss functions, such as square loss function, can be easily derived from the bound described above using geometrical inequalities. This bound justifies the use of the Hedge algorithm in a setting where a mixture of experts is combined using a weighted average; this is in turn a justification for the approach to online MKL by

Jin et al. (2010) which uses the Hedge algorithm to combine multiple kernel perceptrons.

Under a framework of the "best expert" (Herbster & Warmuth, 1998), worst-case loss bounds for online algorithms were generalized to the case where the additional loss of the algorithm on the whole sequence of examples was bounded over the loss of the best expert (which cannot be known a priori). The sequence is partitioned into segments, and the goal is to bound the additional loss of the algorithm over the sum of the losses of the best experts of each segment. The idea behind this was to model situations in which the examples change and different experts are best for certain segments of the sequence of examples, which may happen if the underlying probability distribution is multimodal or there are switches between phases in the data (e.g., caused by underlying trends or transient states). In the single expert case, the additional loss is proportional to $\log(k)$, where k is the number of experts and the constant of proportionality depends on the loss function. When the number of segments is at most $s+1$ and the sequence of length ℓ, the additional loss of their algorithm can be bounded over the best partitioning by $\mathcal{O}(s\log(k) + s\log(\ell/s))$. The algorithms for tracking the best expert are simple adaptations of Vovk's original algorithm for the single best expert case. These algorithms keep one weight per expert and spend $\mathcal{O}(1)$ time per weight in each trial. The results were later extended to linear combinations (Herbster & Warmuth, 2001).

3.2 Perceptron and Variants. The perceptron was proposed as a biologically plausible model for learning from vectorial data, where a learning rule is constructed that is a linear combination of the examples (Rosenblatt, 1958; Block, 1962; Vapnik & Chervonenkis, 1964). Novikoff (1962) showed that the perceptron algorithm converges after a finite number of iterations if the data set is linearly separable. The idea of the proof is that the weight vector is always adjusted by a bounded amount in a direction that it has a negative dot product with, and thus can be bounded above by, $\mathcal{O}(\sqrt{\delta})$, where δ is the number of changes to the weight vector. Furthermore, it can also be bounded below by $\mathcal{O}(\delta)$ because if there exists an (unknown) satisfactory weight vector, then every change makes progress in this (unknown) direction by a positive amount that depends on only the input vector. This can be used to show that the number δ of updates to the weight vector is bounded by $\frac{R^2}{\gamma^2}$, where R is the radius of the ℓ_2-norm ball enclosing the input space and γ is the margin (the distance between the closest points from each class).

Note that the decision boundary of a perceptron is invariant with respect to scaling of the weight vector that is, a perceptron trained with initial weight vector $\mathbf{w}$ and learning rate τ, results in an identical classifier to a perceptron trained with initial weight vector $\frac{\mathbf{w}}{\tau}$, and learning rate 1. Thus, since the initial weights become irrelevant with an increasing number of

iterations, the learning rate does not matter in the case of the perceptron and is usually just set to one.

3.2.1 Kernel Perceptron. The perceptron algorithm was first extended for use in kernel-defined Hilbert spaces by Freund and Schapire (1999), resulting in the kernel perceptron (KP). In addition they combined the classical perceptron algorithm with Helmbold and Warmuth's (1995) leave-one-out method in order to perform the online-to-batch conversion.[5] The resulting algorithm is simpler to implement and much more efficient in terms of computation time than the SVM, while the performance is close to (but not as good as) the performance of maximal-margin classifiers on the same problem.

3.2.2 Budgeted Perceptron Variants. If the data are not linearly separable, the (kernel) perceptron algorithm will never stop updating, meaning that the size of the support set is unbounded. For most practical problems, this makes it impractical for most real-world problems. The first approach to bound the growth of the support set was the budget perceptron (Crammer, Kandola, & Singer, 2003), which removes seemingly redundant examples from the support set by examining the margin conditions of old examples. This was later improved (tigher budget perceptron) by Weston, Bordes, and Bottou (2005) to account for noisy data sets. However, neither of these algorithms has any performance guarantees in terms of bounded numbers of errors. The forgetron (Dekel, Shwartz, & Singer, 2008) was the first budget perceptron that had a formal performance guarantee. It enforces a strict bound on the size of the active set by removing vectors. First, the weights of every vector in the active set are reduced; then it discards the oldest vector in the active set, which has the smallest weight. This is done only when the size of the active set exceeds the budget, so applying this removal procedure on every error will ensure that the size of the active set will not exceed the budget. Due to the repeated weight reductions, the oldest vector will have small weight, so removing it will not change the decision function significantly. The factor by which the forgetron reduces weight is not constant and differs from error to error.

The randomized budget perceptron (RBP) of Cavallanti, Bianchi, and Gentile (2007) is considerably simpler than the forgetron: whenever it makes an error and the size of the active set exceeds the budget, a vector is chosen at random from its active set to be discarded. It was shown that both of these algorithms have bounds of similar order (Sutskever, 2009). Another algorithm, the projectron (Orabona, Keshet, & Caputo, 2009), has been proposed in which the instances are not discarded but projected onto the space spanned by the previous online hypothesis. The authors derive a relative

[5]Note that the KP can be used without the Helmbold and Warmuth method.

mistake bound and compare the algorithm both analytically and empirically to the forgetron, showing favorable results. Since the RBP is by far the simplest of the budgeted perceptron algorithms, we used this method in the experiments. We also ran preliminary experiments with the projectron, but when compared to the RBP, we found it to be much slower without discernable performance gains.

3.2.3 Relation Between the Perceptron and the Support Vector Machine. For linearly separable data, the perceptron algorithm can be modified such that it seeks to find the largest separating margin between the classes. The "perceptron of optimal stability" can be solved through iterative schemes, such as the AdaTron (Anlauf & Biehl, 1989), which exploits the fact that the corresponding quadratic optimization problem is convex. The perceptron of optimal stability is, together with the kernel trick, one of the conceptual foundations of the SVM. Later it was shown that the SVM solution could be approximated by the kernel Adatron with an exponentially fast rate of convergence to the optimal solution (Friess, Cristianini, & Campbell, 1998), which completes the link. Note, however, that the Adatron falls under the category of online-batch algorithms rather than being a true online algorithm.

3.2.4 Online Support Vector Machines. Following on from the AdaTron algorithm, several attempts have been made to adapt the SVM for online learning. The incremental and decremental SVM (Cauwenberghs & Poggio, 2000) is an online recursive algorithm for training SVMs using "adiabatic increments."[6] An adiabatic process in thermodynamics is one in which heat transfer is zero. Here it refers to the "equilibrium" of the training data—that KKT conditions are satisfied for the whole data set, which, in a number of analytical steps, ensure that the KKT conditions (Kuhn & Tucker, 1951) are satisfied on all previously seen training data. Interestingly, the incremental procedure is reversible, and decremental "unlearning" offers an efficient method to exactly evaluate leave-one-out generalization performance, as well as giving some intuition about the relationship between generalization and the geometry of the data. However, since the algorithm requires the storage of the inverse of the Jacobian (second derivatives of the objective function), which scales as $|\mathcal{A}|^2$, which is in turn unbounded, the algorithm is more suited as an online-batch approach than a true online algorithm.

Kivinen, Smola, and Williamson (2002) combine classical stochastic gradient descent within a feature space and some straightforward tricks to develop simple and computationally efficient algorithms for a wide range

[6] An adiabatic process in thermodynamics is one in which heat transfer is zero. Here it refers to the "equilibrium" of the training data—that KKT conditions are satisfied for the whole data set.

of problems such as classification, regression, and novelty detection. The authors specifically consider using online large margin classification algorithms in a setting where the target classifier may change over time, but they also show how to exploit the kernel trick in an online setting. They show that their naive online regularized-risk minimization algorithm (NORMA) has appealing worst-case loss bounds and converges to the minimizer of the regularized risk functional. A similar approach was taken to create the implicit online learning with kernels (ILK) and sparse ILK (SILK) algorithms (Cheng, Vishwanathan, Schuurmans, Wang, & Caelli, 2006). An implicit update technique was used that can be applied to a wide variety of convex loss functions. The authors prove loss bounds, analyze the convergence rate of both algorithms, and show that empirically the algorithms outpeform NORMA.

Later, a family of margin-based online learning algorithms was developed for various prediction tasks under the umbrella term of online SVM and passive agressive (Crammer, Dekel, Keshet, Shalev-Shwartz, & Singer, 2006). The update steps of the algorithms are based on analytical solutions to simple constrained optimization problems. This allowed them to prove worst-case loss bounds for the different algorithms and for the various decision problems based on a single lemma. The bounds on the cumulative loss of the algorithms are relative to the smallest loss that can be attained by any fixed hypothesis, as in the "best expert" framwork (Herbster & Warmuth, 1998), and as such are applicable to both realizable and unrealizable settings.

Bordes et al. (2005) present another online variant of the SVM, LASVM, an approximate SVM solver that uses online approximation and was shown empirically to reach accuracies similar to that of a real SVM after performing a single sequential pass through the training examples. The authors also show that additional benefits can be achieved using selective sampling techniques to choose which example should be considered next—so-called active learning, which was introduced in the context of the SVM by Tong, Koller, and Kaelbling (2001). The algorithm alternates between two steps: process, which decides whether a point should be added into the active set, and reprocess, which examines the active set to see if any points should be removed. For each data vector, these steps can be repeated several times. The algorithm is fast and performs well in the online-batch setting, but this time suffers from the need to choose both the C parameter (SVM regularization parameter) and an additional τ parameter (for active set inclusion) a priori, which limits its use in a true online setting.

Finally, Orabona, Castellini, Caputo, Jie, and Sandini (2009) present online independent support vector machines (OISVMs), which approximately converge to the standard SVM solution each time new observations are added. The approximation is controlled via a user-defined parameter, which again poses a problem in the online setting. The method employs a set of linearly independent observations and tries to project every new observation

onto the set obtained so far, dramatically reducing time and space requirements at the price of a negligible loss in accuracy. This is similar in flavor to the projectron algorithm (Orabona, Keshet, & Caputo, 2009).

3.3 Online Multiple Kernel Learning. Recently there have been attempts to extend the multiple kernel learning (MKL) (Bach, Lanckriet, & Jordan, 2004; Lanckriet, Cristianini, Bartlett, Ghaoui, & Jordan, 2004; Rakotomamonjy et al., 2008) framework to the online learning setting. An incremental multiple kernel learning (IMKL) approach was proposed for object recognition that initializes on a generic training database and then tunes itself to the classification task at hand (Kembhavi, Siddiquie, Miezianko, McCloskey, & Davis, 2009). Their system simultaneously updates the training data set as well as the weights used to combine multiple information sources. An online approach to MKL for structured prediction was outlined (Martins, Smith, Xing, Aguiar, & Figueiredo, 2010) using a new family of online proximal algorithms. These algorithms can be used for MKL as well as for group-lasso (Bach, 2008) and variants thereof, and the authors give regret, convergence, and generalization bounds for the proposed methods. However, in the empirical evaluations, they state that multiple passes are required, meaning that the algorithms are not well suited to the true online setting.

3.3.1 OM2 Algorithm. Luo et al. (2010) introduced a theoretically motivated and efficient online learning algorithm for the multiclass MKL problem called OM2. For this algorithm, they prove a theoretical bound on the number of multiclass mistakes made on any arbitrary data sequence. Moreover, they show empirically that its performance is on par with, or better than, standard batch MKL, such as SimpleMKL (Rakotomamonjy et al., 2008) algorithms.

Using the group norm notation described above, the MKL problem can be defined in generic form as

$$\min_{\bar{\mathbf{w}}} \frac{\lambda}{2} \|\bar{\mathbf{w}}\|_{2,1}^2 + \sum_{t=1}^{T} \mathcal{L}\left(\bar{\mathbf{w}}, \mathbf{x}_t, y_t\right),$$

which can be extended to the more general $\ell_{2,p}$ case as

$$\min_{\bar{\mathbf{w}}} \frac{\lambda}{2} \|\bar{\mathbf{w}}\|_{2,p}^2 + \sum_{t=1}^{T} \mathcal{L}\left(\bar{\mathbf{w}}, \mathbf{x}_t, y_t\right),$$

with $1 < p \leq 2$. This objective function is λ/q strongly convex, where q is the dual norm of p satisfying $1/p + 1/q = 1$. The authors use the "follow the regularized leader" framework (Shalev-Shwartz & Singer, 2007; Kakade,

Shalev-Shwartz, & Tewari, 2009), in which the loss function is replace by its subgradient, meaning that at each step, the following optimization problem must be solved,

$$\bar{\mathbf{w}}_t = \arg\min_{\bar{\mathbf{w}}} \quad \eta \bar{\mathbf{w}}' \sum_{i=1}^{t-1} \partial \mathcal{L}(\bar{\mathbf{w}}_i, \mathbf{x}_i, y_i) + \mathcal{R}(\bar{\mathbf{w}}),$$

where $\mathcal{R}(\cdot)$ is a regularization function and $\eta > 0$ is a parameter. The linearization of the loss function through the subgradient provides an efficient closed form for the updates and allows regret bounds to be proven. The solution of the above gives the following update,

$$\bar{\mathbf{w}}_t = \nabla \mathcal{R}^* \left(-\eta \sum_{i=1}^{t-1} \partial \mathcal{L} \left(\bar{\mathbf{w}}_i, \mathbf{x}_i, y_i \right) \right),$$

where $\mathcal{R}^*$ is the Fenchel conjugate of $\mathcal{R}$, defined as $\mathcal{R}^*(\mathbf{u}) = \sup_{\mathbf{v} \in S}(\mathbf{v}'\mathbf{u} - \mathcal{R}(\mathbf{v}))$. For the OM2 algorithm using the $\ell_{2,p}$-norm formulation, the regularizer is $\mathcal{R}(\bar{\mathbf{w}}) = \frac{q}{2} \|\bar{\mathbf{w}}\|_{2,p}^2$, and therefore

$$\mathcal{R}^*(\bar{\boldsymbol{\theta}}) = \frac{1}{2q} \|\bar{\boldsymbol{\theta}}\|_{2,q}^2,$$

$$\nabla \mathcal{R}^*(\bar{\boldsymbol{\theta}}) = \frac{1}{q} \left(\frac{\|\boldsymbol{\theta}_j\|_2}{\|\bar{\boldsymbol{\theta}}\|_{2,q}^2} \right)^{q-2} \boldsymbol{\theta}_j, \quad j = 1, \ldots, k,$$

which is then used to make updates whenever a mistake is made or when the multiclass loss is greater than 0. The authors further propose a variant of the "follow the regularized leader" framework in which the parameter $\eta = \eta_t$ is changed at each time step if a mistake is made (see algorithm 1 in Luo et al., 2010) and prove that the cumulative number of mistakes made on any sequence of T observations is roughly equal to the optimum value of the original MKL problem.

3.3.2 OMKL Algorithm. Recently an approach to online multiple kernel learning (OMKL) was proposed that aims to learn a kernel-based prediction function from a pool of predefined kernels in an online learning fashion (Jin et al., 2010). The authors consider two setups for OMKL, combining binary predictions or real-valued outputs from multiple kernel classifiers, and they propose both deterministic and stochastic approaches in the two setups for OMKL, leading to six algorithms. The deterministic approach updates all kernel classifiers for every misclassified example, while the stochastic approach randomly chooses a classifier for updating according to some

sampling strategies. The methods are derived from a single simple extension of the kernel perceptron (KP); the idea is to combine the outputs from KPs learned on each feature space separately using the Hedge algorithm. In effect, this makes it more of a classifier combination algorithm than a true MKL algorithm, but we will continue to call the algorithm OMKL in the rest of this review. Mistake bounds are derived for all the proposed OMKL algorithms, along with pseudo-code for the algorithms, but no experimental results are given, and in fact no implementation has been tested before now.[7]

We chose to focus on the first two algorithms, DA for OMKL-P (1) and DA for OMKL-P (2), the only difference between the two being that the second version allows for misclassifications (at the cost of an extra parameter), whereas the first one does not. However in practice, we found very little difference in performance between the two and therefore chose to use the first variant because it has only one parameter (the discount parameter β).

