DIFFEOMORPHIC LEARNING

LAURENT YOUNES

Abstract. We introduce in this paper a learning paradigm in which the training data is transformed by a diffeomorphic transformation before prediction. The learning algorithm minimizes a cost function evaluating the prediction error on the training set penalized by the distance between the diffeomorphism and the identity. The approach borrows ideas from shape analysis, in the way diffeomorphisms are estimated for shape and image alignment, and brings them in a previously unexplored setting, estimating, in particular diffeomorphisms in much larger dimensions. After introducing the concept and describing a learning algorithm, we present diverse applications, mostly with synthetic examples, demonstrating the potential of the approach, as well as some of its current room for improvement.

1. Introduction

We introduce a family of predictors that learns a function of the original data $x \in \mathbb{R}^d$ in the form $x \mapsto F(\phi(x))$ where ϕ is a diffeomorphism of $\mathbb{R}^d$ and F is real-valued, belonging to a class of simple (e.g., linear) predictors. To simplify the exposition, we will consider only classification problems (and therefore speak of classifiers instead of predictors) and let c denote the number of classes, so that F takes values in $C = \{0, \ldots, c-1\}$, or (since we will use logistic regression) in the space of probability distributions on C. Both ϕ and F are parametrized, the parameter space for ϕ being (before discretization) infinite dimensional while F will only depend on a small number of parameters, that will be denoted by $\theta \in \mathbb{R}^q$, and we will write $F(x;\theta)$ instead of $F(x)$ when needed.

Assume that a training set is given, in the form $\mathcal{T}_0 = (x_1, y_1, \ldots, x_N, y_N)$ with $x_k \in \mathbb{R}^d$ and $y_k \in C$ for $k = 1, \ldots, N$. We will assume that this set is not redundant, i.e., that $x_i \neq x_j$ whenever $i \neq j$. (Our construction can be extended to redundant training sets by allowing the training class variables y_k to be multi-valued. This would result in rather cumbersome notation that is better avoided.) If ϕ is a diffeomorphism of $\mathbb{R}^d$, we will let $\phi \cdot \mathcal{T}_0 = (\phi(x_1), y_1, \ldots, \phi(x_N), y_N)$. The learning procedure will consist in minimizing the objective functions

$$G(\phi, \theta) = D(\mathrm{id}, \phi)^2 + \lambda \Gamma(F(\cdot, \theta), \phi \cdot \mathcal{T}_0) \tag{1}$$

with respect to ϕ and θ. Here D is a Riemannian distance in a group of diffeomorphisms of $\mathbb{R}^d$, that will be described in the next section. Γ is a standard cost function. For example

$$\Gamma(F(\cdot, \theta), \phi \cdot \mathcal{T}_0) = -\sum_{k=1}^{N} \log F(\phi(x_k); \theta)(y_k)$$

in the case of logistic regression, where

$$F(z, \theta)(y) = \frac{e^{\theta(y)^T z}}{\sum_{y' \in C} e^{\theta(y')^T z}}$$

Date: June 5, 2018.

and the parameter is $(\theta_1, \ldots, \theta_{c-1}) \in (\mathbb{R}^d)^{c-1}$ with $\theta_0 = 0$.

This cost function is similar to that used in the last layer of convolutional neural nets (CNN). We will see later that the proposed construction shares other properties with CNNs, at least on a high level. The approach is also, by many aspects, a kernel method and therefore shares this property with classifiers such as support vector machines (SVMs) or other classification methods that use the "kernel trick." However, the method itself is directly inspired from "diffeomorphic shape analysis," and can be seen as a variant of the "Large Deformation Diffeomorphic Metric Mapping" (LDDMM) algorithm, which has been introduced for image and shape registration [11, 2, 21].

2. Distance on Diffeomorphisms

We now describe the basic framework leading to the definition of Riemannian metrics on groups of diffeomorphisms. Let $\mathbf{B}_p = C_0^p(\mathbb{R}^d, \mathbb{R}^d)$ denote the space of p-times continuously differentiable functions that tend to 0 (together with their first p derivatives) to infinity. This space is a Banach space, for the norm

$$\|v\|_{p,\infty} = \max_{0 \leq k \leq p} \|d^k v\|_\infty$$

where $\|\cdot\|_\infty$ is the usual supremum norm.

