MKPM: A multiclass extension to the kernel projection machine

Sylvain Takerkart, Liva Ralaivola

MKPM: a Multiclass extension to the Kernel Projection Machine

Sylvain Takerkart
Institut de Neurosciences Cognitives de la Méditerranée. CNRS - Aix-Marseille Université
31, chemin Joseph Aiguier. 13009 Marseille, France
Sylvain.Takerkart[@]incm.cnrs-mrs.fr

Liva Ralaivola
Laboratoire d'Informatique Fondamentale. Aix-Marseille Université
CMI, 39 rue Frédéric Joliot Curie, 13013 Marseille, France
liva.ralaivola[@]lif.univ-mrs.fr

Abstract

We introduce Multiclass Kernel Projection Machines (MKPM), a new formalism that extends the Kernel Projection Machine framework to the multiclass case. Our formulation is based on the use of output codes and it implements a co-regularization scheme by simultaneously constraining the projection dimensions associated with the individual predictors that constitute the global classifier. In order to solve the optimization problem posed by our formulation, we propose an efficient dynamic programming approach. Numerical simulations conducted on a few pattern recognition problems illustrate the soundness of our approach.

1. Introduction

Many real world problems of pattern recognition pose the question of having at hand efficient and effective methods to tackle multiclass learning problems. In this paper, we propose a very simple yet effective new mutliclass *kernel machine*, meaning that the predictors f we are to consider are functions like

$$f(\mathbf{x}) = f^\top \phi(\mathbf{x}), \tag{1}$$

defined on input space $\mathcal{X}$, where $\phi : \mathbf{x} \mapsto \phi(\mathbf{x}) = k(\cdot, \mathbf{x})$ is the mapping from $\mathcal{X}$ to $\mathcal{H}$ associated with some positive definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ [14]; here $f^\top \phi(\mathbf{x})$ denotes the inner product between $f \in \mathcal{H}$ and $\phi(\mathbf{x}) \in \mathcal{H}$.

Our approach rests on the idea of kernel methods regularized by finite-dimensional projections, methods that we broadly refer to Kernel Projection Machines[1] (KPM) [1, 16, 18]. Given a training sample $Z = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ where the data $\mathbf{x}_n$'s are from some space $\mathcal{X}$ and the targets y_n are in $\mathcal{Y}$, the strategy implemented by these machines is the following. First, compute a family $\Psi = (\psi_d)_{d \geq 1}$ of functions in $\mathcal{H}$ (by, e.g., performing a Kernel principal component analysis of the $\mathbf{x}_n$'s) and its associated nested subspaces $S_0 \subset S_1 \subset \ldots \subset S_d \subset \ldots$, where S_d denotes the space spanned by the first d functions $\psi_1, \ldots, \psi_d$ – with $S_0 = \{\mathbf{0}\}$. Then, for each d, solve the *unregularized* empirical minimization problem

$$\hat{f}_d = \operatorname*{argmin}_{f \in S_d} \frac{1}{N} \sum_{n=1}^{N} \ell(f, \mathbf{x}_n, y_n), \tag{2}$$

where $\ell : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a *loss* function, such that $\ell(f, \mathbf{x}, y)$ accounts for the fact that f does not predicts the correct target y for $\mathbf{x}$. Finally, select one of the $\hat{f}_d$'s according to some criterion: either based on theoretical arguments or based on an estimate of the generalization error computed on a validation set. The rationale that supports the use of unregularized empirical risk minimizer is that the process of considering nested subset of *finite* dimension precisely plays the role of regularization.

The approach that we propose for multiclass classification with KPMs builds on the original idea of KPM and the idea of *output coding* for multiclass classification [5]. The formulation that we propose amounts to a minimization problem where several KPMs have to be learned in parallel and *simultaneously* regularized. This specifically means that 'optimal' projection dimensions should be found for each of the KPM learned and that the total number of dimensions retained has to be controlled, which gives rise to a combinatorial problem. In order to solve this problem, we propose a *dynamic programming* approach that allows us to come up with a very efficient learning procedure.

[1] Here, we pervert the name Kernel Projection Machine as, when introduced in [1], it was specifically associated with the binary classification problem and the hinge loss. We provide the name with a broader meaning.

To summarize, our contribution is threefold: (i) a new model of multiclass prediction based on the idea of KPMs where the regularization parameter is a global dimension parameter, (ii) an efficient dynamic programming strategy to solve the combinatorial problem arising from our framework and (iii) numerical simulations conducted on pattern recognition problems that support the soundness of our approach. We may notice that, besides its conceptual simplicity, our approach makes it possible to compute the whole *regularization path* for free, which renders the validation procedure of the global regularization parameter quite inexpensive.