3.4 Boosting and Random Forest Approaches to Online Learning. Recently a set of methods has been proposed that extend the boosting (Freund & Schapire, 1997) and random forest (Breiman, 2001; Bosch, Zisserman, & Munoz, 2007a) methods to the online setting; it is called online multiclass LPBoost, online multiclass gradient boost, and online random forest (Saffari, Leistner, Santner, Godec, & Bischof, 2009; Saffari, Godec, Pock, Leistner, & Bischof, 2010). These were designed for object detection in images. These are not kernel methods per se, as the nonlinearity comes through the use of so-called weak learners. However the link between ℓ_1-norm MKL and LPBoost has been observed (Hussain & Shawe-Taylor, 2010). This perhaps provides a further avenue for research for the derivation of new multiclass MKL methods.

3.5 Probalistic Approaches to Online Learning. The first attempt to discuss online learning from the viewpoint of Bayesian statistical inference was that of Opper (1998) and Opper and Winther (1999). By replacing the true posterior distribution with a simpler parametric distribution, one can define an online algorithm by a repetition of two steps: an update of the approximate posterior, when a new example arrives, and an optimal projection into the parametric family. Choosing this family to be gaussian, Opper showed that the algorithm achieves asymptotic effciency.

The most interesting approaches to online learning in a probabilistic setting have been online and incremental approaches gaussian proscess (GP) regression and classification. One of the first attempts to create an online approximation of the GP model was the sparse online GP of Csató and Opper (2002). The authors developed an approach for sparse representations of GP

[7]Personal correspondence with the authors, March 16, 2011.

models in order to overcome their limitations caused by large data sets. The method is based on a combination of a Bayesian online algorithm together with a sequential construction of an active set that fully specifies the prediction of the GP model. By using an appealing parametrization and projection techniques that use the RKHS norm, recursions for the effective parameters and a sparse gaussian approximation of the posterior process are obtained. This allows both a propagation of predictions and Bayesian error measures. However, the repeated projections into low-dimensional subspaces can be computationally costly because they require matrix multiplications.

Around the same time, the informative vector machine (IVM) was proposed as a practical method for gaussian process regression and classification (Lawrence et al., 2002). The IVM produces a sparse approximation to a gaussian process by combining assumed density filtering (Opper, 1998) with a heuristic for choosing points based on minimizing posterior entropy. This is notionally simper than the approach of Csató and Opper (2002), while obtaining similar results.

A method for the sparse greedy approximation of GP regression was given by Seeger, Seeger, Williams, Lawrence, and Dp (2003), featuring a novel heuristic for very fast-forward selection. The advantage of this method is that it is essentially as fast as an equivalent one that selects the support patterns at random yet was shown to outperform random selection on difficult curve-fitting tasks. More important, it leads to a sufficiently stable approximation of the log marginal likelihood of the training data, which can be optimized to adjust a large number of hyperparameters automatically. It is, however, limited to the regression setting.

Minka et al. (2009) took a slightly different approach, the virtual vector machine (VVM), in which information contained in the preceding data stream is summarized by a gaussian distribution of the classification weights plus a constant number of "virtual" data points, which are designed to include nongaussian information about the classification weights and in theory allows a smooth trade-off between prediction accuracy and memory size. To maintain the constant number of virtual points, the virtual vector machine adds the current real data point into the virtual point set, merges the two most similar virtual points into a new virtual point, or deletes a virtual point that is far from the decision boundary. The information lost in this process is absorbed into the gaussian distribution. The authors suggest that the extra information provided by the virtual points leads to improved predictive accuracy over previous online classification algorithms.

Most recently, Henao and Winther (2010) proposed a new approximation method for GP learning for large data sets, known as PASS-GP, that combines inline active set selection with hyperparameter optimization. The predictive probability of the label is used for ranking the data points. They use the leave-one-out predictive probability (the so-called cavity distribution) available in GPs to make a common ranking for both active and

inactive points, allowing points to be removed again from the active set. This is important for keeping the complexity down and at the same time focusing on points close to the decision boundary. The authors demonstrate state-of-the-art results (e.g., 0.86% error on MNIST) with reasonable time complexity.

One approach that has tried to bridge the gap between probabilistic- and optimization-based methods for online learning is TONGA (topmoumoute online natural gradient algorithm; Roux, Manzagol, & Bengio, 2007). The approach is guided by the goal of obtaining an optimization method that is both fast and yields good generalization, and to this end they study the descent direction that maximally decreases the generalization error or the probability of not increasing generalization error. From both Bayesian and frequentist perspectives, this can yield the natural gradient direction; although this can be expensive to compute, an efficient, general, online approximation is possible. This may be a possible avenue forward for unifiying the two approaches.

Another general approach to the problem of extracting informative exemples from a data stream is stream greedy (Gomes & Krause, 2010). The authors state that this method includes diverse approaches such as exemplar-based clustering and nonparametric inference, such as GP regression, on massive data sets. They show that the common theme underlying these problems is the maximization of a submodular function that captures the informativeness of a set of examples over a data stream. The stream greedy algorithm is guaranteed to obtain a constant fraction of the value achieved by the optimal solution to these NP-hard optimization problems, and could therefore provide a way of improving the efficiency of probabilistic methods such as PASS-GP.

Table 1 gives a taxonomy of the methods discussed in (roughly) chronological order.

3.6 Selection of Kernel Hyperparameters in Online Learning. If linear kernels are being used, the kernel function κ is simply the inner product between data examples, so there is no problem using this form of kernel in online learning. However, when nonlinear mappings are used, such as those defined by the RBF kernel function κ_γ as defined in equation 1.3, usually one or more hyperparameters need to be chosen. In the batch learning setting, a heuristic method such as κ-fold cross-validation is performed on the training set, and then the best parameter over the k-folds is used to retrain the model before testing on a separate test set. With no such deliniation between training and testing phases, the choice of kernel hyperparameters is clearly a tricky problem in the online learning setting. Among the several possible approaches to this problem are these:

- **Pseudo-validation set.** The simplest method is to assume that the first m points form a validation set, in which case the models can

Table 1: Taxonomy of Online Learning Algorithms.

Algorithm	Authors	Year	Paradigm	Motivation	Nonlinearity	Multiple Sources	Multiclass	Active Set
Perceptron	Rosenblatt	1958	True online	Maximum margin	Kernels	No	Yes (with modifications)	Support vectors (unbounded)
Hedge	Freund and Schapire	1997	True online	Decision theory	No	No	No[a]	No
Adatron	Friess, Cristianini, and Campbell	1998	Online batch[b]	Maximum margin	Kernels	No	No	Support vectors
Assumed density filtering (ADF)	Opper and Winther	1999	True online	Bayesian	No	No	No	No
Incremental/ decremental SVM	Cauwenberghs and Poggio	2000	Online batch[b]	Maximum margin	Kernels	No	No	Support vectors
Best expert	Herbster and Warmuth	2001	True online	Decision theory	Weak learners	Yes	Yes (with modifications)	No
NORMA	Kivinen, Smola, and Williamson	2002	True online	Optimization	Kernels	No	No	Support vectors (unbounded)
Informative vector machine	Lawrence, Seeger, and Herbrich	2003	Online batch[d]	Bayesian	Kernels	No	No	Support vectors (fixed size)
Sparse GP regression	Seeger, Williams, and Lawrence	2003	Incremental	Bayesian (GP)	Kernels	No	No (regression)	Support vectors
Budget perceptron (BP)	Crammer, Kandola, and Singer	2003	True online	Maximum margin	Kernels	No	Yes (with modifications)	Support vectors (bounded)
LASVM	Bordes, Ertekin, Weston, and Bottou	2005	Online batch[a,c]	Maximum margin	Kernels	No	No	Support vectors
Tighter BP	Weston, Bordes, and Bottou	2005	True online	Maximum margin	Kernels	No	Yes (with modifications)	Support vectors (bounded)
SimpleSVM	Cheng, Vishwanathan, Schuurmans, Wang, and Caelli	2006	Online batch[c,d]	Maximum margin	Kernels	No	No	Support vectors
Passive aggressive (PA)	Crammer, Dekel, Keshet, Shalev-Schwartz, and Singer	2006	True online	Maximum margin	Kernels	No	Yes	Support vectors (bounded)
Implicit learning with kernels (ILK)	Cheng, Viswanathan, Schuurmans, Wang, and Caelli	2006	Incremental	Maximum margin	Kernels	No		Support vectors

Table 1: *Continued.*

Algorithm	Authors	Year	Paradigm	Motivation	Nonlinearity	Multiple Sources	Multiclass	Active Set
Online SVM and PA	Shalev-Schwartz, and Singer	2007	Online batch 3	Maximum margin	Kernels	No	Yes (with modifications)	Support vectors
Topmoumoute natural gradient	Roux, Manzagol, and Bengio	2007	Online batch[a,c]	Optimization	No	No	Yes (with modifications)	No
Sparse online GP	Csato and Opper	2009	True online	Bayesian (GP)	Kernels	No	Yes	Virtual vectors
Projectron	Orabona, Keshet, and Caputo	2009	True online	Maximum margin	Kernels	No	Yes (with modifications)	Virtual vectors
Incremental MKL	Kembhavi, Siddiquie, Miezianko, McCloskey, and Davis	2009	Incremental	Maximum margin	Kernels	Yes	Yes	Support vectors
Virtual vector machine	Minka, Xiang, and Qi	2009	True online	Bayesian	Random Fourier features	No	No	Virtual vectors
Online multiclass MKL (OM2)	Luo, Orabona, Fornoni, and Cesa-Bianchi	2010	True online	Maximum margin	Kernels	Yes	Yes	Support vectors (unbounded)
Online MKL (structured)	Martins, Smith, Xing, Aguiar, and Figueriedo	2010	Online Batch[b,c]	Maximum margin	Kernels	Yes	Yes	Support vectors (unbounded?)
Online MKL (perceptron + hedge)	Jin, Hoi, and Yang	2010	True online	Maximum margin	Kernels	Yes	Yes (with modifications)	No
Stream greedy	Gomes and Krause	2010	True online	Decision theory	Kernels	No	Yes (with modifications)	Support vectors (bounded)
Online multiclass LPBoost	Saffari, Godec, Pock, Leistner, and Bischof	2010	Incremental	Maximum margin	Weak learners	No	Yes	No
PASS-GP	Henao and Winther	2010	Incremental	Bayesian (GP)	Kernels	No	No	Support vectors (bounded)

[a]Order of examples matters.
[b]Function must be calculated using all training examples.
[c]Requires multiple passes through the data.
[d]Requires a random selection of points from the training set to be available.

be run using a range of parameters, and the model chosen would be the one with the highest cumulative accuracy after m steps. The problem with this method is that without knowing which model will be selected, the predictions for the first m points will have to be made by randomly choosing a classifier from the pool of classifiers, arbritarily selecting one of them, using a voting scheme, or some other classifier combination method.

- **Large sets of kernels.** For the MKL algorithms, one could define a range of kernel functions for each feature space and each parameter setting. The algorithm would then select from among these automatically to find the best hyperparameters. This would lead to a large number of possibly redundant kernels and, consequently, a large number of unnecessary kernel evaluations.
- **Hoeffding races.** Hoeffding races (Maron & Moore, 1994), or the more recent Bernstein races (Heidrich-Meisner & Igel, 2009) are a technique for finding a good model from a selection of models by quickly discsarding bad models and concentrating the computational resources on differentiating among the better ones. These methods provide a more principled approach than the pseudo-validation set and offer a promising avenue of research.
- **Sequential Monte Carlo.** A recent study in the field of reinforcement learning (RL) develops replacing-kernel RL (RKRL) (Reisinger et al., 2008), an online model selection method for gaussian process temporal difference (GPTD: a Bayesian RL model by Engel, Engel, Mannor, & Meir, 2005) using sequential Monte Carlo (SMC) methods Doucet, De Freitas, & Gordon, 2001). SMC is used to select good kernel hyperparameter settings by choosing models according to their relative predictive likelihood instead of the true model likelihood. As a result, RKRL devotes more time to evaluating hyperparameter settings that correspond to areas with high predictive likelihood (i.e., maximizes online reward). When GPTD is used, the current value function estimate is formed from the combination of the kernel parameterization determining the prior covariance function and the dictionary gathered incrementally from observing state transitions. Each sampling step increases information about the predictive likelihood in the sample (exploitation), while sampling from the transition kernel reduces such information (exploration). This approach is certainly promising for the gaussian process–based approaches.
- **Nonparametric approaches.** Nonparametric kernel learning (NPKL) was introduced by Hoi, Jin, and Lyu (2007) as an alternative to MKL, in which a fully nonparametric kernel matrix is learned using pairwise constraints and can be solved using standard semidefinite programming (SDP) techniques. An efficient approach to NPKL from side information, SimpleNPKL, which can efficiently learn nonparametric kernels from large sets of pairwise constraints, was recently

introduced (Zhuang, Tsang, & Hoi, 2009). It would be interesting to investigate possible online extensions of this methodology.

The application of these methods is outside of the scope of this study but offers several potential avenues for further research.

3.7 Normalization. Normalizing features (so that each feature has ℓ_2-norm one across all training examples) or standardizing them (so that each feature has mean 0 and standard deviation 1 across all training examples) is known to be important for regularized linear classifiers or kernel classifiers (Duda & Hart, 1973; Shawe-Taylor & Cristianini, 2004). Empirically this has been shown to be especially important for MKL (Bach et al., 2004; Lanckriet et al., 2004), whether the kernels are linear or nonlinear. Kloft et al. (2011) suggest that the importance of normalization is owed to the bias introduced by regularization. As the optimal feature and kernel weights are requested to be small by imposing penalties on their norms, it stands to reason that this will be easier to achieve for features (or entire feature spaces, as implied by kernels) that are scaled to be of large magnitude, while downscaling them would require a correspondingly upscaled weight for representing the same predictive model. Hence the upscaling or downscaling of features is equivalent to modifying regularizers such that they penalize those features less or more. Generally the solution to this is to use isotropic regularizers, which penalize all dimensions uniformly. As a result, the kernels should be normalized (or standardized) in a sensible way in order to represent an isotropic prior over the features and the feature spaces, one that penalizes all weights in the same way.

Kloft et al. (2011) describe several approaches to normalization and use two particular types in their empirical analysis. They describe these methods:

1. **Multiplicative normalization.** As described by Shawe-Taylor and Cristianini (2004) and examined empirically by Zien and Ong (2007), this involves normalizing the kernels to have uniform variance of data points in the features space. Normally this method is combined with centering (the empirical mean of the data points in the feature space lies on the origin), which simplifies the normalization rule.
2. **Spherical normalization.** Each data point is rescaled to lie on the unit sphere. This may also have an effect on the scale of the features, as a spherically normalized and centered kernel is also always multiplicatively normalized.
3. **Input space normalization.** Each data point is normalized in the original space.
4. **Input space standardization.** Each data point is standardized in the original space, that is, shifted and rescaled to have mean 0 and standard deviation 1, before mapping into the feature space. Similar to spherical normalization, this will also affect the scale of the features

5. **Incremental standardization.** The mean and standard deviation of each feature within each feature space are updated incrementally using the update equations (see equation 3.1). Note, however, that the active set needs to be restandardized at each step as well,[8] leading to additional computational burden not found in methods 3 and 4:

$$\boldsymbol{\mu}_t = \frac{t-1}{t}\boldsymbol{\mu}_{t-1} + \frac{1}{t}\mathbf{x}_t,$$
$$\sigma_t = \sqrt{\frac{t-1}{t}\sigma_{t-1}^2 + \frac{t-1}{t^2}\left(\mathbf{x}_t - \boldsymbol{\mu}_{t-1}\right)^2}. \tag{3.1}$$

However, in online learning, where the full kernels are not observed beforehand, methods 1 and 2 are clearly not possible, even where the explicit feature space is available (as in the case of linear kernels for example), since it requires the entire training set to be present. Since normalization is important for (most) multiple kernel methods, this poses a problem. Methods 3 to 5 are the only possibilities available, but they represent a weaker compromise. Normalization or standardization in the input space clearly does not imply normalization or standardization in the feature space. However, it does represent a form of control over the vector sizes in the feature space and, as seen in the experimental section, does improve performance over no normalization.

The final method on the surface appears to make sense for linear kernels, as it should asymptotically converge to the standardization of the entire data set in the original space. We present experimental results for methods 3 to 5, as well as using no normalization.