Introduce a Hilbert space V of vector fields on $\mathbb{R}^d$ which continuously embedded in $\mathbf{B}_p$ for some $p \geq 1$, which means that there exists a constant C such that

$$\|v\|_{p,\infty} \leq C\|v\|_V$$

for all v in V, where $\|\cdot\|_V$ denotes the Hilbert norm on V (and we will denote the associated inner product by $\langle \cdot\,, \cdot \rangle_V$). This assumption implies that V is a reproducing kernel Hilbert space (RKHS). Because V is a space of vector fields, the definition of the associated kernel slightly differs from the usual case of scalar valued functions in that the kernel is matrix valued. More precisely, a direct application of Riesz's Hilbert space representation theorem implies that there exists a function

$$K : \mathbb{R}^d \times \mathbb{R}^d \to \mathcal{M}_d(\mathbb{R})$$

where $\mathcal{M}_d(\mathbb{R})$ is the space of d by d real matrices, such that

(1) The vector field $K(\cdot, y)a : x \mapsto K(x, y)a$ belongs to V for all $y, a \in \mathbb{R}^d$.
(2) For $v \in V$, for all $y, a \in \mathbb{R}^d$, $\langle K(\cdot, y)a\,,\, v \rangle_V = a^T v(y)$.

These properties imply that $\langle K(\cdot, x)a\,,\, K(\cdot, y)b \rangle_V = a^T K(x, y)b$ for all $x, y, a, b \in \mathbb{R}^d$, which in turn implies that $K(y, x) = K(x, y)^T$ for all $x, y \in \mathbb{R}^d$.

Diffeomorphisms can be generated as flows of ordinary differential equations (ODEs) associated with time-dependent elements of V. More precisely, let $v \in L^2([0, 1], V)$, i.e., $v(t) \in V$ for $t \in [0, 1]$ and

$$\int_0^1 \|v(t)\|_V^2 \, dt < \infty.$$

Then, the ODE $\partial_t y = v(t, y)$ has a unique solution over $[0, 1]$ given any initial condition $y(0) = x$ the flow associated with this ODE is the function $\phi^v : (t, x) \mapsto y(t)$ where y is the solution starting at x. This flow is, at all times, a diffeomorphism of $\mathbb{R}^d$, and satisfies the equation $\partial_t \phi = v(t) \circ \phi$, $\phi(0) = \mathrm{id}$ (the identity map). The set of diffeomorphisms that can be generated in such a way form a group, denoted Diff_V since it depends on V. Given $\phi_1 \in \mathrm{Diff}_V$, one defines the optimal deformation cost $\Gamma(\phi_1)$ from id to ϕ_1 as the minimum of $\int_0^1 \|v(t)\|_V^2 \, dt$ over all $v \in L^2([0, 1], V)$ such that $\phi^v(1) = \phi_1$. If we let $D(\phi_1, \phi_2) = \Gamma(\phi_2 \circ \phi_1^{-1})^{1/2}$ then D is a geodesic distance on Diff_V

associated with the right-invariant Riemannian metric generated by $v \mapsto \|v\|_V$ on V. We refer to [21] for more details and additional properties on this construction. For our purposes here, we only need to notice that the minimization of the objective function in (1) can be rewritten as an optimal control problem minimizing

$$E'(v,\theta) = \int_0^1 \|v(t)\|_V^2 \, dt + \lambda \Gamma(F(\cdot,\theta), \phi(1)\cdot \mathcal{T}_0) \tag{2}$$

over $v \in L^2([0,1],V)$, $\theta \in \mathbb{R}^q$ and subject to the constraint that $\phi(t) = \phi(t,\cdot)$ satisfies the equation $\partial_t \phi = v \circ \phi$ with $\phi(0) = \mathrm{id}$. We point out that, under mild regularity conditions on the dependency of Γ with respect to $\mathcal{T}_0$ (continuity in $x_1,\dots,x_N$ suffices), a minimizer of this function in v for fixed θ always exists, with $v \in L^2([0,1],V)$.