The paper is organized as follows. Section 2 introduces the notation that is used throughout the paper, briefly recalls the approach of output coding for multiclass problem and presents our new Multiclass Kernel Projection Machine (MKPM). Section 3 discusses several interesting features of our learning procedure, such as large-scale learning, model selection and generalization properties. In Section 4, results from numerical simulations conducted on benchmark datasets are reported as well as very positive results for a task of classifying images from a functional magnetic resonance imaging (fMRI) experiment; all the numerical results support the soundness of our proposed model.

2. Multiclass Kernel Projection Machines

From here on, the following notational conventions hold. For a positive integer N, $[N]$ denotes the set $\{1,\dots,N\}$ and $[N]_0$ the set $\{0,\dots,N\}$. For a vector $\mathbf{x}$, $\mathbf{x}^\top$ denotes its transpose, the same notation being used for matrices.

The input space $\mathcal{X}$ where the data to classify live is provided with a positive kernel function $k : \mathcal{X}\times\mathcal{X}\to\mathbb{R}$ (see, e.g., [14] for a detailed account on kernel functions/kernel machines). The reproducing kernel Hilbert space $\mathcal{H}$ associated with k is such that, $\forall f\in\mathcal{H}$ (note that $\mathcal{H}$ is a functional space):

$$f(\mathbf{x}) = \langle f, k(\cdot,\mathbf{x})\rangle_{\mathcal{H}} = f^\top\phi(\mathbf{x}), \quad \forall \mathbf{x}\in\mathcal{X}, \tag{3}$$

where $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ is the inner product of $\mathcal{H}$ and the notation $f^\top g$ is a shorthand for $\langle f,g\rangle_{\mathcal{H}}$. The mapping $\phi:\mathcal{X}\to\mathcal{H}$ is such that $\phi(\mathbf{x}) = k(\cdot,\mathbf{x})$.

The set of labels is denoted by $\mathcal{Y}$, and, to simplify notation, we further suppose that for a categorization problem with Q classes, the set of labels is $\mathcal{Y} = \{1,\dots,Q\}$. A training set $Z = \{(\mathbf{x}_n, y_n)\}_{n\in[N]}$ is made of N labeled pairs $(\mathbf{x}_n, y_n)$ such that $y_n\in\mathcal{Y}$ is the class of $\mathbf{x}_n\in\mathcal{X}$; as commonly assumed in statistical learning, these pairs are independent realizations of a random variable (X,Y) distributed according to an unknown and fixed distribution D on $\mathcal{Z} := \mathcal{X}\times\mathcal{Y}$.

Given Z, Φ denotes the (possibly infinite dimensional) matrix

$$\Phi = [\phi(\mathbf{x}_1)\cdots\phi(\mathbf{x}_N)] \tag{4}$$

such that the Gram matrix $K = (k(\mathbf{x}_i,\mathbf{x}_j))_{i,j\in[N]}$ of the data with respect to k may be written as

$$K = \Phi^\top\Phi. \tag{5}$$

Finally we will consider loss functions $\ell : \mathcal{H}\times\mathcal{X}\times\mathcal{Y}\to\mathbb{R}_+$ that are *convex* in their first argument. An example of such a loss function is the square loss ℓ_{square} defined as:

$$\ell_{\text{square}}(f,\mathbf{x},y) = \left(f^\top\phi(\mathbf{x}) - y\right)^2 = \left(f(\mathbf{x}) - y\right)^2, \tag{6}$$

which we will subsequently make extensive use of.

2.1. Multiclass Learning with Output Codes

Multiclass prediction is a machine learning problem on it own, which raises questions related to modelling, as well as statistical and algorithmic issues. A very powerful and effective way to address this problem is to make use of output codes, which is thoroughly described in [5]. Output coding consists in associating a unique binary code $\mathbf{c}_q\in\{-1,1\}^L$ of length L to each category q of $\mathcal{Y}$. Given these codes, the traning set can be rewritten as $Z = \{(\mathbf{x}_n,\mathbf{y}_n)\}_{n\in[N]}$, where $\mathbf{y}_n\in\{-1,1\}^L$ is one of the Q codewords $\mathbf{c}_q$. The initial learning problem is then converted to L learning problems defined with respect to the sequence of training sets

$$Z_l = \{(\mathbf{x}_n, y_{nl})\}_{n\in[N]}, \quad l\in[L], \tag{7}$$

where, with a slight abuse of notation, we have reused the notation y and where $y_{nl}\in\{-1,1\}$ is the l-th entry of $\mathbf{y}_n$.