Note also that it is possible to design online algorithms that automatically adapt to the norm of the samples observed up to any given time point. For example, Figures 5.2 and 5.3 of Shalev-Shwartz (2007) describe self-adaptive variants of the Winnow algorithm and aggressive quasi-additive family of algorithms, respectively, for binary classification. More recent approaches to solving this problem include an algorithm that adaptively chooses its regularization function based on the loss functions observed so far (McMahan & Streeter, 2010) and a new family of subgradient methods that dynamically incorporate the geometry of the data observed in earlier iterations to perform more informative gradient-based learning (Duchi, Hazan, & Singer, 2011). However, these methods are much more complex and are outside of the scope of this study.

[8]The restandardization of the active set can be ignored but will lead to poor performance if data points are kept in the active set from early on in the learning process, at which point the online estimate of the moments of the data may have been poor. We found this to be an essential step for this method to work. It should be noted that this actually breaks the theoretical guarantees of the algorithms.

4 Experiments

In this section, we present some empirical comparisons of a representative selection of the algorithms discussed earlier. In section 4.2, we present the analysis of a relatively small (in online learning terms) protein fold prediction data set. Due to the fact that multiple feature spaces are available, this data set has been used to benchmark and develop various different multiple kernel learning algorithms. Its compact nature allows us to rapidly evaluate a wide range of algorithms, discarding any that are too computationally expensive or show poor performance. Following on from this, we narrow down our selection of algorithms and analyze two object categorization data sets in section 4.3.

4.1 Algorithms and Specific Implementation Issues. This section lists the algorithms we evaluated, along with any specific implementation issues that arose with these algorithms. All algorithms were implemented in Matlab version 7.7 (R2008b).

Since the data sets we are using have multiple feature spaces, they naturally lend themselves to the application of multiple kernel learning (MKL) algorithms. Note, however, that most of the algorithms presented here are single kernel algorithms. Of course we can create MKL algorithms from any of the single kernel algorithms by using ad hoc kernel combination rules. One such rule is simply to use an (unweighted) sum of kernels, which corresponds to concatenating the feature spaces before creating a single kernel. Other kernel combinations could be used, such as products of kernels or nonlinear combinations of kernels, but these are outside the scope of this study. In order to do a complete analysis, ideally we would test each of the kernels separately using the single kernel methods as well as the kernel combinations. However, to keep the analysis contained, we treat the kernel perceptron (KP) as the benchmark algorithm. In doing so, we ran KP on each of the kernels separately, as well as with an unweighted sum of kernels (KP-sum). We then ran each of the single kernel methods using the unweighted sum in order to compare them with both KP-sum and the MKL methods. Of course, updating the kernel weights as well as the (dual) weight vectors in an online fashion is the ultimate goal, which at present only the OM-2 and OMKL methods attempt to do. Similar approaches could be taken to each of the single kernel methods, leading to many variants of online MKL, but that is outside of the scope of this work:

- **Kernel perceptron.** We used the vanilla (dual) implementation, which was then extended to the multiclass setting using the method of Zien and Ong (2007) as provided by the DOGMA toolbox (Orabona, 2009). We used a budget of 1000 (see section 3.2.2 for details) in the Caltech101 experiements and 5000 in the Caltech256 experiments to bound the computation time in the cases where the algorithm was

performing poorly (while this effectively turns this into the budget perceptron, this budget is not reached for any "useful" kernels).

- **Budget perceptron.** We used the random budget perceptron (RBP) variant, which throws away existing support vectors at random when the budget is exceeded. This method is seemingly naive but has good performance bounds and is extremely efficient. We used a maximum active set size of 200 samples.
- **Projectron.** We used the implementation from Orabona's (2009) DOGMA toolbox. The η (sparseness) parameter was set to its default value. We found that the algorithm was very insensitive to this parameter.
- **OM2.** The OM2 algorithm has a sparsity parameter p, which by default is $p = \frac{1}{1-\frac{1}{2\log(k)}}$. In the experiments that follow, this is approximately 1.25. We also tried the values $p = 1.01$ and $p = 1.99$ to approximate the ℓ_1 and ℓ_2 norms (these will sometimes be referred to as $p = 1$ and $p = 2$ for simplicity).
- **OMKL.** We implemented algorithm 1 from Jin et al. (2010), which has a discount (inverse sparsity) parameter $0 < \beta < 1$. We used the settings $\beta = 0.1, 0.5, 0.9, 0.99, 0.999$, since small values enforce sparsity over the kernel weights extremely quickly.
- **IVM.** We used active set sizes 50, 100, and 500, with a window size of 20 points (EP updates are performed only when 20 points are received). The buffer was chosen because if the initial EP did not include all of the classes, the future EP updates would never give posterior mass to those classes not included. We set the number of EP optimization iterations to 5.
- **PASS-GP.** We set the EP optimzation iterations, initial buffer, and window size to be the same as the IVM. Following Henao and Winther (2010), we set the inclusion parameter to 0.1, 0.3, and 0.6 and left the exclusion parameter at 0.99 after some experimenation.

Of course, we could have used any number of other possible choices of the various hyperparameters; these should be taken only as a representative sample rather than a definitive coverage of the various parameter spaces.

4.2 Protein Fold Prediction. The original data set from Ding and Dubchak (2001), based on SCOP PDB-40D, consists of 313 examples for training and 385 examples for testing with less than 35% sequence identity between any two proteins in the train and the test set. Furthermore, the extensions that Shen and Chou (2006) proposed exclude four proteins from the original data set (proteins 2SCMC and 2GPS from the training set, plus 2YHX 1 and 2YHX 2 from the test set), due to a lack of sequence records. The original data set is available online at http://ranger.uta.edu/~chqding/protein/, which also describes the 27 SCOP fold types (classes) (Dubchak, Muchnik, Holbrook, & Kim, 1995) together with the original

Table 2: Dimensionality of the Feature Spaces: Protein Fold Data.

Features	Dimensions
Composition	20
Hydrophobicity	21
Polarity	21
Polarizability	21
Secondary	21
Volume	21
L1	22
L4	28
L14	48
L30	80
SWblosum62	311
SWpam50	311

Note: Number of classes $c = 27$.

feature spaces in Ding and Dubchak (2001), the four proposed by Shen and Chou (2006) that describe pseudo–amino acid compositions (PseAA) estimated on different intervals of the protein sequence, and the two local alignment Smith-Waterman (SW)–based feature spaces, with different scoring matrices from Damoulas and Girolami (2008). The sizes of feature spaces are given in Table 2.

In order to calculate the holdout test accuracy, we stop the online algorithms after the first 313 examples have been seen and evaluate the resulting decision functions on the remaining 385 examples using the online-to-batch conversion described in section 1.1. To calculate the final cumulative accuracy, we run the online algorithms through the whole data set (train and test) and calculate the number of online errors made during learning. Note that the two metrics therefore have different training set sizes.

Tables 3 and 4 give holdout test accuracy and final cumulative (online) accuracy of the KP on all of the feature spaces individually, a KP using an unweighted sum of kernels, OM2, OMKL, and PASS-GP. In the columns are three kernel types:

- **lin:** linear
- **poly/lin:** second-order polynomial for global characteristics and linear for local characteristics (SW) following Damoulas and Girolami (2008)
- **RBF:** radial basis function kernels with $\sigma = \frac{1}{\sqrt{n}}$ as the width parameter.[9]

[9]This is by no means optimal but serves as a simple heuristic method for choosing the parameter value.

Table 3: Holdout Test Accuracies on Protein Fold Recognition Data.

Holdout Accuracy	lin U	lin N	lin S	lin SO	poly U	poly N	poly S	poly SO	RBF U	RBF N	RBF S	RBF SO
KP composition	16.61	14.26	27.15	26.89	19.22	17.75	31.64	32.43	17.86	33.63	**42.14**	34.62
KP hydrophobicity	3.81	3.81	5.01	17.02	3.19	3.81	4.49	19.84	13.68	4.65	25.22	**29.82**
KP polarity	3.81	3.81	5.01	15.56	3.55	3.81	4.49	18.38	13.73	5.07	23.13	**28.93**
KP polarizability	3.81	3.81	5.22	14.15	3.86	3.81	6.11	17.08	15.09	6.68	24.96	**29.50**
KP secondary	8.41	12.48	12.58	18.64	12.06	9.50	17.65	20.05	15.35	17.70	**33.58**	31.07
KP volume	3.81	3.81	5.43	16.97	5.07	3.81	5.69	19.90	14.57	6.63	26.89	**30.86**
KP L1	15.61	14.99	20.78	26.53	18.28	13.94	24.80	26.84	13.32	30.97	**35.51**	27.26
KP L4	21.31	18.96	23.03	26.48	22.04	18.02	24.96	27.89	12.90	**34.99**	30.81	23.86
KP L14	18.43	20.21	20.57	25.22	23.19	18.85	27.36	23.45	13.05	**36.08**	16.03	15.67
KP L30	15.77	17.28	19.53	20.57	19.58	20.31	29.30	16.03	12.79	**32.17**	13.52	14.20
KP SWblosum62	30.97	27.94	**54.46**	38.90	30.97	27.94	52.22	38.90	12.27	37.23	16.97	15.40
KP SWpam50	25.95	19.53	**50.65**	50.34	25.95	19.53	51.44	50.34	12.27	39.37	15.93	14.88
KP UWS	23.39	11.64	**57.18**	45.07	7.05	13.58	33.47	30.70	19.53	27.73	34.93	36.08
Proj UWS	4.39	10.08	**57.18**	7.52	4.44	10.08	29.45	3.71	21.83	26.89	27.26	1.57
OM2 ($p = 1.01$)	21.57	13.37	29.50	**48.25**	3.34	9.97	23.86	32.79	24.44	12.90	37.49	40.84
OM2 ($p = 1.25$)	21.36	17.49	34.57	**46.06**	7.73	12.48	24.18	32.06	24.44	16.29	37.60	40.84
OM2 ($p = 1.99$)	25.33	17.44	**55.20**	45.17	5.22	12.48	34.41	31.17	24.44	30.91	38.33	40.84
OMKL (beta = 0.1)	18.17	16.45	26.37	**60.31**	12.11	16.19	20.63	25.33	22.61	20.52	37.23	36.81
OMKL (beta = 0.5)	18.17	16.45	26.37	**60.31**	12.11	16.19	20.63	25.33	22.61	20.52	37.23	36.81
OMKL (beta = 0.9)	18.17	16.45	26.37	**60.31**	12.11	16.19	20.63	25.33	22.61	20.52	37.23	36.81
OMKL (beta = 0.99)	18.17	16.45	26.37	**60.31**	12.11	16.19	20.63	25.33	22.61	20.52	37.23	36.81
OMKL (beta = 0.999)	18.17	16.45	26.37	**60.31**	12.11	16.19	20.63	25.33	22.61	20.52	37.23	36.81
IVM (50/20)	28.88	28.09	**43.92**	10.03	30.91	32.38	29.56	13.63	19.22	24.44	24.44	1.57
IVM (100/20)	28.88	28.09	**43.92**	11.12	30.91	32.38	29.56	13.58	19.22	24.44	24.44	1.57
IVM (500/20)	29.09	28.62	**44.49**	18.85	28.77	32.01	29.87	11.85	19.22	25.27	25.54	1.57
PASS-GP (0.1/0.99/20)	7.73	25.33	**34.67**	11.85	30.91	32.38	20.52	13.42	19.22	25.12	17.81	1.57
PASS-GP (0.3/0.99/20)	5.74	31.59	**38.59**	12.85	8.83	17.44	16.45	11.59	19.22	37.86	31.17	1.57
PASS-GP (0.6/0.99/20)	11.44	45.07	**60.05**	8.25	39.58	49.71	47.89	13.32	40.99	39.22	41.04	1.57
VBpMKL	48.36	33.68	**62.14**	NA	32.38	28.67	46.32	NA	21.62	40.84	39.53	N/A

Note: Entries in boldface indicate the best combination of kernels and normalization for each classifier.

Table 4: Final Cumulative Accuracies on Protein Fold Recognition Data.

Final Cumulative Accuracy	lin U	lin N	lin S	lin SO	poly U	poly N	poly S	poly SO	RBF U	RBF N	RBF S	RBF SO
KP composition	9.65	9.45	15.50	17.11	12.80	9.07	19.61	17.62	19.23	19.61	**22.83**	21.29
KP hydrophobicity	3.99	3.79	3.67	9.20	3.92	3.79	3.34	9.52	7.91	4.05	8.94	**13.44**
KP polarity	3.99	3.92	3.86	9.39	4.12	3.92	3.60	9.26	7.14	3.99	10.48	**13.12**
KP polarizability	4.05	4.18	3.86	9.07	4.05	4.18	3.47	8.81	3.47	4.95	9.26	**10.10**
KP secondary	8.10	8.10	9.65	14.98	10.16	7.97	11.00	16.33	3.86	13.12	**25.02**	24.24
KP volume	3.99	4.31	3.92	10.03	3.92	4.31	3.99	9.97	5.08	4.82	11.06	**12.15**
KP L1	9.26	9.97	14.21	16.08	13.63	9.77	16.08	17.11	14.98	18.26	**20.51**	20.45
KP L4	11.38	10.61	14.02	18.01	14.66	10.03	14.79	17.30	7.91	18.84	19.42	**19.49**
KP L14	10.87	10.23	10.93	**15.88**	12.80	9.90	13.89	12.60	0.00	15.43	14.53	14.60
KP L30	9.84	8.62	9.39	11.51	10.68	9.00	11.90	7.65	0.00	13.38	**11.77**	11.25
KP SWblosum62	15.82	14.60	30.29	**35.95**	15.82	14.60	31.70	**35.95**	0.00	20.06	0.32	0.32
KP SWpam50	12.48	10.80	31.38	**33.76**	12.48	10.80	30.87	**33.76**	0.00	22.70	0.32	0.32
KP UWS	12.73	9.13	26.95	**40.13**	5.47	7.91	15.24	29.71	19.68	13.83	14.47	21.48
Proj UWS	5.02	6.62	**26.95**	7.46	5.02	6.62	26.50	7.46	6.88	13.63	15.18	0.00
OM2 ($p = 1.01$)	15.82	17.04	**37.30**	34.53	9.65	10.48	11.90	11.19	20.77	25.40	25.53	24.12
OM2 ($p = 1.25$)	14.15	14.34	33.57	**37.75**	8.04	12.48	12.67	17.56	20.77	23.79	17.43	24.05
OM2 ($p = 1.99$)	12.73	9.39	26.82	**40.71**	5.59	7.91	16.21	29.77	20.77	14.66	16.40	24.05
OMKL (beta = 0.1)	11.45	12.80	36.59	**39.04**	12.93	12.41	36.27	38.78	19.94	18.52	24.82	23.79
OMKL (beta = 0.5)	11.32	12.35	37.11	**40.51**	12.86	12.41	36.40	39.81	19.87	19.16	23.86	23.34
OMKL (beta = 0.9)	10.35	12.22	34.86	**42.19**	9.65	11.90	25.98	40.96	20.06	16.98	19.81	21.99
OMKL (beta = 0.99)	10.16	11.19	18.59	**44.05**	8.30	11.06	15.11	28.17	19.74	14.34	15.11	21.80
OMKL (beta = 0.999)	10.23	11.19	17.23	**44.31**	8.23	11.06	14.98	23.73	19.74	13.96	14.98	21.80
IVM (50/20)	18.07	13.96	23.09	**27.91**	18.78	16.72	16.53	23.99	10.10	12.60	12.86	16.01
IVM (100/20)	18.01	13.96	23.09	**28.62**	18.52	16.66	16.59	24.12	10.10	12.60	12.86	16.01
IVM (500/20)	18.14	14.15	23.86	**30.16**	16.72	16.72	16.27	23.79	10.10	12.73	13.38	16.08
PASS-GP (0.1/0.99/20)	18.97	15.63	25.98	**29.71**	18.78	16.72	17.17	23.73	10.10	13.38	14.34	16.01
PASS-GP (0.3/0.99/20)	29.39	19.81	**32.09**	30.87	18.65	23.34	21.16	30.03	10.10	17.94	17.62	16.01
PASS-GP (0.6/0.99/20)	29.45	19.94	32.22	**38.39**	23.79	23.67	21.87	35.31	11.51	17.94	17.62	20.64

Note: Entries in boldface indicate the best combination of kernels and normalization for each classifier.