3. Reduction

The minimization in (2) can be reduced using an RKHS argument, similar to the "kernel trick" invoked in standard kernel methods [1, 4, 14, 20, 18]. Let $z_k(t) = \phi(t,x_k)$. Because the endpoint cost Γ only depends on $(z_1(1),\dots,z_N(1))$, it only suffices to compute these trajectories, which satisfy $\partial_t z_k = v(t,z_k)$. This implies that an optimal v must be such that, at every time t, $\|v(t)\|_V^2$ is minimal over all $\|w\|_V^2$ with w satisfying $w(z_k) = v(t,z_k)$, which requires $v(t)$ to take the form

$$v(t,\cdot) = \sum_{k=1}^N K(\cdot, z_k(t)) a_k(t) \tag{3}$$

where K is the kernel of the RKHS V and $a_1,\dots,a_N$ are unknown time-dependent vectors in $\mathbb{R}^d$, which provide our reduced variables. Letting $\boldsymbol{a} = (a_1,\dots,a_N)$, the reduced problem requires to minimize

$$E(\boldsymbol{a}(\cdot),\theta) = \int_0^1 \sum_{k,l=1}^N a_k(t)^T K(z_k(t),z_l(t)) a_l(t)\, dt + \lambda \Gamma(F(\cdot,\theta),\mathcal{T}(1)) \tag{4}$$

subject to $\partial_t z_k = \sum_{l=1}^N K(z_k,z_l)a_l$, $z_k(0) = x_k$, with the notation $\mathcal{T}(t) = (z_1(t),\dots,z_N(t))$.

4. Optimality Conditions and Gradient

We now consider the minimization problem with fixed θ (optimization in θ will depend on the specific choice of classifier F and risk function Γ). For the optimal control problem (9), the "state space" is the space in which the "state variable" $\boldsymbol{z} = (z_1,\dots,z_N)$ belongs, and is therefore $Q = (\mathbb{R}^d)^N$. The control space contains the control variable $\boldsymbol{a}$, and is $U = (\mathbb{R}^d)^N$ (even though these spaces are identical, it is conceptually useful to keep separate notation for both).

Optimality conditions for the variable $\boldsymbol{a}$ are provided by Pontryagin's maximum principle (PMP). They require the introduction of a third variable (co-state), denoted $\boldsymbol{p} \in Q$, and of a control-dependent Hamiltonian $H_{\boldsymbol{a}}$ defined on $Q \times Q$ given, in our case, by

$$H_{\boldsymbol{a}}(\boldsymbol{p},\boldsymbol{z}) = \sum_{k,l=1}^N (p_k - a_k)^T K(z_k,z_l) a_l. \tag{5}$$

(In this expression, $\boldsymbol{a}$, $\boldsymbol{p}$ and $\boldsymbol{z}$ do not depend on time.) The PMP [10, 13, 16, 19] then states that any optimal solution $\boldsymbol{a}$ must be such that there exists a time-dependent co-state satisfying

$$\begin{cases} \partial_t \boldsymbol{z} = \partial_{\boldsymbol{p}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \partial_t \boldsymbol{p} = -\partial_{\boldsymbol{z}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \boldsymbol{a}(t) = \mathrm{argmax}_{\boldsymbol{a}'} H_{\boldsymbol{a}'}(\boldsymbol{p}(t), \boldsymbol{z}(t)) \end{cases} \tag{6}$$

with boundary conditions $\boldsymbol{z}(0) = (x_1, \ldots, x_N)$ and

$$\boldsymbol{p}(1) = -\lambda \partial_{\boldsymbol{z}} \Gamma(F(\cdot, \theta), \mathcal{T}(1)). \tag{7}$$

These conditions are closely related to those allowing for the computation of the differential of E with respect to $\boldsymbol{a}(\cdot)$, which is given by

$$\partial_{\boldsymbol{a}(\cdot)} E(\boldsymbol{a}(\cdot), \theta) = \boldsymbol{u}(\cdot)$$

with

$$u_k(t) = \sum_{l=1}^{N} K(z_k(t), z_l(t))(p_l(t) - 2a_l(t))$$

where p solves

$$\begin{cases} \partial_t \boldsymbol{z} = \partial_{\boldsymbol{p}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \partial_t \boldsymbol{p} = -\partial_{\boldsymbol{z}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \end{cases} \tag{8}$$

with boundary conditions $\boldsymbol{z}(0) = (x_1, \ldots, x_N)$ and

$$\boldsymbol{p}(1) = -\lambda \partial_{\boldsymbol{z}} \Gamma(F(\cdot, \theta), \mathcal{T}(1)).$$

Concretely, the gradient is computed by solving the first equation of (10), which does not involve $\boldsymbol{p}$, using the obtained value of $\boldsymbol{z}(1)$ to compute $\boldsymbol{p}(1)$ from the boundary condition, and then solving the second equation of (10) backward in time to obtain $\boldsymbol{p}$ at all times, and plug it in the definition of $\boldsymbol{u}(t)$.