Given these L training sets, a usual strategy is to *independently* learn L predictors $f_1,\dots,f_L$ such that each $f_l : \mathcal{X}\to\mathbb{R}$ is inferred from Z_l. The prediction $\hat{y}(\mathbf{x})$ for a new instance $\mathbf{x}$ is then computed from the vector $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x})\cdots f_L(\mathbf{x})]^\top$ as

$$\hat{y}(\mathbf{x}) = \operatorname*{argmin}_{q\in[Q]} \operatorname{dist}(\mathbf{f}(\mathbf{x}),\mathbf{c}_q), \tag{8}$$

where $\operatorname{dist} : \mathbb{R}^L\times\mathbb{R}^L\to\mathbb{R}$ is some predefined divergence measure; for instance, this divergence measure may be the usual Euclidean distance, which we make use of later on.

The approach we propose builds on the framework of output codes and differs from the usual learning strategy in that the L classifiers inferred are simultaneously regularized or, in other words, *co-regularized*. This essentially means that their learning are *no longer* independent.

2.2. Proposed Learning Model

As briefly recalled in the introduction, Kernel Projection Machines work in the following way. Given an integer D and a family $\Psi = (\psi_d)_{d\geq 1}$ of linearly independent functions of $\mathcal{H}$, defining $S_d = \operatorname{span}(\psi_1,\dots,\psi_d)$ as the space spanned by the first d vectors of Ψ (with $S_0 = \{\mathbf{0}\}$), KPMs implement a two-step strategy:

1. For each dimension $d \in [D]_0$, solve the empirical risk minimization problem

$$\hat{f}_d = \operatorname*{argmin}_{f \in S_d} \frac{1}{N} \sum_{n \in [N]} \ell(f, \mathbf{x}_n, y_n). \tag{9}$$

2. Pick the optimal dimension $\hat{d}$ and corresponding classifier $\hat{f}_{\hat{d}}$ according to some model selection criterion.

As stated above, ℓ is a convex loss function. For classification as well as regression, it has been showed that constraining the search of a function f in a finite-dimensional subspace S_d incurs some kind of regularization and that the pivotal parameter of regularization is precisely d [1, 16]. This explains why there is no explicit regularization term in the empirical risk functional of (9).

The approach that we propose pushes the idea of KPM learning one step further. Namely, if we assume the same family Ψ as before and an output coding scheme with code length L for the multiclass learning problem. The two-step learning process based on the training sample $Z = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n \in [N]}$ is the following.

1. For each $d \in [D]_0$, solve the empirical minimization problem

$$\hat{\boldsymbol{\delta}}(d) = \operatorname*{argmin}_{\substack{d_1, \ldots, d_L \in [d]_0 \\ \sum_{l \in [L]} d_l = d}} R(d_1, \ldots, d_L) \tag{10}$$

where the *empirical risk* $R(d_1, \ldots, d_L)$ is defined as

$$R(d_1, \ldots, d_L) := \min_{f_l \in S_{d_l}} \frac{1}{N} \sum_{\substack{n \in [N] \\ l \in [L]}} \ell(f_l, \mathbf{x}_n, y_{nl}). \tag{11}$$

2. Pick the (global) dimension $\hat{d}$, the corresponding vector $\hat{\boldsymbol{\delta}}(\hat{d}) = [\hat{\delta}_1(\hat{d}) \cdots, \hat{\delta}_L(\hat{d})]$ of dimension L and the corresponding L predictors $\hat{f}_{1,\hat{d}}, \ldots, \hat{f}_{L,\hat{d}}$ that realize $R(\hat{\boldsymbol{\delta}}(\hat{d}))$ according to some model selection criterion.

We now elaborate more precisely on how the first step of the procedure is performed: this comes down to studying how Equations (10) and (11) interplay with each other. As for the second step, we will assume that estimates of the generelization performances achieved for models associated with different values of d are computed on an held-out validation set or *via* a cross-validation procedure.

Viewing (11) as a sum of precomputed terms. First note that the objective function of minimization problem (11) that defines R may be rewritten as:

$$\frac{1}{N} \sum_{\substack{n \in [N] \\ l \in [L]}} \ell(f_l, \mathbf{x}_n, y_{nl}) = \sum_{l \in [L]} G_l(f_l), \tag{12}$$

where

$$G_l(f) := \frac{1}{N} \sum_{n \in [N]} \ell(f, \mathbf{x}_n, y_{nl}). \tag{13}$$