The experiments were run using no normalization (lin U, poly/lin U, and RBF U), normalizing each example to be norm 1 (lin N, poly/lin N, RBF N), standardizing each example (lin S, poly/lin S, and RBF S), and online standardization (lin SO, poly/lin SO, and RBF SO). The results are reasonably consistent for both holdout test accuracy and cumulative accuracy. For the KP on individual feature spaces, the RBF kernel performed best for all feature spaces except for the local characteristics (SW), where linear kernels performed best. In both cases, standardizing each example before computing kernels performed better than normalizing or no normalization. For KP on the unweighted sum of kernels, linear kernels on standardized data performed best under both metrics, and the same is true for the OM2 algorithm and PASS-GP. For OMKL, linear kernels on standardized data were best in terms of cumulative accuracy, but RBF kernels on standardized data performed better in terms of holdout test accuracy.

The best method overall in terms of holdout test accuracy was the OMKL algorithm using linear kernels and the online standardization method (60.31%) for all values of the discount parameter β, followed closely by PASS-GP with the inclusion parameter at 0.6 (60.05%), both of which are close to the accuracy achived by the the variational Bayes probalistic multiple kernel learning (VBpMKL) method of Damoulas and Girolami (2008) (an offline method), which was included for comparison (62.14%). For reference, the KP and the projectron on the unweighted sum of kernels (KP UWS and Proj UWS) performed best using linear kernels on standardized data, and both achieved an accuracy of 57.18%; the OM2 followed behind this at 55.20%, with $p = 1.99$. The best-performing single kernel was SWblosum62 (linear kernel, standardized) at 54.46%. The IVM performed worse than the best single kernel.

The best method overall in terms of cumulative test accuracy was again OMKL ($\beta = 0.999$) with 44.31%, followed by OM2 with the least sparse setting ($p = 1.99$) with 40.71% (both using linear kernels and the online standardization method). In terms of cumulative accuracy, KP UWS was close behind at 40.13%, and PASS-GP was further behind still (38.39%). The best single kernel was again SWblosum62 (linear kernel, online standardization). The projectron performed poorly according to this metric, as did the IVM.

For all methods, it is clear that either the example-by-example standardization or the online standardization method is extremely important to achieve good results. In most cases, the online standardization method led to the best performance in terms of both holdout test accuracy and final cumulative accuracy, but there were cases where the example-by-example standardization performed just as well if not better, such as for the KP UWS and OM2 algorithms in terms of holdout test accuracy, which possibly indicates that the smaller training set in this setting meant that the online standardization estimates had not settled down.

Note also that there is no a priori method for choosing the discount parameter β in OMKL or the sparsity parameter p in the OM2 algorithm unless a validation set is available. The same is true for any kernel parameters, such as the polynomial degree or σ parameter of the RBF kernel. In the true online setting, this makes a simple algorithm such as the KP with an unweighted sum of kernels, combined with example-by-example standardization, an appealing option.

The results for each of the methods using linear kernels on standardized data are repeated in Tables 3 and 4 for holdout test accuracy and final cumulative accuracy respectively.

Figure 1 shows the effect of the different types of normalization on the performance of the KP using the best-performing feature space (SWblosum62, top left), KP on an unweighted sum of kernels (top right), OM2 (bottom left), and OMKL (bottom right). Notice that for all four algorithms, the best normalization method is the online standardization method, with a markable improvement being seen especially for the OMKL algorithm. However, Figure 2 shows that the online standardization method is at least an order of magnitude slower than the other methods (in total execution time) due to the fact that the active set needs to be restandardized each time it is updated. This could prove to be especially inefficient for algorithms that maintain large (or unbounded) active sets. The fastest method (significantly) was the (offline) standardization method, which was also the second best in terms of average error rate.

Figures 3 and 4 show the final cumulative accuracy of the algorithms on the protein fold prediction data set using linear kernels with (example-by-example) standardization and online standardization, respectively. For the standardized data, the KP with an unweighted sum of kernels performs in a similar manner to the KP using either of the two best kernels (SWblosum62 and SWpam50). The performance of OMKL is also similar. The IVM and PASS-GP both perform poorly, with error rates significantly worse than the best kernel or unweighted sum. The OM2 algorithm (with $p = 1$ or $p = 1.2$) is the only algorithm to outperform the other methods. The story is similar with the online standardization method, except that the OM2 algorithm now performs on a par with the unweighted sum, and the OMKL algorithm (with $\beta = 0.99$ or $\beta = 0.999$) is now the only one that outperforms the other methods. The projectron algorithm performs poorly in this setting. In general, the online standardization method yields better results, though it is not clear that these differences are significant.

The holdout test accuracies of the algorithms on the protein fold prediction data set are given in Figures 5 and 6 using linear kernels with (row-wide) standardization and online standardization, respectively. As discussed previously, the OMKL algorithm appears to be the best-performing algorithm in this setting, with accuracies close to that of the VBpMKL algorithm (although, interestingly, with a much smaller variance).

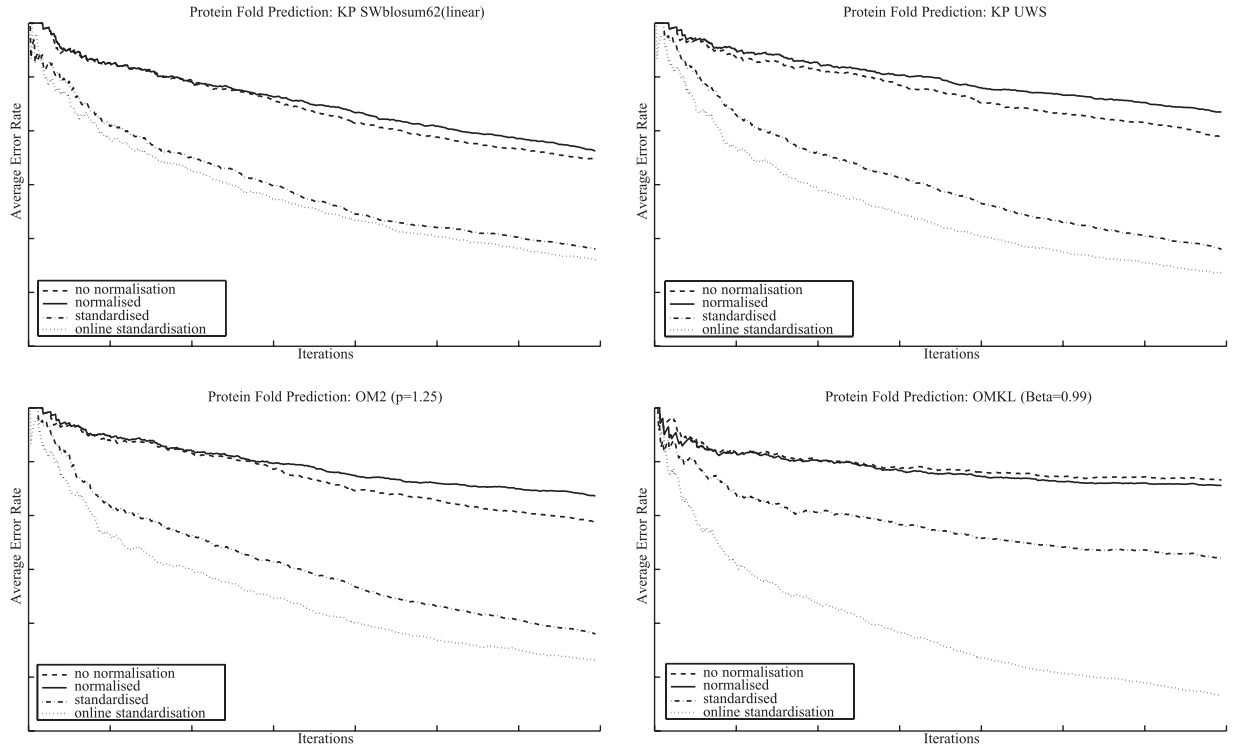

Figure 1: Protein fold prediction: Effect of normalization. Average error rates for the KP using the best-performing feature space (SWblosum62, top left), KP on an unweighted sum of kernels (top right), OM2 (bottom left), and OMKL (bottom right) for the four normalization types.

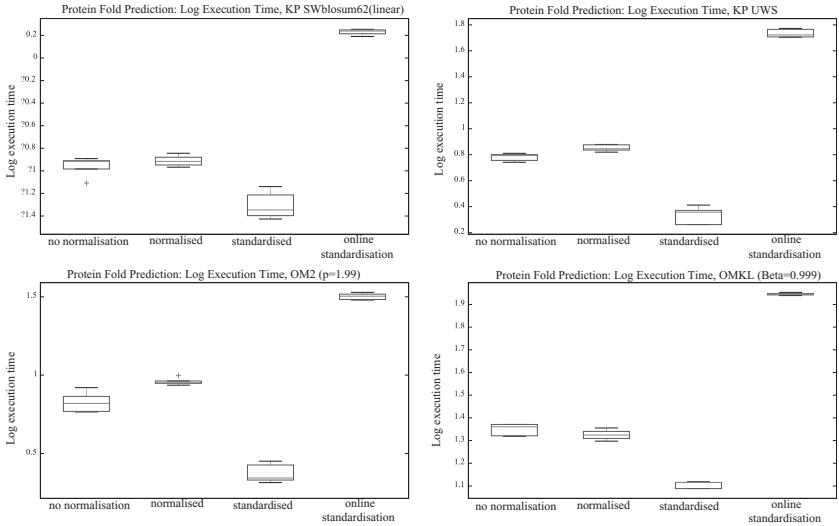

Figure 2: Protein fold prediction: Execution time of normalization methods. Log of execution times for KP using the best-performing feature space (SWblosum62, top left), KP on an unweighted sum of kernels (top right), OM2 (bottom left), and OMKL (bottom right) for the four normalization types.

The log of the execution time of the algorithms is shown in Figure 7. Here we can see that the multiple kernel methods (OM2 and OMKL) are an order of magnitude slower than the KP on a single feature space, but not signficantly slower than the KP on an unweighted sum of kernels (and in some cases faster). This shows that for these methods, the main bottleneck is in the kernel evaluations. Both the IVM and PASS-GP are two to three orders of magnitude slower than the multiple kernel methods and are using only an unweighted sum of kernels. Here the bottleneck is in the expectation propagation (EP) step, which needs to be run each time new examples are included in the active set. Since the EP step needs to be run for each one-versus-rest classifier separately, this also scales with the number of classes. From this, we can conclude that for practical problems with many classes, IVM and PASS-GP (and indeed any other method that uses EP for parameter updating) are unsuitable. We will therefore not include these methods in the evaluation on the larger data sets to follow.

Figure 8 shows the final kernel weights for the OM2 algorithm (top), the OMKL algorithm (middle), and the VBpMKL algorithm (bottom) on the protein fold prediction data set using linear kernels on standardized data. Note that the two kernels found by OM2 and OMKL were the two best-performing individual kernels in terms of both cumulative accuracy (see Table 5) and holdout test accuracy (see Table 4), with most weight

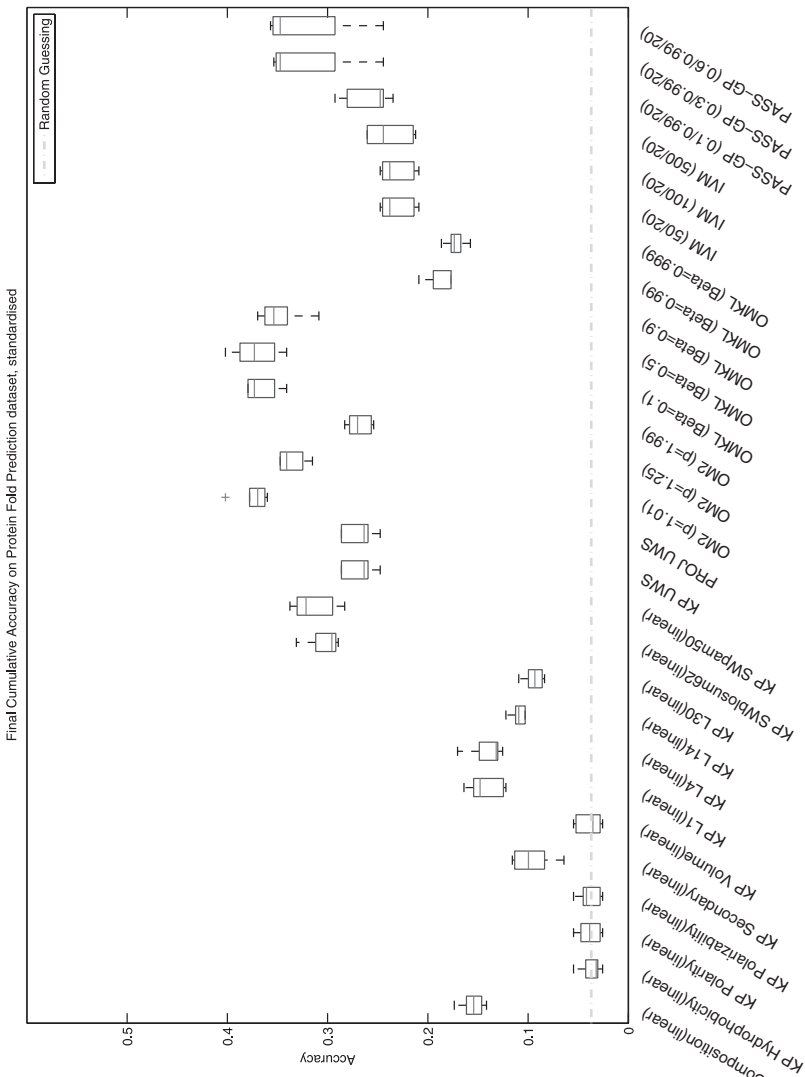

Figure 3: Protein fold prediction (standardized). Final cumulative accuracy of the KP on all of the feature spaces individually, a KP using an unweighted sum of kernels, OM2, OMKL, and PASS-GP, using standardized data. Linear kernels were used for all feature spaces. The baseline of random guessing is shown by the dash-dotted line.

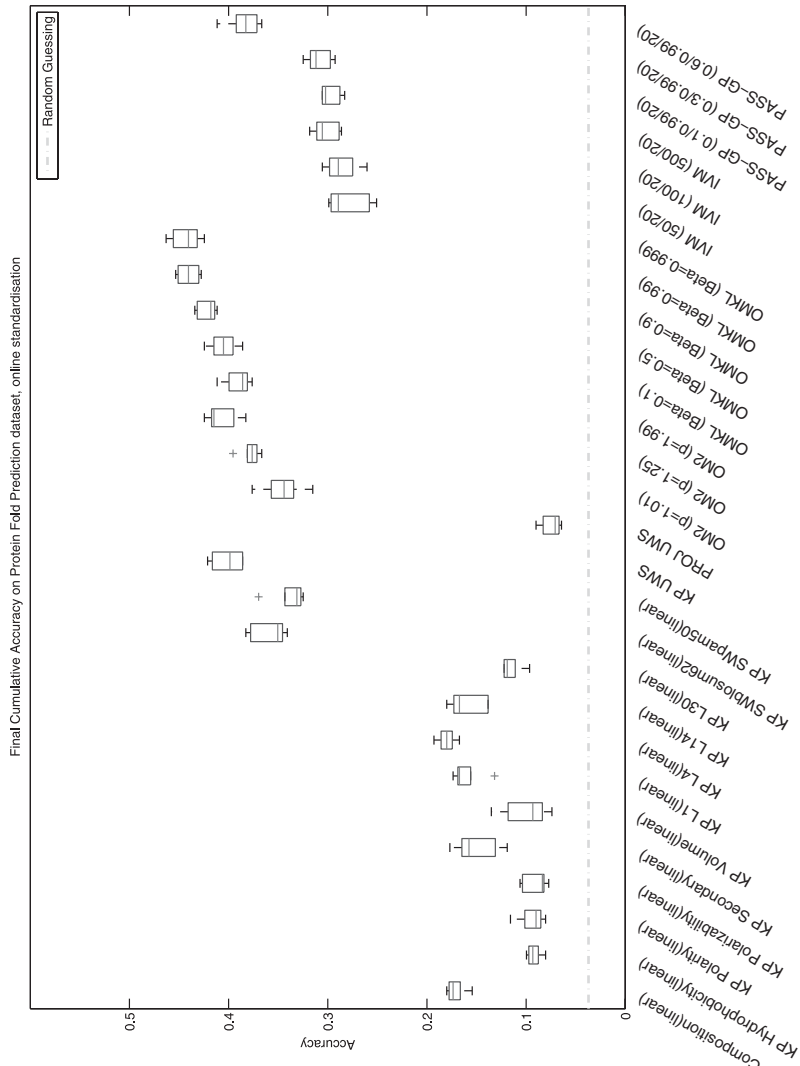

Figure 4: Protein fold prediction (online standardization). Final cumulative accuracy of the KP on all of the feature spaces individually, a KP using an unweighted sum of kernels, OM2, OMKL, and PASS-GP, using online standardization. Linear kernels were used for all feature spaces. The baseline of random guessing is shown by the dash-dotted line.