For practical purposes, the discrete time version of the problem is obviously more useful, and its differential is obtained from a similar dynamic programming (or back-propagation) computation. Namely, discretize time over $0, 1, \ldots, T$ and consider the objective function

$$E(\boldsymbol{a}(\cdot), \theta) = \frac{1}{T} \sum_{t=0}^{T-1} \sum_{k,l=1}^{N} a_k(t)^T K(z_k(t), z_l(t)) a_l(t)\, dt + \lambda \Gamma(F(\cdot, \theta), \mathcal{T}(T)) \tag{9}$$

subject to $z_k(t+1) = z_k(t) + \frac{1}{T} \sum_{l=1}^{N} K(z_k(t), z_l(t)) a_l(t)$, $z_k(0) = x_k$. We here use a simple Euler scheme to discretize the state ODE. Note that the state is discretized over $0, \ldots, T$ and the control over $0, \ldots, T-1$. In this case, the differential of E is given by the following expression, very similar to the one obtained in continuous time.

$$\partial_{\boldsymbol{a}(\cdot)} E(\boldsymbol{a}(\cdot), \theta) = \boldsymbol{u}(\cdot)$$

with

$$u_k(t) = \sum_{l=1}^{N} K(z_k(t), z_l(t))(p_l(t) - 2a_l(t)), \quad t = 0, \ldots, T-1$$

where p (discretized over $0, \dots, T-1$), can be computed using

$$\begin{cases} \boldsymbol{z}(t+1) = \boldsymbol{z}(t) + \dfrac{1}{T}\partial_{\boldsymbol{p}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \\ \boldsymbol{p}(t-1) = \boldsymbol{p}(t) + \dfrac{1}{T}\partial_{\boldsymbol{z}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t)) \end{cases} \tag{10}$$

with boundary conditions $\boldsymbol{z}(0) = (x_1, \dots, x_N)$ and

$$\boldsymbol{p}(T-1) = -\lambda \partial_{\boldsymbol{z}} \Gamma(F(\cdot, \theta), \mathcal{T}(T)).$$

We emphasize the fact that we are talking of differential of the objective function rather than its gradient. Our implementation uses a Riemannian (sometimes called "natural") gradient with repect to the metric

$$\left\langle \eta_1(\cdot)\,,\, \eta_2(\cdot) \right\rangle_{\boldsymbol{a}(\cdot)} = \int_0^1 \sum_{k,l=1}^{n} \eta_k(t)^T K(z_k(t), z_l(t)) \eta_l(t) dt$$

with $\partial_t z_k = \sum_{l=1}^n K(z_k(t), z_l(t)) a_l(t)$. With respect to this metric, one has

$$\nabla_{\boldsymbol{a}(\cdot)} E(\boldsymbol{a}(\cdot), \theta) = \boldsymbol{p} - 2\boldsymbol{a},$$

a very simple expression that can also be used in the discrete case.

5. Kernel

To fully specify the method, one needs to choose the RKHS V, or, equivalently, its reproducing kernel, K. It constitutes an important component of the model because it constrains the regularity of the estimated diffeomorphism. We recall that K is a kernel over vector fields, and therefore is matrix valued. One simple way to build such a kernel is to start with a scalar positive kernel $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to R$ and let

$$K(x, y) = \kappa(x, y) \mathrm{Id}_{\mathbb{R}^d}. \tag{11}$$

We will refer to such kernels as "scalars."