Hence, according to (11) and (12), the convex minimization problem that defines R has an objective that is the *sum* of L convex functions (the G_l's) defined on independent variables (the f_l's). In order to evaluate $R(d_1, \ldots, d_L)$ for some $d_1, \ldots, d_L$, it therefore suffices to independently solve the L convex minimization problems

$$G_l^*(d_l) := \min_{f \in S_{d_l}} G_l(f), \quad l \in [L], \tag{14}$$

where the $G_l^*(d_l)$ are uniquely defined since the G_l's are convex. In other words,

$$R(d_1, \ldots, d_L) = \sum_{l \in [L]} \min_{f \in S_{d_l}} G_l(f, Z_l) \tag{15}$$

$$= \frac{1}{L} \sum_{l \in [L]} G_l^*(d_l), \tag{16}$$

and, if all the $G_l^*(d)$ for $l \in [L]$ and $d \in [D]$ are computed beforehand, i.e., each problem (14) is solved for all values of l and d_l, $R(d_1, \ldots, d_L)$ is merely a sum of L precomputed terms.

Problem (10) and dynamic programming. This view of R as a sum of L terms, makes it easier to observe that the problem posed by our learning strategy is a combinatorial minimization problem. Indeed, problem (10) rewrites as:

$$\boldsymbol{\delta}(d) = \operatorname*{argmin}_{\substack{d_1, \ldots, d_L \in [d]_0 \\ \sum_{l \in [\lambda]} d_l = d}} \sum_{l \in [L]} G_l^*(d_l). \tag{17}$$

A direct enumeration procedure to solve this problem would require to scan all the combinations $d_1, \ldots, d_L \in [d]_0$ of L integers such that $\sum_l d_l = d$, which is obviously prohibitive. Henceforth, a more clever way than enumeration should be devised to solve (17) and it turns out a dynamic programming [15] approach may be implemented to this end.

Indeed, let us consider the following minimization problem

$$\Delta_\lambda(d) := \min_{\substack{d_1, \ldots, d_\lambda \in [d]_0 \\ \sum_{l \in [\lambda]} d_l = d}} \sum_{l \in [\lambda]} G_l^*(d_l). \tag{18}$$

When λ is set to L, then we observe that $\Delta_L(d)$ is directly linked to $\boldsymbol{\delta}(d)$ of (10) and (17) by

$$\Delta_L(d) = R(\hat{\boldsymbol{\delta}}(d)). \tag{19}$$

Therefore, being capable of computing $\Delta_L(d)$ is equivalent to being able to solve (17), and we therefore focus on dealing with (18). We have the following proposition, which gives us the dynamic programming scheme, i.e., a recursive formulation, to efficiently solve (18).

Proposition 1. *If we define the series* $\left(\hat{\Delta}_\lambda(d)\right)_{\substack{\lambda\in[L]\\ d\in[D]_0}}$ *such that*

$$\hat{\Delta}_\lambda(d) := \begin{cases} G_1^*(d) & \text{if } \lambda = 1, \\ \min\limits_{\delta\in[d]_0} \left\{\hat{\Delta}_{\lambda-1}(\delta) + G_\lambda^*(d-\delta)\right\} & \text{otherwise.} \end{cases} \tag{20}$$

then, $\forall \lambda \in [L],\ \forall d \in [D]_0$,

$$\hat{\Delta}_\lambda(d) = \Delta_\lambda(d). \tag{21}$$

Proof. Let us fix $d \in [D]_0$ and proceed by induction on λ. For $\lambda = 1$, Equation (18) gives

$$\Delta_1(d) = \min_{\substack{d_1\in[d]_0\\ \sum_{l\in[1]} d_l = d}} \sum_{l\in[1]} G_l^*(d_l) = G_1^*(d). \tag{22}$$

And, by definition of $(\hat{\Delta}_\lambda(d))$ in (20)

$$\hat{\Delta}_1(d) = G_1^*(d). \tag{23}$$

Hence for $\lambda = 1$ assertion (21) holds.

Assume that (21) is valid for $\lambda - 1$, that is

$$\forall d \in [D]_0,\ \hat{\Delta}_{\lambda-1}(d) = \Delta_{\lambda-1}(d). \tag{24}$$

By construction, it is straightforward to see that $\hat{\Delta}_\lambda(d)$ writes as

$$\hat{\Delta}_\lambda(d) = \sum_{l\in[\lambda]} G_l^*(\hat{d}_l), \tag{25}$$

for some $\hat{d}_1, \ldots, \hat{d}_\lambda$ such that $\sum_{l\in[\lambda]} \hat{d}_l = d$. Hence, according to the definition of $\Delta_\lambda(d)$ (see (18)):

$$\hat{\Delta}_\lambda(d) \geq \Delta_\lambda(d). \tag{26}$$

Conversely, for all $d_1, \ldots, d_\lambda \in [d]_0$ such that $\sum_{l\in[\lambda]} d_l = d$,