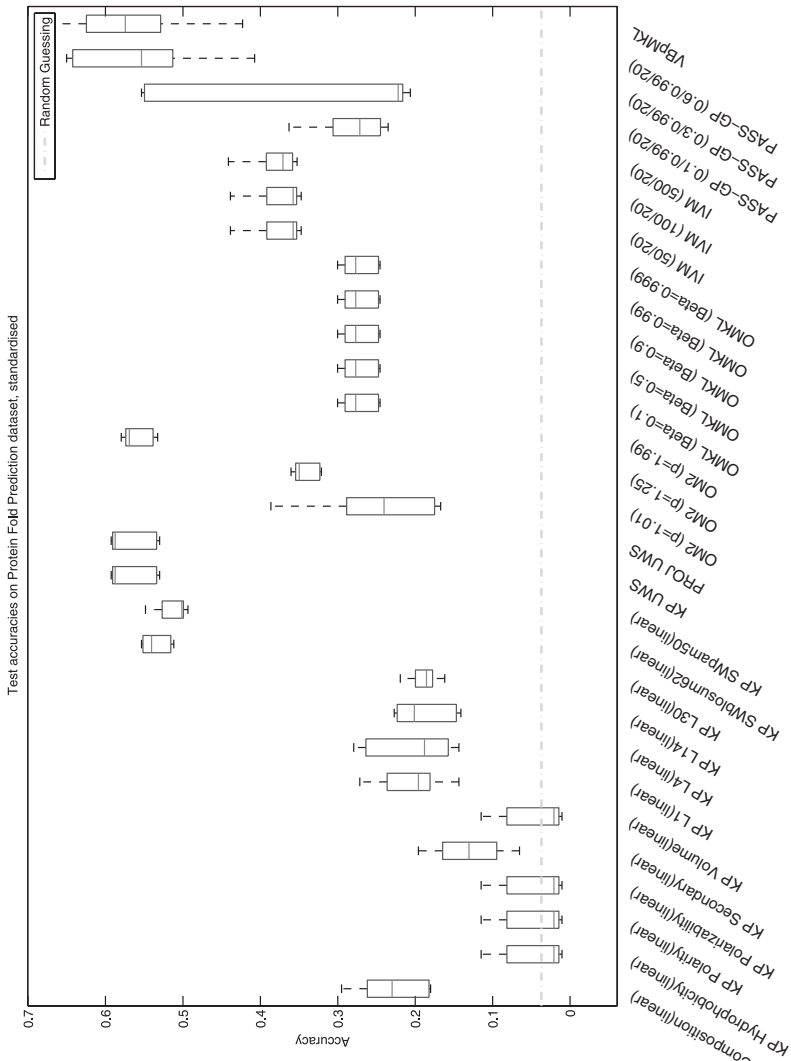

Figure 5: Protein fold prediction (standardized). Holdout test accuracy of the KP on all of the feature spaces individually, a kernel perceptron and projectron using an unweighted sum of kernels, OM2, OMKL, IVM, and PASS-GP. Linear kernels were used for all feature spaces. The VBpMKL method of Damoulas and Girolami (2008; an offline method) is included for comparison. The baseline of random guessing is shown by the dash-dotted line.

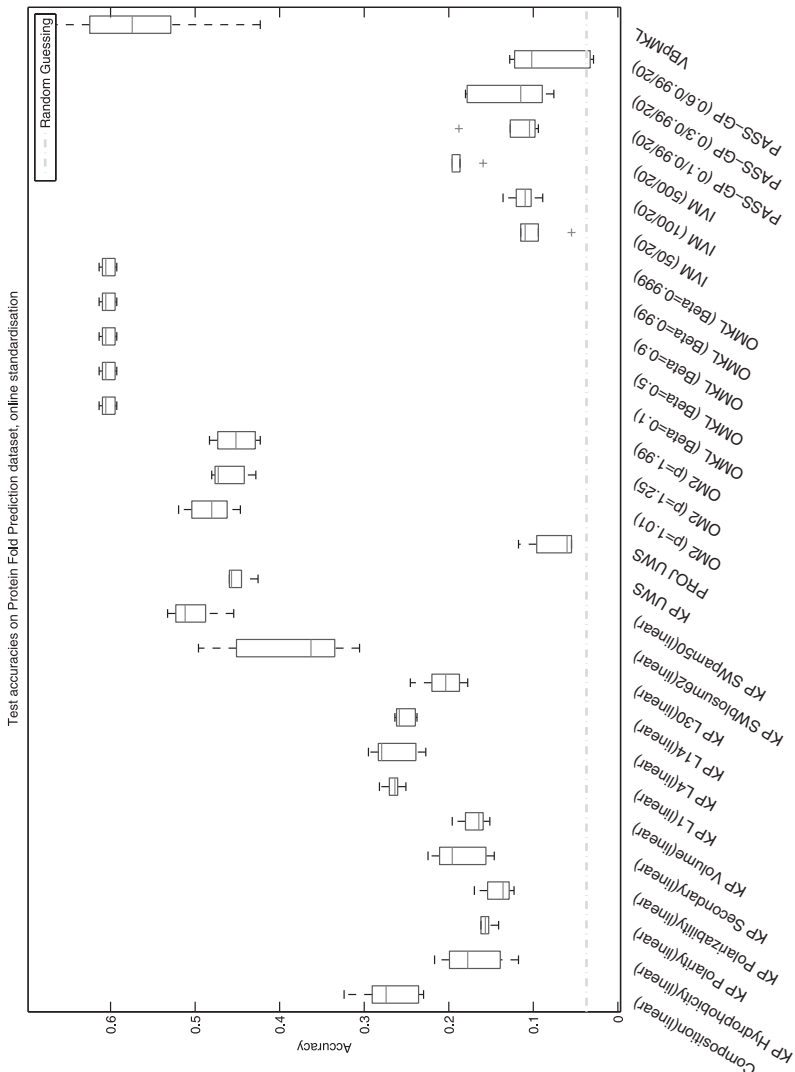

Figure 6: Protein fold prediction (online standardization). Holdout test accuracy of the KP on all of the feature spaces individually, a KP and projectron using an unweighted sum of kernels, OM2, OMKL, IVM, and PASS-GP. Linear kernels were used for all feature spaces. The VBpMKL method of Damoulas and Girolami (2008; an offline method) is included for comparison. The baseline of random guessing is shown by the dash-dotted line.

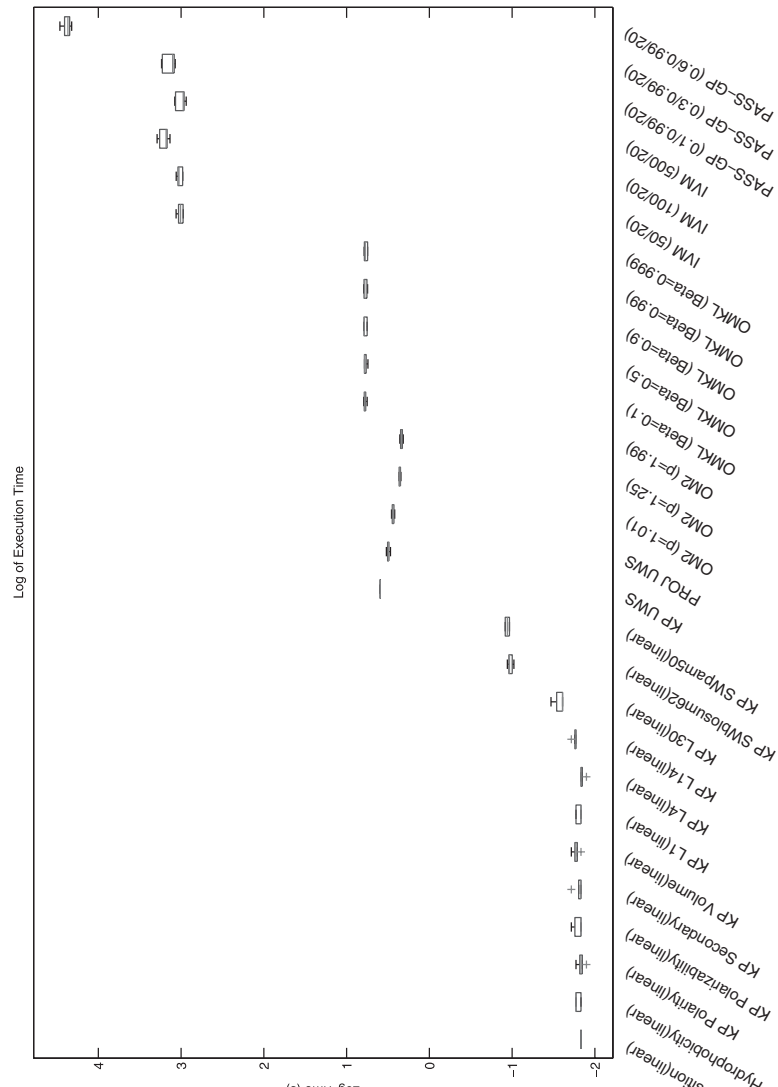

Figure 7: Protein fold prediction (standardized). Log execution time of the KP on all of the feature spaces individually, a KP and projectron using an unweighted sum of kernels, OM2, OMKL, IVM, and PASS-GP. Linear kernels were used for all feature spaces together with example-by-example standardization.

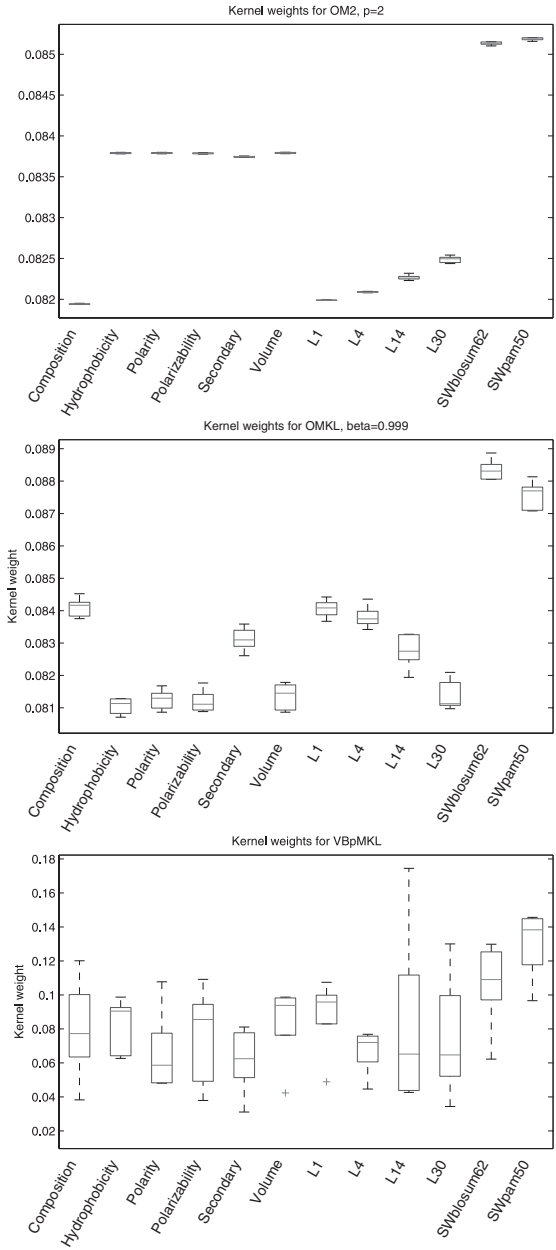

Figure 8: Protein fold prediction: OM2 ($p = 2$) (top), OMKL ($\beta = 0.999$) (middle), and VBpMKL (bottom) kernel weights. Box plot of final kernel weights on the protein fold prediction data.

in both cases being given to the best-performing kernel (SWblosum62). This seems to validate the kernel selection method in both algorithms. The VBpMKL algorithm does not enforce sparsity, which results in much more even weightings, but the two best-performing kernels do tend to get the greatest mass. It is worth noting that the OM2 algorithm has the least variation in weights and the VBpMKL algorithm has the most.

4.3 Visual Object Recognition. In this section we provide results of empirical analysis of the two Caltech data sets, Caltech101 and Caltech256. These data sets have been used extensively in the online learning and MKL literature (see Bosch, Zisserman, & Munoz, 2007a, 2007b, 2008; Varma & Ray, 2007; Gehler & Nowozin, 2009; Orabona et al., 2009). In general, most studies tend to use a small subselection of the data to train the algorithms (e.g., 30 examples from each class). Here we report results using the entire data set but do not examine the holdout test error. Note that the results stated in Bosch et al. (2007a, 2007b, 2008) and Varma and Ray (2007) were found not to be reproducible (see Gehler & Nowozin, 2009) as the distance matrices were found to contain perfect test label information. (See the appendix, which contains Figures 14–20, placed there for ease of reading of the text.) The results of Gehler and Nowozin (2009) seem to suggest that linear programming boosting (LPBoost) Demiriz et al. (2002) outperformed MKL approaches (using SimpleMKL; Rakotomamonjy et al., 2008) on the data.

4.3.1 Caltech101. The Caltech101 data set consists of 101 object categories collected by Fei-Fei, Fergus, and Perona (2004, 2006). Each object category contains between 40 and 800 images. The size of each image is roughly 300×200 pixels. All images are annotated with the following information: a bounding box of the object, and a carefully traced silhouette of the objects by a human subject. There are 9146 images in total.

The Caltech101 data set has several advantages over other similar data sets: almost all the images within each category are uniform in image size and in the relative position of interest objects, meaning that no cropping and scaling the images needs to be done before they can be used; there is a low level of clutter or occlusion; the data set has detailed annotations. However, there are also several weaknesses to the Caltech101 data set (Pinto, Cox, & DiCarlo, 2008; Fei-Fei et al., 2004): The Caltech101 data set contains a limited number of the possible object categories (although this number is still high compared to many typical multiclass machine learning problems); certain categories are not represented as well as others, containing as few as 31 images, meaning that number of images used for training must be 30 or fewer; the images are very uniform in presentation, left and right aligned, and usually not occluded, meaning that the images are not always representative of practical inputs that the algorithm being trained might be expected to see (clutter, occlusion, and variance in relative position and orientation);

some images have been rotated and scaled from their original orientation and suffer from some amount of artifacts or aliasing. The Caltech256 data set attempts to overcome some of these shortcomings.

4.3.2 Caltech256. The Caltech 256 data set consists of 256 object categories collected by Griffin, Holub, and Perona (2007). Caltech256 was collected in a similar manner with several improvements: (1) the number of categories was more than doubled, (2) the minimum number of images in any category was increased from 31 to 80, (3) artifacts due to image rotation are avoided, and (4) a new and larger clutter category is introduced for testing background rejection. There are 30,607 images in total.

4.3.3 Image Features. We used the set of image features made available by Peter Gehler and Sebastian Nowozin at http://www.vision.ee.ethz.ch/~pgehler/projects/iccv09/#download as described in Gehler and Nowozin (2009). Several software packages were used for the computation of image features. The code for the creation of the PHOG features was obtained from http://www.robots.ox.ac.uk/~vgg/research/caltech/phog/phog.zip, and SIFT descriptors are computed with http://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/compute_descriptors.ln.gz. The *v1plus* descriptors were obtained using the implementation by Pinto et al. (2008). The region covariance and local binary pattern descriptors were implemented by Gehler et al. (2009). The features are described in more detail below:

- **PHOG shape descriptor.** Shape information is modeled using the PHOG descriptor proposed in Bosch et al. (2007a). The descriptor is a histogram of oriented (Shp360) or unoriented (Shp180) gradients computed on the output of a Canny edge detector. The oriented histogram, Shp360, contains 40 bins and the unoriented Shp180 20 bins, yielding a total of 2×4 kernels.
- **Appearance descriptor.** Appearance information is modeled using SIFT descriptors (Lowe, 1999), which are computed on a regular grid on the image with a spacing of 10 pixels and for the four different radii, $r = 4, 8, 12, 16$. The descriptors are subsequently quantized into a vocabulary of visual words generated by k-means clustering. Here four variants are used: two codebook sizes (300 and 1000 elements) and gray image descriptors (128 dims), as well as HSV-SIFT ($3 * 128 = 384$ dims), with a total of 4×4 kernels.
- **Region covariance.** Covariances of simple per pixel features described in Tuzel (2007) are used and tangent space projected. A pyramid representation yields three kernels.
- **Local binary patterns.** Local binary pattern features (LBP) as described by Ojala, Pietikäinen, and Mäenpää (2002) are implemented

Table 5: Dimensionality of the Feature Spaces: Caltech101 and Caltech256.