One can choose κ from the large collection of known positive kernels (and their infinite number of possible combinations) [1, 7, 18, 3]. Most common options are Gaussian kernels

$$\kappa(x, y) = \exp(-|x-y|^2/2a^2)$$

or Matérn kernels

$$\kappa(x, y) = P_k(|x-y|/a) \exp(-|x-y|/a)$$

where P_k is a reversed Bessel polynomial of order k. In both cases, a is a positive scale parameter. The Matérn kernels have the nice property that their associated RKHS is equivalent to a Sobolev space of order $k + d/2$.

Vector fields v in the RKHS associated with a matrix kernel such as (11) where κ is a radial basis function (RBF) are such that each coordinate function of v belongs to the scalar RKHS associated with κ, which is translation and rotation invariant (i.e., the transformations that associate to a scalar function h the functions $x \mapsto h(R^T(x-b))$ are isometries, for all rotation matrices R and all vectors $b \in \mathbb{R}^d$).

While (11) provide a simple recipe for the definition of matrix-valued kernels, many more interesting choices can be made within this class. Rotation and translation invariance more adapted to spaces of vector fields, in which one requires that replacing $v : \mathbb{R}^d \to \mathbb{R}^d$ by $x \mapsto Rv(R^T(x-b))$ is an isometry of V for all R, b, leads to a more general class of matrix kernels extensively discussed in

[15]. When the data is structured, however (e.g., when it is defined over a grid), rotation invariance may not be a good requirement since it allows, for example, for permuting coordinates, which would break the data structure. In this context, other choices can be made.

Assuming that the data is defined over a graph, say $\mathcal{G}$ with d vertices, one can, for example, define matrix kernels relying on this structure. For example, letting $\mathcal{N}_i$ denote the set of nearest neighbors of i in $\mathcal{G}$, one can simply take

$$K(x,y) = \mathrm{diag}(\Phi(|P_i x - P_i y|), i = 1, \ldots, d) \tag{12}$$

where $P_i x$ is the vector $(x_j, j \in \mathcal{N}_i)$ and Φ is an RBF associated to a positive radial scalar kernel.

6. Algorithm

Everything is now in place to fully describe the basic learning algorithm for the diffeomorphism part of the model, at least up to the computation of the differential of the end-point cost Γ, which depends on the choice made for the final classifier. We summarize it below. In our implementation, we have used a first-degree Matérn kernel. Typically, the kernel is chosen up to a scale coefficient, a, which becomes a parameter of the classifier.

Algorithm 1. Input. *Training set* $\mathcal{T}_0 = (x_1, y_1, \ldots, x_N, y_N)$.
Parameters. *Scale parameter* a*; data weight* λ*; time discretization* T.
Variables: *Coefficients* $\boldsymbol{a} = (a_j(t) \in \mathbb{R}^d, j = 1, \ldots, N, t = 0, \ldots T)$. *Parameter* θ *of the classifier.*
Initialization (step $\tau = 0$**):** *Set* $\boldsymbol{a}_0 = 0$.
Gradient Descent Loop (step τ**):** *Let* $\boldsymbol{a}_\tau$ *and* θ_τ *be the current variables.*

1. *Compute* $\boldsymbol{z}(t), t = 0, \ldots T$ *using* $z_k(t+1) = z_k(t) + \frac{1}{T}\sum_{l=1}^N K(z_k(t), z_l(t))a_l(t)$, $z_k(0) = x_k$. *Let* $T_\tau = (z_1(T), y_1, \ldots, z_N(t), y_N)$.
2. *Learn* θ_τ *based on* T_τ.
2. *Compute* $\boldsymbol{p}(T-1) = -\lambda \partial_{\boldsymbol{z}} \Gamma(F(\cdot, \theta_\tau), \mathcal{T}_\tau)$.
3. *Compute* $\boldsymbol{p}(t), t = T-2, \ldots, 0$ *using* $\boldsymbol{p}(t-1) = \boldsymbol{p}(t) + \frac{1}{T}\partial_{\boldsymbol{z}} H_{\boldsymbol{a}}(t)(\boldsymbol{p}(t), \boldsymbol{z}(t))$ *with* $H_{\boldsymbol{a}}$ *given in* (5).
4. *Use a line search to update* $\boldsymbol{a}_{\tau+1} = \boldsymbol{a}_\tau - \epsilon(\boldsymbol{p} - \boldsymbol{a}_\tau)$.
5. *Test stopping criterion and exit loop if satisfied.*