$$\begin{aligned} \sum_{l\in[\lambda]} G_l^*(d_l) &= \sum_{l\in[\lambda-1]} G_l^*(d_l) + G_\lambda^*(d_\lambda) && (27)\\ &\geq \Delta_{\lambda-1}(d-d_\lambda) + G_\lambda^*(d_\lambda) && (28)\\ &= \hat{\Delta}_{\lambda-1}(d-d_\lambda) + G_\lambda^*(d_\lambda) && (29)\\ &\geq \min_{d_\lambda\in[d]_0} \left\{\hat{\Delta}_{\lambda-1}(d-d_\lambda) + G_\lambda^*(d_\lambda)\right\} && (30)\\ &= \hat{\Delta}_\lambda(d), && (31) \end{aligned}$$

where (28) comes from (18) and (29) comes from induction assumption (24). Taking the minimum of the left-hand side of (27) with respect to $d_1, \ldots, d_\lambda$ such that $\sum_{l\in[\lambda]} d_l = d$ gives

$$\Delta_\lambda(d) = \min_{\substack{d_1,\ldots,d_\lambda\in[d]_0\\ \sum_{l\in[\lambda]} d_l = d}} \sum_{l\in[\lambda]} G_l^*(d_l) \geq \hat{\Delta}_\lambda(d), \tag{32}$$

which, combined with (26), gives that $\hat{\Delta}_\lambda(d) = \Delta_\lambda(d)$ and finishes the proof. □

Proposition 1 provides us with a way to compute $\Delta_\lambda(d)$ for all $\lambda \in [L]$ and $d \in [D]_0$. As a matter of fact, note that the computation of $\Delta_L(D)$ directly implies the computation of all the values of $\Delta_\lambda(d)$ for all $\lambda \in [L]$ and $d \in [D]_0$. If the $G_\lambda^*(d)$ are precomputed, then the space complexity of evaluating $\Delta_L(D)$ is $O(LD)$ and the time complexity is $O(D^2L)$.

Algorithm 1 summarizes the learning algorithm of MKPMs in the case of a generic convex loss function ℓ and a predefined family $\Psi = (\psi_d)_{d\geq 1}$ of linearly independent functions from $\mathcal{H}$.

Algorithm 1 MKPM with output codes of length L

Require:
$Z = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n\in[N]}$ (training set),
$Z' = \{(\mathbf{x}'_n, \mathbf{y}'_n)\}_{n\in[N']}$ (validation set),
$\Psi = (\psi_d)_{d\geq 1}$, $D \geq 1$

Ensure: $\hat{d}, \hat{d}_1, \ldots, \hat{d}_L \in [D]_0$, $\hat{f}_1, \ldots, \hat{f}_L \in \mathcal{H}$

for $l = 1$ to L **do**
  for $d = 0$ to D **do**
    $G_l^*(d) = \min_{f\in S_d} \frac{1}{N} \sum_{n=1}^N \ell(f^\top \phi(\mathbf{x}_n), y_{nl})$
  end for
end for
for $d = 0$ to D **do**
  $\Delta_1(d) = G_1^*(d)$
end for
for $\lambda = 2$ to L **do**
  $\Delta_\lambda(d) = \min_{\delta\in[d]_0} \{\Delta_{\lambda-1}(\delta) + G_\lambda^*(d-\delta)\}$
end for
Choose and return $\hat{d}$ (and the corresponding $\hat{d}_1, \ldots, \hat{d}_L \in [D]_0$, and $\hat{f}_1, \ldots, \hat{f}_L \in \mathcal{H}$) according to an estimate of the generalization error computed on Z'.

2.3. Least-Squares MKPM and KPCA

So far, we have described our new model with an abstract convex loss function and we have assumed that the family Ψ of vectors from $\mathcal{H}$ was predefined. In this section, we detail the computations that need be implemented for two particular choices for the setting presented in the previous section. Namely, we make use of the square loss function (6) and the family of vectors that we use corresponds to the principal components of the vectors $\phi(\mathbf{x}_1), \ldots \phi(\mathbf{x}_N)$.

Computing the family Ψ with Kernel PCA. Here, we just briefly recall how to perform a kernel PCA. Further details can be found in [13]. The computation of the principal components $\psi_1, \ldots, \psi_D$ starts by observing that they are defined as the eigenvectors of the (empirical) covariance matrix C (we do not consider the problem of centering the data):

$$C = \frac{1}{N} \Phi\Phi^\top. \tag{33}$$

A quick analysis shows that in order to compute these vectors, it suffices to extract the eigenvectors $A = [\mathbf{a}_1 \cdots \mathbf{a}_D]$ of the Gram matrix K, with corresponding strictly positive eigenvalues $\gamma_1, \ldots \gamma_D$, to have

$$\psi_d := \frac{1}{\sqrt{\gamma_d}} \Phi \mathbf{a}_d, \tag{34}$$

or, in matrix form

$$\Psi := \Phi A \sqrt{\Gamma^{-1}}, \tag{35}$$

where Γ is the diagonal matrix with diagonal elements $\gamma_1, \ldots \gamma_D$.