Features	Dimensions
dense_bow/oneForAll_nr1_K1000/	1000
dense_bow/oneForAll_nr1_K1000/	1000
dense_bow/oneForAll_nr1_K1000/	1000
dense_bow/oneForAll_nr1_K300/	300
dense_bow/oneForAll_nr1_K300/	300
dense_bow/oneForAll_nr1_K300/	300
lbp	777
phog/A180_K20/Level0	20
phog/A180_K20/Level1	80
phog/A180_K20/Level2	320
phog/A180_K20/Level3	1280
phog/A360_K40/Level0	40
phog/A360_K40/Level1	160
phog/A360_K40/Level2	640
phog/A360_K40/Level3	2560
phog/subwindows/A180_K20	2000
phog/subwindows/A360_K40	4000
regcovn	588
v1plus	4000

using histograms of uniform rotation-invariant LBP8 features, resulting in three kernels.

- **v1plus.** In Pinto et al. (2008), a population of locally normalized, thresholded Gabor functions spanning a range of orientations and spatial frequencies is derived and advocated as particular simple features. This generates one kernel.

The sizes of feature spaces are given in Table 5. Note that we downsampled *v1plus* from 127,165 to 4000 by selecting the last 4000 features in order to make the dimensionality more manageable.[10]

We chose to use linear kernels for all feature spaces to avoid the computational burden associated with running the algorithms with multiple kernel hyperparameter settings. It is possible that the use of nonlinear feature mappings, such as provided by polynomial, RBF or chi-squared kernels (the last of which is particularly appropriate for histogram like features), could improve overall performance. However, in this study, we are not

[10]This choice of subsampling was done through observation: the last 4000 features on a random subset of the data were nearly full rank, whereas the first 4000, or a random subset of features, had much lower rank. Higher-order Gabor filters would therefore appear to be more meaningful in this setting. Note that we do not claim that this is the optimum subsampling of these features.

trying to give definitive best results on the problem; rather, our purpose is to compare the algorithms in a fair way.

4.3.4 Results. For each of the data sets, we chose to use two baseline algorithms: a blind algorithm that simply performs random guessing and the vanilla KP run on each feature space individually using linear kernels. We also ran KP using an unweighted sum of kernels as before, followed by OM2 and OMKL. PASS-GP was omitted from the larger experiments because the performance was no better on the protein fold prediction data set despite being several orders of magnitude slower.

Table 6 summarizes the final cumulative accuracies (mean and standard deviations) for the KP on all feature spaces individually, a KP using an unweighted sum of kernels (KP-UWS), the OM2 using three different settings for the sparsity parameter (p = 1, 1.2, 2), and OMKL using four different settings of the discount parameter ($\beta = 0.1, 0.5, 0.9, 0.99$). Note that the performance of some of the individual kernels individually is very good, even without normalization (e.g., phog/A360_K20/Level3, phog/A360_K40/Level3 and phog/subwindows/A360_K40, all of them variations on the PHOG Shape Descriptor). KP-UWS performs poorly without normalization but performs well using example-by-example standardization (although the final accuracy is less than the best individual kernels). The OM2 algorithm achieves the best overall accuracy (98.34%), although with a relatively large standard deviation (2.95%) using example-by-example standardization and with the default sparsity setting ($p = 1.2$). The OMKL algorithm achieves the next highest accuracy (97.24%) with a much smaller standard deviation (0.04%) using the same standardization and the discount parameter $\beta = 0.99$. The OMKL algorithm is quite insensitive to the particular setting of β. However, for the OM2 algorithm, there seem to be specific cases of catastrophic failure, such as when $p = 2$ and example-by-example standardization is used.

Figures 18, 19 and 20 in the appendix show the evolution of the kernel weights for the OM2 algorithm for the first 1000 iterations on the Caltech101 data set (standardized) with $p = 1$, $p = 1.2$ (default), and $p = 2$, respectively. Note that in Figure 20, the scale is finer to show subtle variation in the kernel weights; on a $0 \rightarrow 1$ scale, the weights appear uniform. Note that the single kernel found by the ℓ_1-norm version was the best-performing individual kernel in Figure 15, and the two kernels that have the largest weight in the $\ell_{1.2}$ and ℓ_2 versions were the two best-performing individual kernels. This seems to further validate the kernel selection method in the OM2 algorithm, while outlining the effect of the sparsity parameter p.

As above, Figure 9 shows the evolution of the kernel weights for the OMKL algorithm. The two most prominent kernels are the same as those chosen by the OM2 algorithm (default sparsity level), the best-performing

Table 6: Caltech101: Summary of Final Cumulative Accuracies for the Kernel Perceptron on All Feature Spaces Individually, a Kernel Perceptron Using an Unweighted Sum of Kernels (KP-UWS), the OM2, and OMKL Algorithms.

	Unnormalized		Normalized		Standardized		Online SD	
Algorithm	Mean	SD	Mean	SD	Mean	SD	Mean	SD
KP-dense_bow/oneForAll_nr1_K1000	18.79	0.23	18.76	0.16	18.80	0.35	18.79	0.23
KP-dense_bow/oneForAll_nr1_K1000	38.29	0.46	39.73	0.13	40.07	0.30	38.29	0.46
KP-dense_bow/oneForAll_nr1_K1000	43.43	0.48	43.76	0.29	44.27	0.21	43.43	0.48
KP-dense_bow/oneForAll_nr1_K300	34.86	0.87	35.31	0.09	36.56	0.24	34.86	0.87
KP-dense_bow/oneForAll_nr1_K300	34.45	0.41	35.80	0.14	36.16	0.24	34.45	0.41
KP-dense_bow/oneForAll_nr1_K300	38.39	0.45	39.98	0.19	40.39	0.27	38.39	0.45
KP-lbp	33.72	0.39	32.95	0.34	34.06	0.30	33.72	0.39
KP-phog/A180_K20/Level0	14.00	0.22	14.85	0.19	12.08	0.33	14.00	0.22
KP-phog/A180_K20/Level1	92.20	0.23	92.30	0.21	92.65	0.13	92.20	0.23
KP-phog/A180_K20/Level2	91.98	0.09	93.82	0.06	94.14	0.08	91.98	0.09
KP-phog/A180_K20/Level3	95.63	0.52	96.60	0.05	96.63	0.03	95.63	0.52
KP-phog/A360_K40/Level0	29.30	0.21	29.68	0.28	28.84	0.39	29.30	0.21
KP-phog/A360_K40/Level1	94.75	0.05	95.10	0.10	95.41	0.08	94.75	0.05
KP-phog/A360_K40/Level2	92.99	0.40	96.07	0.06	96.19	0.05	92.99	0.40
KP-phog/A360_K40/Level3	96.57	0.03	96.67	0.04	96.68	0.02	96.57	0.03
KP-phog/subwindows/A180_K20	42.61	0.16	42.81	0.18	42.34	0.27	42.61	0.16
KP-phog/subwindows/A360_K40	96.15	0.03	96.19	0.07	96.30	0.05	96.15	0.03
KP-regcovn	44.80	0.16	42.68	0.27	44.16	0.23	44.80	0.16
KP-v1plus	26.86	0.26	26.00	0.32	26.67	0.29	26.86	0.26
KP-UWS	33.87	1.10	91.59	0.17	95.37	0.02	91.97	0.24
OM2, $p = 1$	4.57	1.03	95.08	4.02	95.93	0.06	92.83	0.10
OM2, $p = 1.2$	96.80	0.06	26.68	2.08	98.34	2.95	95.39	0.06
OM2, $p = 2$	96.06	0.07	96.72	0.14	33.50	1.32	91.93	0.17
OMKL (Beta = 0.1)	96.40	0.06	96.61	0.06	97.07	0.05	96.40	0.06
OMKL (Beta = 0.5)	96.16	0.05	96.61	0.06	97.10	0.06	96.16	0.05
OMKL (Beta = 0.9)	93.31	0.33	96.60	0.05	97.24	0.04	93.31	0.33
OMKL (Beta = 0.99)	70.05	2.79	94.06	0.24	96.39	0.12	70.05	2.79

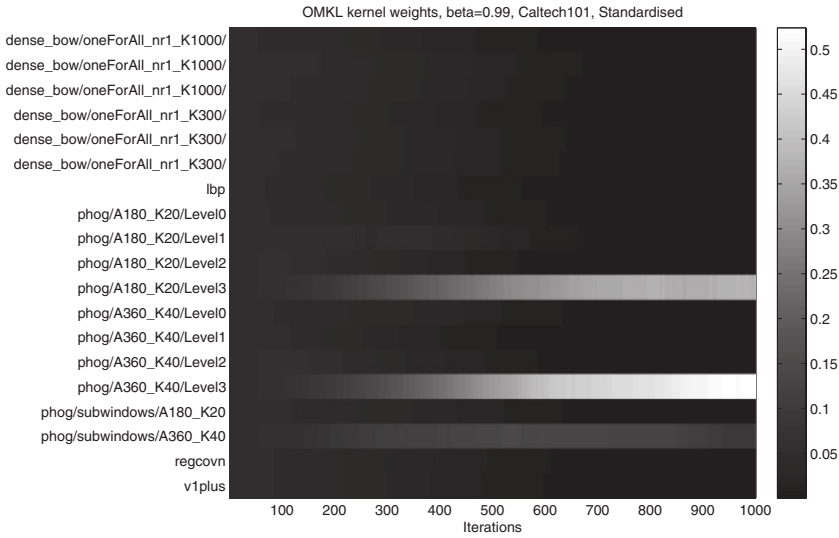

Figure 9: Caltech101: OMKL kernel weights ($\beta = 0.99$). Kernel weights over the first 1000 iterations on the Caltech101 data set using example-by-example standardization.

individual kernels. This also seems to validate the kernel selection method in the OMKL algorithm.

Figure 10 shows learning curves over the time for KP with the best individual kernel on the unnormalized data (PU-phog/A180_K20/Level3) and normalized data (PN-phog/A180_K20/Level3) and an unweighted sum of kernels on the unnormalized (UWS-U) and normalized data (UWS-N). Note that the best individual kernels differ from those that were found on the Caltech101 data set. It can be observed from the plot that the normalization has a huge effect on the number of errors committed by KP using an unweighted sum of kernels (in fact, the unnormalized version barely performs better than random guessing). Note, however, that for the normalized version, the error is significantly better than that of the best individual kernel.

Figure 11 shows learning curves over the time for the random budget perceptron (RBP) with the best individual kernel on the normalized data (BPN-phog/A180_K20/Level3) and an unweighted sum of kernels on the normalized data (UWSB-N), with the respective KP algorithms (PN-phog/A180_K20/Level3 and UWS-N) for comparison. We can see that on the best-performing kernel and for the unweighted sum of kernels, the RBP learning curves are significantly worse than the respective KP learning curves. Although a direct comparison is somewhat unfair, as the resulting classifier in this case is five times sparser in the case of the RBP, it shows that

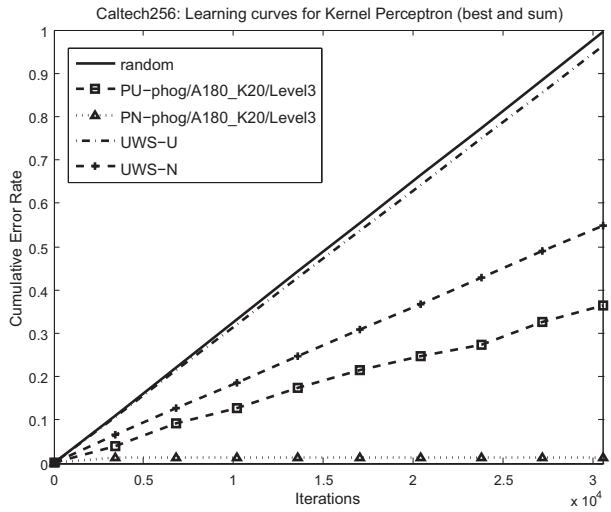

Figure 10: Caltech256: KP (best and sum). Learning curves for the KP with the best individual kernel on the unnormalized data (PU-phog/A360_K40/Level3) and normalized data (PN-phog/A360_K40/Level3) and an unweighted sum of kernels on the unnormalized (UWS-U) and normalised data (UWS-N).

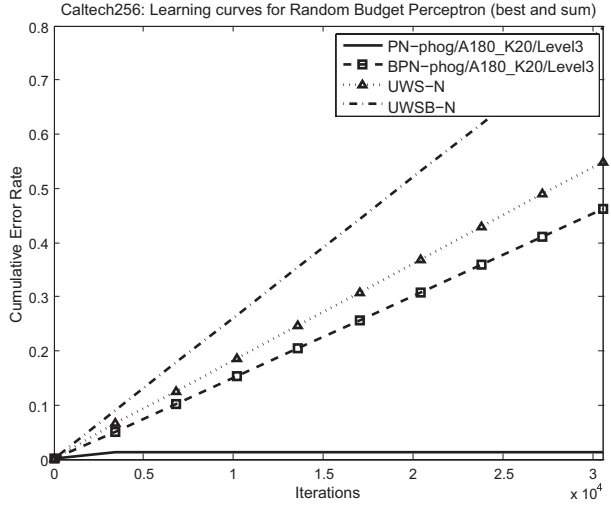

Figure 11: Caltech256: RBP (best and sum). Learning curves for the random budget perceptron (RBP) with the best individual kernel on the normalized data (BPN-phog/A180_K20/Level3) and an unweighted sum of kernels on the normalized data (UWSB-N), with the respective KP algorithms (PN-phog/A180_K20/Level3 and UWS-N) for comparison.

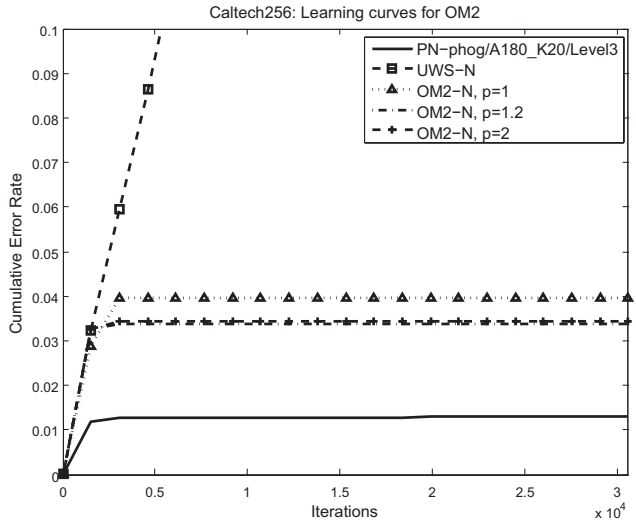

Figure 12: Caltech256: OM2. Learning curves for the OM2 algorithm for three different sparsity levels ($p = 1$, $p = 1.2$, $p = 2$), plotted along with KP using the best individual kernel (BPN-phog/A180_K20/Level3) and using an unweighted sum of kernels (UWS-N) on the normalized data.

care must be taken when attempting to reduce the solution size of online algorithms.

Figure 12 shows the learning curves over time for the OM2 algorithm for three sparsity levels ($p = 1$, $p = 1.2$, $p = 2$), plotted along with the KP using the best individual kernel (BPN-phog/A180_K20/Level3) and an unweighted sum of kernels (UWS-N) on the normalized data. It can be seen that as with the Caltech101 data set, the performance using the sparse selection of kernels ($p = 1$) or the default (semisparse) setting ($p = 1.2$) is much better than that of the nonsparse selection ($p = 2$). On this data set, however, the best single kernel consistently outperforms the OM2 algorithm.