Combining the square loss and the KPCA basis. We now recall how Kernel PCA and least-square regression gently combine to evaluate the values of the coefficients $G_l^*(d)$. Suppose that we are interested in the lth subproblem $Z_l = \{(\mathbf{x}_n, y_{nl})\}_{n \in [N]}$ (among the L possible problems). Searching for $f \in S_d := \text{span}(\psi_1, \ldots, \psi_d)$ that solves the least square problem

$$\min_{f \in S_d} \left\{ G_l(f) := \sum_{n=1}^{N} \left(f^\top \phi(\mathbf{x}_n) - y_{nl} \right)^2 \right\}, \tag{36}$$

may be achieved by parameterizing f as $f = \Psi_{[d]} \boldsymbol{\alpha}$, where $\boldsymbol{\alpha} \in \mathbb{R}^d$ and $\Psi_{[d]}$ is the matrix made of the first d columns of Ψ (cf. (35)), and by noting that G_l can be written in matrix form as:

$$G_l(f) = f^\top \Phi \Phi^\top f - 2 f^\top \Phi \mathbf{y}_1 + \|\mathbf{y}_l\|^2. \tag{37}$$

where $\mathbf{y}_l = [y_{1l} \cdots y_{nl}]^\top$. Thus,

$$G_l(\Psi_{[d]} \boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \Psi_{[d]}^\top \Phi \Phi^\top \Psi_{[d]} \boldsymbol{\alpha} - 2 \boldsymbol{\alpha}^\top \Psi_{[d]}^\top \Phi \mathbf{y}_l + \|\mathbf{y}_l\|^2. \tag{38}$$

According to the definition of Ψ (and thus $\Psi_{[d]}$),

$$\Psi_{[d]}^\top \Phi = \sqrt{\Gamma_{[d]}} A_{[d]}^\top, \tag{39}$$

where $\Gamma_{[d]}$ is the upper left $d \times d$ block of Γ and $A_{[d]}$ is the matrix made of the first d columns of A. This gives

$$G_l(\Psi_{[d]} \boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \Gamma_{[d]} \boldsymbol{\alpha} - 2 \boldsymbol{\alpha}^\top \sqrt{\Gamma_{[d]}} A_{[d]}^\top \mathbf{y}_l + \|\mathbf{y}_l\|^2. \tag{40}$$

Computing the gradient of $G_l(\Psi_{[d]} \boldsymbol{\alpha})$ with respect to $\boldsymbol{\alpha}$ and having it be equal to $\mathbf{0}$ leads to:

$$\boldsymbol{\alpha}^* = \sqrt{\Gamma_{[d]}^{-1}} A_{[d]}^\top \mathbf{y}_l. \tag{41}$$

This is then straightforward to compute $G_l^*(d) = G_l^*(\Psi_{[d]} \boldsymbol{\alpha}^*)$ by

$$G_l^*(d) = -\|A_{[d]}^\top \mathbf{y}_l\|^2 + \|\mathbf{y}_l\|^2 \tag{42}$$

This means that when using the square loss, it suffices to extract the eigenvectors and eigenvalues of K to get a very simple expression for the f_l's and the $G_l^*(d)$. This is the reason why we have chosen to implement this specific strategy in the numerical simulations presented below.

3. Discussion

Here we discuss a few interesting points of our proposed multiclass learning approach.

How does the learning procedure scale? One of the prominent questions related to kernel methods is to know whether they can adapt to large-scale datasets, as such types of datasets are plentiful in realworld applications. In order to answer this question, it is interesting to observe that the bulk of computations for MKPM is in (i) computing the family Ψ (for instance, KPCA has a $O(N^3)$ time complexity) and (ii) evaluating the $G_l^*(d)$'s. As for the first point, many efficient methods exist to (help) perform approximate kernel principal component analysis: among the most popular, we may cite the Nyström method [6, 17], and, the incomplete Gram-Schmidt decomposition [4, 10]. These methods greatly reduce the cost of extracting a family Ψ as they run in $O(v^2 N)$ time where $v \ll N$ is a user defined parameter; meanwhile, the informativeness of the extracted family is almost as good as that of the family of principal components. As for the computations of the $G_l^*(d)$'s, which essentially depends on the largest dimension D to explore, we anticipate that this process is not this critical. Indeed, if the family Ψ is ordered in such a way that the first vectors explain most of the inertia of the $\phi(\mathbf{x}_i)$'s, then a warm-start strategy that uses the solution computed for S_d as a starting point for the optimization procedure when considering S_{d+1} should be very efficient.

What about the regularization path? We have mentioned in several occasions that $\hat{d}$ may be selected according to either theoretically-grounded criteria or empirically based estimates of the generalization error. We would like to elaborate on the empirical approach – the theoretical approach is related to the next question. We actually want to emphasize the fact that with our approach to multiclass Kernel Projection Machines, the *regularization path* is automatically computed at the time of learning. This means that the learning procedure is such that it produces all the multiclass predictors associated with dimensions 1 to D, even when only the predictor associated with D is sought. This comes from the dynamic programming approach described in Proposition 1 and it is a very nice feature of the learning process. It makes it possible to implement model selection procedures based on hold-out sets very easily without having to subsample the space of possible dimensions.

What are the generalization properties of MKPMs? This is a question that goes beyond the scope of this paper. However, building on results presented in [1, 16], we think that our model should inherit the nice adaptive properties of KPMs. On a closely related side, we would like to mention

that the multiclass predictors learned with large values of d exhibited overfitting behaviours (not shown here). This suggests that d is precisely the complexity parameters associated with our model, just as it is the case for classical (i.e. not multiclass) KPMs.

4. Numerical results

For all numerical experiments described in this section, the model selection criterion used in the learning phase was to choose the value $\hat{d}$ that minimized the classification error rate on the training data. If a leave-one-out cross-validation was performed, this value was therefore different for each fold/iteration. Furthermore, we used output codes of length $L = Q$, where $c_q(l) = +1$ if $q = l$ and $c_q(l) = -1$ if $q \neq l$. Finally, we used the Euclidean distance in the prediction rule defined in (8). The criterion for measuring the performances of an algorithm was the rate of correct classification.

4.1. Benchmark datasets

We first applied our algorithm on three multiclass datasets issued from the Statlog database, the *dna*, *segment* and *satimage* datasets (downloaded from the *libsvm* website [2]), chosen because they are of medium size, thus limiting computing time, and because they were also included in the benchmarking study [9]. In order to compare our results with the ones from this study, we limited ourselves to using the same RBF kernel $K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$, with different values of γ around the ones that were reported to be optimal γ_o; namely, we used $2^{-v}\gamma_o$, with $v \in \{-2, -1, 0, 1, 2\}$. We arbitrarily decided to use $N = 1500$ to further limit computing time.

The results of the simulations are reported in Table 1, where the last column represents the median performance of the five Multiclass SVM methods evaluated in [9]. Overall, the classification rates obtained with our MKPM model were equivalent to the SVM performances, always for $v = 0$. Those performances might be improved by using more appropriate and possibly continuous output codes [3, 5] but the room for improvement on the *segment* and *dna* problems is potentially very small: i) as just noted, the results are comparable with those of SVM and ii) those SVM results have been obtained thanks to a very thorough validation procedure to choose the hyperparameters and they might be looked at as 'near optimal'. Finally, note that for the *satimage* dataset, the chosen $\hat{d}$ was very close to N, suggesting that considering higher values of N might entail better classification rates.

4.2. Functional MRI data

We now focus on one particular problem that can be formulated as a multiclass learning problem: the classification of functional Magnetic Resonance Imaging (fMRI) spatio-temporal data. This problem is sometimes refered to as "brain-reading" since it consists in trying to infer, from the spatial pattern of brain activity, what the subject was doing at a certain point in time, among a number of choices (the classes) defined by the experimental design. We prefer the more "down-to-earth" Multi-Voxel Pattern Analysis appelation (MVPA, see a review [11]); indeed it calls for using pattern recognition techniques to analyse fMRI data in a multivariate (multi-voxels) framework, thus possibly revealing effects that would not be detected with a standard univariate (in the sense that each voxel is processed independently) SPM-like approach [7]. Despite the exponentially growing number of studies using this framework, KPMs have never been used to analyse fMRI data to our knowledge.

We used fMRI data from [8], which is the study that originally raised the interest of the neuroimaging community towards these approaches. The data of one of the six subjects that were scanned is available online at `www.pni.princeton.edu/mvpa/downloads/nifti_set.tar.gz`. It contains ten acquisition sessions, each with examples of the eight categories of visual stimuli that were presented to the subject, which define eight classes. The inputs are points in a 577-dimensional space (the number of voxels in a cortical region defined anatomically, that was of particular interest for the authors; note that no feature selection was performed), whose coordinates are the raw activation values measured at single time-points (no temporal averaging was performed). We fed those examples both to our MKPM model and to a linear SVM (which is one of the algorithms giving the best performing according to the comparison study [12]), and performed a leave-one-session-out cross-validation to estimate the generalization capability of both methods.