Figure 13 shows the learning curves over time for the OMKL algorithm for five values of the discount parameter ($\beta = 0.1, \beta = 0.5, \beta = 0.9, \beta = 0.99$, $\beta = 0.999$), plotted along with KP using the best individual kernel (BPN-phog/A180_K20/Level3) and an unweighted sum of kernels (UWS-N) on the normalized data. As with the Caltech101 data set, the performance using the sparsest selection of kernels ($\beta = 0.99$) was the worst, but again the performance is almost indistinguishable among the three settings ($\beta = 0.1$, $\beta = 0.5$, $\beta = 0.9$) that do not enforce sparsity as much. For these three settings of the discount parameter, the OMKL algorithm appears to significantly

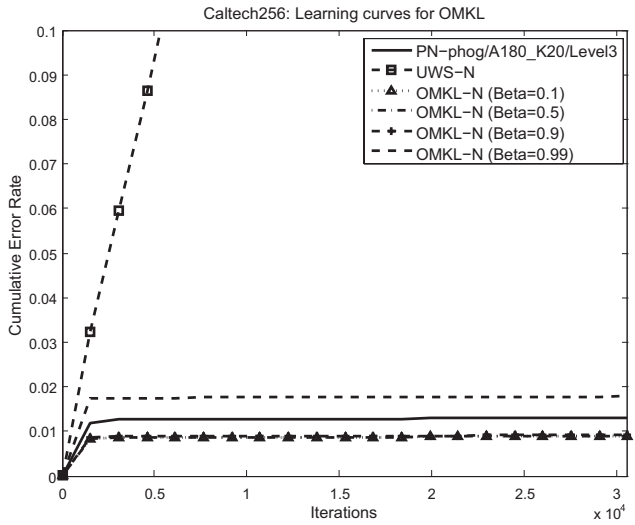

Figure 13: Caltech256: OMKL. Learning curves for the OMKL algorithm for four values of the discount parameter ($\beta = 0.1$, $\beta = 0.5$, $\beta = 0.9$, $\beta = 0.99$), plotted along with KP using the best individual kernel (BPN-phog/A180_K20/Level3) and an unweighted sum of kernels (UWS-N) on the normalized data.

outperform KP using the best individual kernel (which of course could not be selected a priori).

5 Conclusion

It should be noted that some of the algorithms, such as the kernel perceptron and its derivatives, are extremely simple to implement and widely available, whereas the GP-based algorithms (the IVM, PASS-GP) are much more complicated to implement (requiring, for example, expectation propagation solvers) and are publicly available only in specific languages. The OM2 algorithm is contained within the DOGMA toolbox for Matlab (Orabona, 2009), as well as other algorithms such as passive-aggressive, ALMA, NORMA, SILK, projectron, RBP, and banditron. We implemented the OMKL algorithm (Jin et al., 2010), which was previously unavailable, within the framework of the DOGMA toolbox.

One of the main issues for all of the online algorithms was that of computational complexity and how the algorithms try to overcome this using sparsity. For the margin-based algorithms, the computational time cost is dominated by kernel evaluations, as the update equations are in closed

form and are cheap to compute, while the memory cost is linear in the size of the active set.[11] Note that in the standard form, the KP has an unbounded active set size and therefore for noisy problems could prove to be extremely inefficient. The randomized budget perceptron (RBP) is the simplest way to control this, but if used indiscriminantly, it can lead to extremely poor performance. For the GP-based algorithms, the computational time cost is dominated by the EP updates. This is mitigated against through the use of mini-batches—performing EP updates only once a set number of data points has been observed—but this is at the expense of accuracy and there is no clear way of setting the batch size.

Generally the GP-based algorithms achieve a high degree of accuracy (even using only an unweighted sum of kernels) and provide additional information (in terms of posterior variances), but they are not feasible in the "true" online setting or for massive data sets. In this situation, it would be much more sensible to choose one of the margin-based methods. Given mutliple sets of features, and hence multiple kernels, using a KP with an unweighted sum results in classification performance that is better than if the best kernel had been known a priori, and would make a sensible choice because there are effectively no parameters to tune, apart from the maximum active set size if the RBP variant is being used. There is little to distinguish in the performance of the two MKL algorithms, OM2, and OMKL. Both have a parameter to control the computational complexity (p, which is the regularizer norm for OM2 and the discount parameter β for OMKL). OMKL seems to be less sensitive to the particular setting of this parameter than OM2 and had results with lower variance. However, OMKL is not actually a true MKL algorithm (it is really a classifier combination scheme) and hence has (slightly) larger memory requirements. Both come with guarantees in terms of regret bounds.

The OMKL algorithm (Jin et al., 2010) was created by combining the Hedge algorithm (Freund & Schapire, 1997; Vovk, 1998) with the KP. This can of course be applied to the the online SVM of Bordes et al. (2005) to create another online multiple kernel classifier or, for that matter, any other online classification algorithm. It would also be interesting to see if any such algorithms resulted in tighter regret bounds.

One promising line of research is the online (or stochastic) expectation maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977; Sato, 2000; Cappé & Moulines, 2009), also known as quasi–Monte Carlo (see Jank, 2006, for a review), which could, for example be applied to probabilistic multiple kernel learning algorithms Damoulas and Girolami (2008, 2009). The main problem with this approach is likely to be poor performance for the initial

[11] This is assuming that the active set is stored in primal form. The kernel evaluations can be stored, which trades memory storage for time complexity.

learning phase, as there are many parameters to estimate from very few data points, but for large data sets, this may not be an issue.

Another related approach would be to try to create an online version of multiple gaussian process models (Girolami & Rogers, 2005) using similar ideas to that of PASS-GP (Henao & Winther, 2010). For applications where computational resources are not an issue, this might allow the harnessing of the power of the GP framework with the MKL framework to produce highly accurate online classifiers with associated probabilistic outputs.

Finally, there seems to be scope to convert the simple nonparametric kernel learning (Zhuang et al., 2009) approach to the online framework, although it would require building a graph Laplacian online as well as the classifier. There is some work on online learning over graphs (Herbster, Pontil, & Wainer, 2005), but it is generally assumed that the graph is given rather than constructed online.

Appendix: Additional Figures

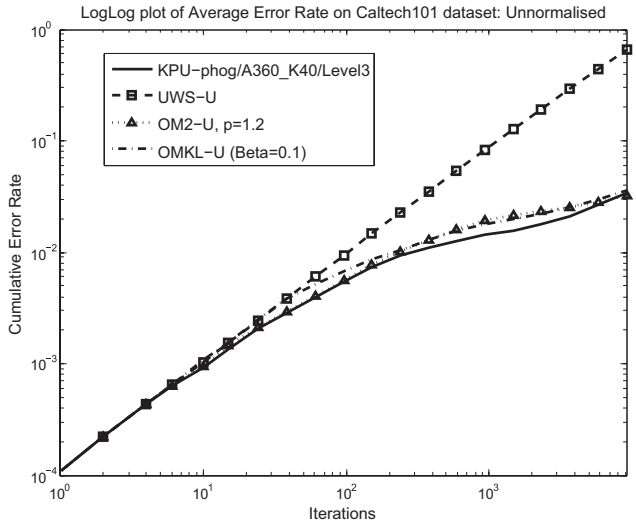

Figure 14: Caltech101 (unnormalized). Learning curves (log-scaled) for the KP with the best individual kernel (KPU-phog/A360_K40/Level3) and an unweighted sum of kernels (UWS-U), the OM2 algorithm for the default sparsity level, and OMKL with $\beta = 0.1$. Note the poor performance in this setting of the unweighted sum and that the MKL algorithms perform only as well as the best individual kernel.

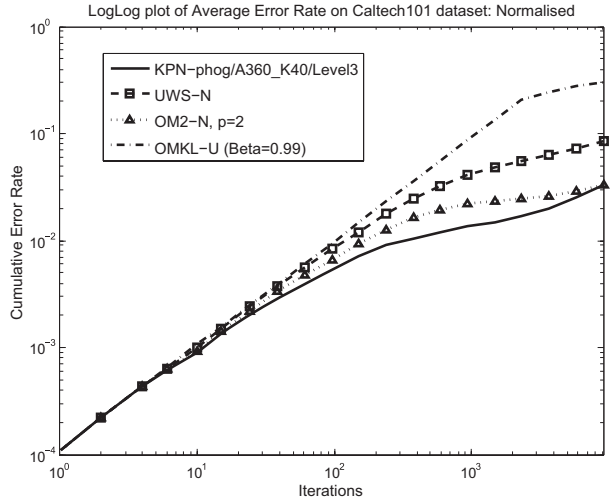

Figure 15: Caltech101 (normalized). Learning curves (log-scaled) for the KP with the best individual kernel (KPN-phog/A360_K40/Level3), and an unweighted sum of kernels (UWS-N), the OM2 algorithm with $p = 2$, and OMKL with $\beta = 0.1$. This normalization method is not particularly effective, and the results are therefore similar to Figure 15.

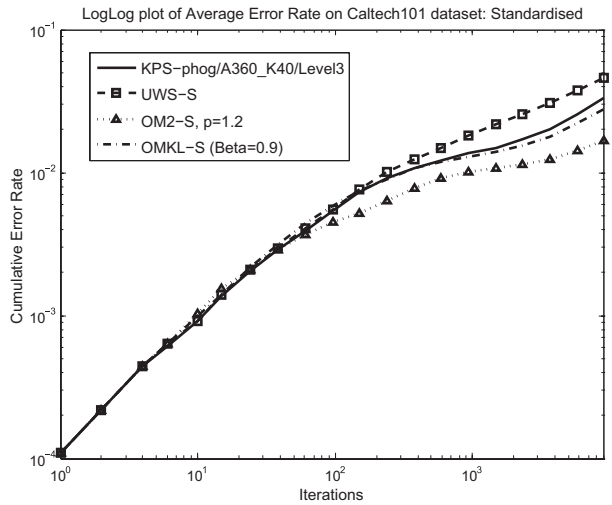

Figure 16: Caltech101 (standardized). Learning curves (log-scaled) for the KP with the best individual kernel (KPS-phog/A360_K40/Level3) and an unweighted sum of kernels (UWS-S), the OM2 algorithm for the default sparsity level, and OMKL with $\beta = 0.9$.

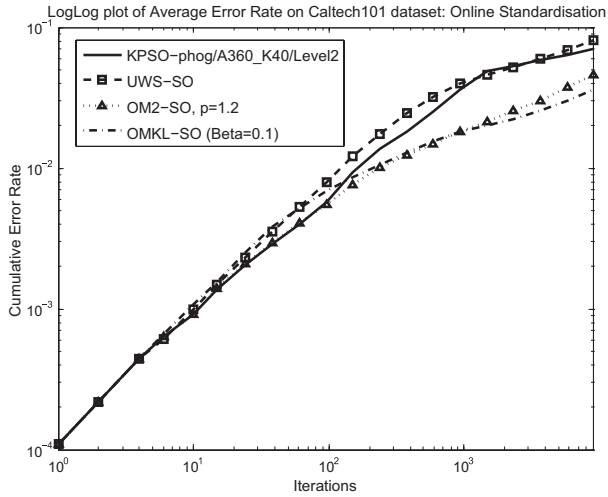

Figure 17: Caltech101 (online standardization). Learning curves (log scaled) for the KP with the best individual kernel (KPSO-phog/A360_K40/Level3) and an unweighted sum of kernels (UWS-SO), the OM2 algorithm for the default sparsity level, and OMKL with $\beta = 0.1$. For this data set, this standardization method is again not as effective, leading to results similar to the normalized setting (see Figure 15).

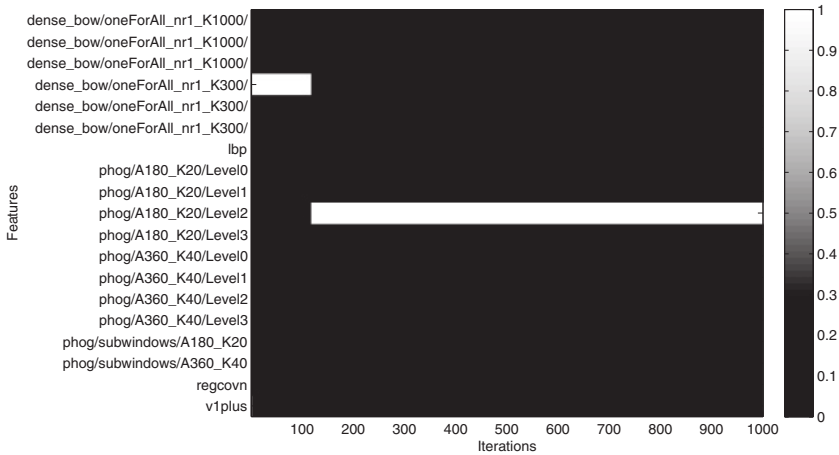

Figure 18: Caltech101: OM2 kernel weights ($p = 1$). Kernel weights over the first 1000 iterations on the Caltech101 data set using example-by-example standardization.

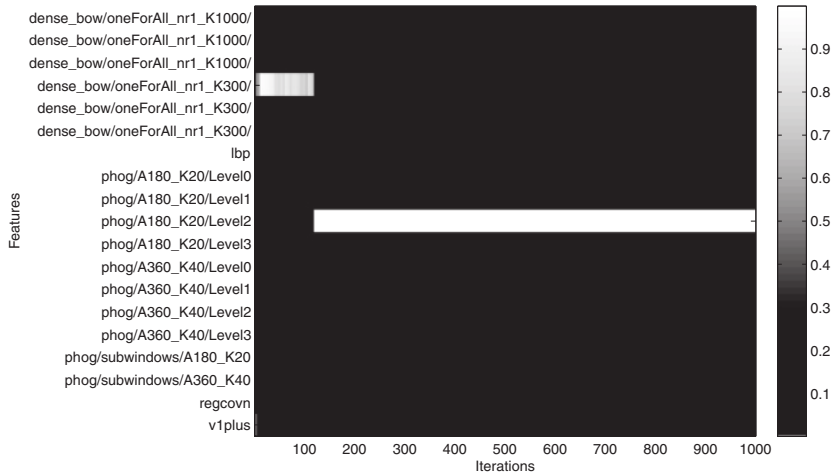

Figure 19: Caltech101: OM2 kernel weights ($p = 1.2$). Kernel weights over the first 1000 iterations on the Caltech101 data set using example-by-example standardization.

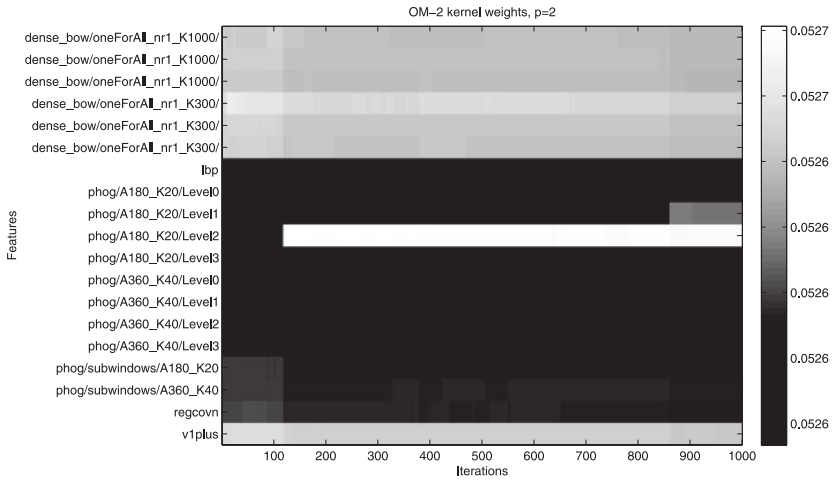

Figure 20: Caltech101: OM2 kernel weights ($p = 2$). Kernel weights over the first 1000 iterations on the Caltech101 data set using example-by-example standardization. Note that the scale is much finer than the two previous plots.

Acknowledgments

This work is supported by NCR Financial Solutions Group under the project title "Developments of Multiple Kernel Learning, Algorithmic Efficiency &

Adaptation for Produce Recognition." We are grateful to M. Filippone for his comments and suggestions.

References

Anlauf, J. K., & Biehl, M. (1989). The adatron: An adaptive perceptron algorithm. *Europhys. Letters*, *10*, 687–692.