In practice, we used the implementation of the linear SVM available in libsvm ([2]), which uses a "one-against-one" method for the multiclass problem. We ran it with $C = 10^{-t}$ (t integer so that $-6 < t < 4$) and obtained the best performances for $t = -2$ (classification rate = 0.681). To test our MKPM model, we used $N = 577$ (the number of features) and tried different standard kernels (linear, polynomial of different orders, and RBF with different γ). The best performances were obtained with the linear kernel (classification rate = 0.734), confirming that it is the most adapted for fMRI data; it is also to be noted that the order $\hat{d}$ of the selected model varied significantely across the iterations of the cross-validation (when using the linear kernel, it varied between 303 and 431, with a mean value of 390). We believe that our model outperforms the "one-against-one" SVM mostly because it relies on a true multiclass formulation; this should be tested on one hand with a multiclass SVM and on the other hand on different datasets, because if confirmed, the superiority of multiclass models would be

Dataset	N	$\hat{d}$	Kernel	MKPM	SVM
segment	1500	451	$\gamma = 2^{-3}$	0.956	0.974
dna	1500	690	$\gamma = 2^{-6}$	0.957	0.956
satimage	1500	1492	$\gamma = 2^{-0}$	0.907	0.913
fmri	577	390	linear	0.734	0.681

Table 1. Results obtained with our MKPM model and SVM.

an important result for the fMRI field.

5. Conclusion

In this paper, we have proposed MKPM, a multiclass extension of the kernel projection machines. The optimization problem posed by our approach is a combinatorial problem that can be nicely tackled by means of a dynamic programming procedure. We have conducted several numerical simulation on pattern recognition problems, from which MKPM appears to be very competitive with state-of-the-art methods. It seems that the co-regularization scheme implemented by our strategy is very relevant, but theoretical arguments need still be elaborated to support this feeling. This is going to be the topic of further research.

References

[1] G. Blanchard and L. Zwald. Finite-dimensional projection for classification and statistical learning. *IEEE Trans. on Information Theory*, 54(9):4169 – 4182, 2008. 2785, 2787, 2789

[2] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. 2790

[3] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233, 2002. 2790

[4] N. Cristianini, J. Shawe-Taylor, and H. Lodhi. Latent Semantic Kernels. *Journal of Intelligent Information Systels*, 18(2–3):127–152, 2002. 2789

[5] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995. 2785, 2786, 2790

[6] P. Drineas and M. W. Mahoney. On the nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6(Dec):2153–2175, 2005. 2789

[7] K. J. Friston, A. P. Holmes, K. J. Worsley, J.-P. Poline, C. D. Frith, and R. S. J. Frackowiak. Statistical parametric maps in functional imaging: A general linear approach. *Human Brain Mapping*, 2(4):189–210, 1994. 2790

[8] J. V. Haxby, M. I. Gobbini, M. L. Furey, A. Ishai, J. L. Schouten, and P. Pietrini. Distributed and Overlapping Representations of Faces and Objects in Ventral Temporal Cortex. *Science*, 293(5539):2425–2430, 2001. 2790

[9] C.-W. Hsu and C.-J. Lin. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, 13(2):415 –425, Mar. 2002. 2790

[10] Z. Hussain and J. Shawe-Taylor. Theory of matching pursuit. In *Ad. In Neural Information Processing Systems*, pages 721–728, 2008. 2789

[11] G. J. D. K. A. Norman, S. M. Polyn and J. V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fmri data. *Trends in Cognitive Science*, 10(3):424–430, 2006. 2790

[12] M. Misaki, Y. Kim, P. A. Bandettini, and N. Kriegeskorte. Comparison of multivariate classifiers and response normalizations for pattern-information fmri. *NeuroImage*, 53(1):103 – 118, 2010. 2790

[13] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. 2788

[14] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. 2785, 2786

[15] M. Sniedovich. *Dynamic Programming Foundations and Principles*. Taylor & Francis, 2010. 2787

[16] E. D. Vito, L. Rosasco, A. Caponnetto, U. D. Giovannini, and F. Odone. Learning from examples as an inverse problem. *Journal of Machine Learning Research*, 6:883–904, 2005. 2785, 2787, 2789

[17] C. K. I. Williams and M. Seeger. Using the Nystrm Method to Speed Up Kernel Machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, 2001. 2789

[18] L. Zwald, R. Vert, G. Blanchard, and P. Massart. Kernel projection machine: a new tool for pattern recognition. In *Adv. in Neural Information Processing Systems*, volume 17, 2005. 2785