Bach, F. R. (2008). Consistency of the group lasso and multiple kernel learning. *J. Mach. Learn. Res.*, *9*, 1179–1225.

Bach, F. R., Lanckriet, G. R. G., & Jordan, M. I. (2004). Multiple kernel learning, conic duality, and the SMO algorithm. In *ICML '04: Proceedings of the Twenty-First International Conference on Machine Learning* (p. 6). New York: ACM.

Block, H. D. (1962). The perceptron: A model for brain functionning. *Reviews of Modern Physics*, *34*, 123–135.

Bordes, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, *6*, 1579–1619.

Bosch, A., Zisserman, A., & Munoz, X. (2007a). Image classification using random forests and ferns. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1–8). Piscataway, NJ: IEEE.

Bosch, A., Zisserman, A., & Munoz, X. (2007b). Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval* (pp. 401–408). New York: ACM.

Bosch, A., Zisserman, A., & Munoz, X. (2008). Image classification using ROIs and multiple kernel learning. *International Journal of Computer*, *4*, 1–25.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Cappé, O., & Moulines, E. (2009). Online EM algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *71*(3), 593–613.

Cauwenberghs, G., & Poggio, T. (2000). Incremental and decremental support vector machine learning. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems, 13* (pp. 409–415). Cambridge, MA: MIT Press.

Cavallanti, G., Bianchi, N., & Gentile, C. (2007). Tracking the best hyperplane with a simple budget perceptron. *Mach. Learn.*, *69*(2–3), 143–167.

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2004). On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, *50*(9), 2050–2057.

Cheng, L., Vishwanathan, S.V.N., Schuurmans, D., Wang, S., & Caelli, T. (2006). Implicit online learning with kernels. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems, 19* (pp. 249–256). Cambridge, MA: MIT Press.

Collobert, R., & Bengio, S. (2004). Links between perceptrons, MLPs and SVMs. In *Proceedings of the International Conference on Machine Learning*. New York: ACM.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, *7*, 551–585.

Crammer, K., Kandola, J. S., & Singer, Y. (2003). Online classification on a budget. In S. Thrün, L. K. Saul, & B. Schölkopf (Eds.), *Advances in neural information processing systems*, *16*. Cambridge, MA: MIT Press.

Csató, L., & Opper, M. (2002). Sparse on-line gaussian processes. *Neural Computation*, *14*(3), 641–668.

Damoulas, T., & Girolami, M. A. (2008). Probabilistic multi-class multi-kernel learning: On protein fold recognition and remote homology detection. *Bioinformatics*, *24*(10), 1264–1270.

Damoulas, T., & Girolami, M. A. (2009). Combining feature spaces for classification. *Pattern Recognition*, *42*(11), 2671–2683.

Dekel, O., Shwartz, S., & Singer, Y. (2008). The forgetron: A kernel-based perceptron on a budget. *SIAM J. Comput.*, *37*, 1342–1372.

Demiriz, A., Bennett, K. P., & Shawe-Taylor, J. (2002). Linear programming boosting via column generation. *Journal of Machine Learning Research*, *46*(13), 225–254.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, *39*(1), 1–22.

Ding, C., & Dubchak, I. (2001). Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, *17*(4), 349–358.

Doucet, A., De Freitas, N., & Gordon, N. (Eds.). (2001). *Sequential Monte Carlo methods in practice*. New York: Springer.

Dubchak, I., Muchnik, I., Holbrook, S., & Kim, S. (1995). Prediction of protein folding class using global description of amino acid sequence. *PNAS*, *92*(19), 8700–8704.

Duchi, J. C., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*, 2121–2159.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New York: Wiley.

Engel, Y., Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with gaussian processes. In *Proc. of the 22nd International Conference on Machine Learning* (pp. 201–208). New York: ACM.

Fei-Fei, L., Fergus, R., & Perona, P. (2004). Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. In *Proceedings of the 2004, Workshop on Generative-Model Based Vision*. Piscataway, NJ: IEEE.

Fei-Fei, L., Fergus, R., & Perona, P. (2006). One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(4), 594–611.

Fink, M., & Singer, Y. (2006). Online multiclass learning by interclass hypothesis sharing. In *Proc. 23rd International Conference on Machine Learning*. New York: ACM.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*, 119–139.

Freund, Y., & Schapire, R. (1999). Large margin classification using the perceptron algorithm. *Mach. Learn.*, *37*(3), 277–296.

Friess, T.-T., Cristianini, N., & Campbell, C. (1998). The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In *15th Intl. Conf. Machine Learning*. San Francisco: Morgan Kaufmann.

Gehler, P. V., & Nowozin, S. (2009). On feature combination for multiclass object classification. In *IEEE International Conference on Computer Vision (ICCV)*. Piscataway, NJ: IEEE.

Girolami, M., & Rogers, S. (2005). Hierarchic Bayesian models for kernel learning. In *Proceedings of the 22nd International Conference on Machine Learning* (pp. 241–248). New York: ACM.

Gomes, R., & Krause, A. (2010). Budgeted nonparametric learning from data streams. In J. Fürnkranz & T. Joachims (Eds.), *Proceedings of the International Conference on Machine Learning* (pp. 391–398). Madison, WI: Omnipress.

Gönen, M., & Alpaydin, E. (2011). Multiple kernel learning algorithms. *Journal of Machine Learning Research*, *12*, 2211–2268.

Griffin, G., Holub, A., & Perona, P. (2007). *The Caltech-256 Object Category Dataset* (Tech. Rep. 7694). Pasadena, CA: Caltech.

Heidrich-Meisner, V., & Igel, C. (2009). Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In A. P. Danyluk, L. Bottou, & M. L. Littman (Eds.), *ICML* (p. 51). New York: ACM.

Helmbold, D. P., & Warmuth, M. K. (1995). On weak learning. *J. Comput. Syst. Sci.*, *50*(3), 551–573.

Henao, R., & Winther, O. (2010). PASS-GP: Predictive active set selection for gaussian processes. In *2010 IEEE International Workshop on Machine Learning for Signal Processing*. Piscataway, NJ: IEEE.

Herbster, M. (2001). Learning additive models online with fast evaluating kernels. In D. P. Helmbold & B. Williamson (Eds.), *Proceedings of COLT/EuroCOLT* (pp. 444–460). New York: Springer.

Herbster, M., Pontil, M., & Wainer, L. (2005). Online learning over graphs. In *Proc. 22nd Int. Conf. Machine Learning* (pp. 305–312). New York: ACM Press.

Herbster, M., & Warmuth, M. (1998). Tracking the best expert. *Machine Learning*, *32*(2), 151–178.

Herbster, M., & Warmuth, M. (2001). Tracking the best linear predictor. *J. Mach. Learn. Res.*, *1*, 281–309.

Hoi, S. C., Jin, R., & Lyu, M. R. (2007). Learning nonparametric kernel matrices from pairwise constraints. In *Proceedings of the 24th International Conference on Machine Learning (ICML2007)*. New York: ACM.

Hussain, Z., & Shawe-Taylor, J. (2010). *Metric learning analysis*. London: University College London.

Jank, W. (2006). The EM algorithm, its stochastic implementation and global optimization: Some challenges and opportunities for OR. In F. Alt, M. Fu, & B. Golden (Eds.), *Topics in modeling, optimization, and decision technologies: Honoring Saul Gass' contributions to operations research* (pp. 367–392). New York: Springer-Verlag.

Jin, R., Hoi, S.C.H., & Yang, T. (2010). Online multiple kernel learning: Algorithms and mistake bounds. M. Hutter, F. Stephan, V. Vovk, & T. Zeugmann (Eds.), *Proceedings of the 21st International Conference on Algorithmic Learning Theory* (pp. 390–404). Berlin: Springer.

Kakade, S., Shalev-Shwartz, S., & Tewari, A. (2009). *On the duality of strong convexity and strong smoothness: Learning applications and matrix regularization* (Tech. Rep.). Chicago: Toyota Technological Institute.

Kembhavi, A., Siddiquie, B., Miezianko, R., McCloskey, S., & Davis, L. S. (2009). Incremental multiple kernel learning for object recognition. In *ICCV* (pp. 638–645). Piscataway, NJ: IEEE.

Kivinen, J., Smola, A. J., & Williamson, R. C. (2002). Large margin classification for moving targets. In N. Cesa-Bianchi, M. Numao, & R. Reischuk (Eds.), *Proceedings of the International Conference on Algorithmic Learning Theory* (pp. 113–127). Berlin: Springer.

Kloft, M., Brefeld, U., Sonnenburg, S., & Zien, A. (2011). l_p-norm multiple kernel learning. *Journal of Machine Learning Research*, *12*, 953–997.

Kuhn, H., & Tucker, A. (1951). Nonlinear programming. In *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics* (pp. 481–492). Berkeley: University of California.

Kuss, M., Rasmussen, C. E., & Herbrich, R. (2005). Assessing approximate inference for binary gaussian process classification. *Journal of Machine Learning Research*, *6*, 1679–1704.

Lanckriet, G.R.G., Cristianini, N., Bartlett, P. L., Ghaoui, L. E., & Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, *5*, 27–72.

Lawrence, N., Seeger, M., & Herbrich, R. (2002). Fast sparse gaussian process methods: The informative vector machine. In S. Becker, S. Thrün, & K. Obermayer (Eds.), *Advances in neural information processing systems*, *15*. Cambridge, MA: MIT Press.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the 7th IEEE International Conference on Computer Vision* (pp. 1150–1157). Piscataway, NJ: IEEE.

Luo, J., Orabona, F., Fornoni, M., Caputo, B., & Cesa-Bianchi, N. (2010). *Om-2: An online multi-class multi-kernel learning algorithm*. Idiap-RR Idiap-RR-06-2010, Idiap.

Maron, O., & Moore, A. (1994). Hoeffding races: Accelerating model selection search for classification and function approximating. In G. Tesauro, D. S. Toureteky, & T. K. Leen (Eds.), *Advances in neural information processing systems*, 7 (pp. 59–66). Cambridge, MA: MIT Press.

Martins, A.F.T., Smith, N., Xing, E., Aguiar, P., & Figueiredo, M. (2010). Online learning of structured predictors with multiple kernels. In G. Gordon, D. Dunson, & M. Dudík (Eds.), *International Conference on Artificial Intelligence and Statistics (AISTATS'11)*. N.p.

McMahan, H. B., & Streeter, M. J. (2010). Adaptive bound optimization for online convex optimization. In A. T. Kalai & M. Mohri (Eds.), *23rd International Conference on Learning Theory* (pp. 244–256). Madison, WI: Omnipress.

Minka, T., Xiang, R., & Qi, Y. A. (2009). Virtual vector machine for Bayesian online classification. In *Proceedings of the Twenty-Fifth Conference Annual Conference on Uncertainty in Artificial Intelligence* (pp. 411–418). Corvallis, OR: AUAI Press.

Nemirovski, A., Juditsky, A., Lan, G ., & Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, *19*(4), 1574–1609.

Novikoff, A. (1962). On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata* (Vol. 12, pp. 615–622). New York: Brooklyn Research Institute.

Ojala, T., Pietikäinen, M., & Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, *24*, 971–987.

Opper, M. (1998). *A Bayesian approach to on-line learning*. Cambridge: Cambridge University Press.

Opper, M., & Winther, O. (1999). Mean field methods for classification with gaussian processes. In M. S. Kearns, S. A. Solla, & D. A. Cohn (Eds.), *Advances in neural information processing systems*, *11*. Cambridge, MA: MIT Press.

Orabona, F. (2009). *DOGMA: A MATLAB toolbox for Online Learning*. http://dogma.sourceforge.net

Orabona, F., Castellini, C., Caputo, B., Jie, L., & Sandini, G. (2009). On-line independent support vector machines. *Pattern Recognition*, *43*, 1402–1412.

Orabona, F., Keshet, J., & Caputo, B. (2009). Bounded kernel-based online learning. *Journal of Machine Learning Research*, *10*, 2643–2666.

Pinto, N., Cox, D., & DiCarlo, J. (2008). Why is real-world visual object recognition hard? *PLoS Comput. Biol.*, *4*(1), e27.

Rakotomamonjy, A., Bach, F., Canu, S., & Grandvalet, Y. (2008). Simplemkl. *Journal of Machine Learning Research*, *9*, 2491–2521.

Rasmussen, C. E., & Williams, C. K. I. (2005). *Gaussian processes for machine learning (adaptive computation and machine learning)*. Cambridge, MA: MIT Press.

Reisinger, J., Stone, P., & Miikkulainen, R. (2008). Online kernel selection for Bayesian reinforcement learning. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning*. New York: ACM.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408.

Roux, N. L., Manzagol, P.-A., & Bengio, Y. (2007). Topmoumoute online natural gradient algorithm. In J. C. Platt, D. Koller, Y. Singer, & S. T. Roweis (Eds.), *Advances in neural information processing systems*, *20*. Cambridge, MA: MIT Press.

Saffari, A., Godec, M., Pock, T., Leistner, C., & Bischof, H. (2010). Online multiclass LPBoost. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 3570–3577). Piscataway, NJ: MIT Press.

Saffari, A., Leistner, C., Santner, J., Godec, M., & Bischof, H. (2009). On-line random forests. In *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)* (pp. 1393–1400).

Sato, M.-A. (2000). Convergence of on-line EM algorithm. In *Proceedings of the 7th International Conference on Neural Information Processing (ICONIP)* (Vol. 1, pp. 476–481). Korea Advanced Institute of Science and Technology.

Seeger, M., Seeger, M., Williams, C. K. I., Lawrence, N. D., & Dp, S. S. (2003). Fast forward selection to speed up sparse Gaussian process regression. *In Workshop on AI and Statistics, Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*. N. P.: Society for Artificial Intelligence and Statistics.

Shalev-Shwartz, S. (2007). *Online learning: Theory, algorithms, and applications*. Unpublished doctoral dissertation, Hebrew University.

Shalev-Shwartz, S., & Singer, Y. (2007). A primal-dual perspective of online learning algorithms. *Machine Learning*, *69*(2), 115–142.

Shawe-Taylor, J., & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge: Cambridge University Press.

Shen, H.-B., & Chou, K.-C. (2006). Ensemble classifier for protein fold pattern recognition. *Bioinformatics*, *22*(14), 1717–1722.

Sutskever, I. (2009). A simpler unified analysis of budget perceptrons. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 985–992). New York: ACM.

Tong, S., Koller, D., & Kaelbling, P. (2001). Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, *2*, 45–66.

Tuzel, O. (2007). P.: Human detection via classification on riemannian manifolds. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition* (pp. 1–8).

Vapnik, V. (1995). Inductive principles of statistics and learning theory. In P. Smolensky, M. C. Mozer, & D. E. Rumelhart (Eds.), *Mathematical perspectives on neural networks*. Mahwah, NJ: Erlbaum.

Vapnik, V., & Chervonenkis, A. (1964). A note on one class of perceptrons. *Automation and Remote Control*, *25*, 103–109.

Varma, M., & Ray, D. (2007). Learning the discriminative power-invariance trade-off. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1–8). Piscataway, NJ: IEEE.

Vovk, V. (1998). A game of prediction with expert advice. *Journal of Computer and System Sciences*, *56*, 153–173.

Weston, J., Bordes, A., & Bottou, L. (2005). Online (and offline) on an even tighter budget. In R. G. Cowell & Z. Ghahramani (Eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics* (pp. 413–420). N.p.: Society for Artificial Intelligence and Statistics.

Winther, O., & Solla, S. A. (1998). Optimal Bayesian online learning. In K.-Y. N. Wong & D.-Y. Yeung (Eds.), *Theoretical aspects of neural computation*. Berlin: Springer Verlag.

Zhang, T. (2004). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In C. E. Brodley (Ed.), *ICML*, volume 69 of *ACM International Conference Proceeding Series*. ACM.

Zhuang, J., Tsang, I. W., & Hoi, S. C. H. (2009). SimpleNPKL: Simple non-parametric kernel learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (pp. 1273–1280). New York: ACM.

Zien, A., & Ong, C. S. (2007). Multiclass multiple kernel learning. In *Proceedings of the 24th International Conference on Machine Learning* (pp. 1191–1198). New York: ACM.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In T. Fawcett & N. Mishra (Eds.), *ICML* (pp. 928–936). N.p.: AAAI Press.

Received November 21, 2011; accepted September 24, 2012.