7. Remarks

7.1. **Adding a dimension.** As stated in the introduction, we use, in our experiments, logistic regression as final classifier applied to transformed data. This classifier estimates a linear separation between the classes, but it should be clear that not every training set can be transformed into a linearly separable one using a diffeomorphism of the ambient space. A very simple one-dimensional example is when the true class associated to an input $x \in \mathbb{R}$ is 0 if $|x| < 1$ and 1 otherwise: no one-dimensional diffeomorphism will make the data separable, since such diffeomorphisms are necessarily monotone. This limitation can be fixed easily, however, by adding a dummy dimension and apply the model to a $(d+1)$-dimensional model in which X is replaced by $(X, 0)$. In the example just mentioned, for example, the transformation $(x, \mu) \mapsto (x, \mu + x^2 - 1)$ is a diffeomorphism of $\mathbb{R}^2$ that separates the two classes (along the y axis).

Notice that any binary classifier that can be expressed as $x \mapsto \mathrm{sign}(f(x) - a)$ for some smooth function f can be included in the model class we are considering after adding a dimension, simply taking (letting again μ denote the additional scalar variable) $\phi(x, \mu) = (x, \mu + f(x))$, which is a

diffeomorphism of $\mathbb{R}^{d+1}$, and $u = (0_{\mathbb{R}^d}, 1)$. Even when it works perfectly on the training set, this transformation will not be optimal in general, and the diffeomorphic classifier would typically prefer a diffeomorphism ϕ that will minimize the overall distortion, essentially trading off some non-linear transformation of the data (x) to induce a "simpler" classification rule.

When adding one or more dimensions (in a c-class problem, it makes sense to add $c-1$ dimensions), one may want to expand training data in the form $x_k \to (x_k, \delta u_k)$ for small δ, with u_k a realization of a standard Gaussian variable to help breaking symmetries in the training phase (test data being still expanded as $x_k \to (x_k, 0)$).

7.2. **Deformable Templates.** It is important to strengthen the fact that, even though our model involves diffeomorphic transformations, it is not a deformable template model. The latter type of model typically works with small dimensional images (k=2 or 3), say $I : \mathbb{R}^k \to \mathbb{R}$, and tries to adjust a k-dimensional deformation (using a diffeomorphism $g : \mathbb{R}^k \to \mathbb{R}^k$) such that the deformed image, given by $I \circ g^{-1}$ mimics a fixed template (and classification based on a finite number of templates would pick the one for which a combination of the deformation and the associated residuals is smallest).

The transformation $\phi_g : I \mapsto I \circ g^{-1}$ is a homeomorphism of the space of, say, continuous images. Once images are discretized over a grid with d points, ϕ_g becomes (assuming that the grid is fine enough) a one-to-one transformation of $\mathbb{R}^d$, but a very special one. In this context, the model described in this paper would be directly applied to discrete images, looking for a d-dimensional diffeomorphism that would include deformations such as ϕ_g, but many others, involving also variations in the image values or more complex transformations.

8. Experiments

We now provide a few experiments that illustrate some of the advantages of the proposed diffeomorphic learning method, and some of its limitations as well. We will compare the performance of the diffeomorphic learning algorithm with a few off-the-shelve methods, namely k-nearest-neighbors, linear and non-linear SVM, random forests (RF), multi-layer perceptrons (MLP) and logistic regression. For all but the last logistic regression, we use the algorithms implemented in the scikit-learn Python package [17] with default parameters. (For logistic regression, we use the same implementation as the one implemented in the final cost of our algorithm.) In particular, ten random trees are learned for RFs (increasing this number does not help much in our examples) and one hidden layer with 100 nodes for MLPs (increasing the number of layers would generally yield poorer performances).

We use a penalized form of logistic regression for our final classifier, minimizing

$$-\sum_{k=1}^{N} \log F(\phi(x_k);\theta)(y_k) + \lambda \sum_{i=1}^{c-1}\sum_{j=1}^{d} \theta_{ij}^2 \sigma_j^2$$

where σ_j is the standard deviation of the jth coordinate of $(\phi(x_1), \ldots, \phi(x_N))$. The dataset is also normalized so that the median value of $(|x_k - x_l|^2, k, l = 1, \ldots, N)$ is equal to 1.

In the following examples, we will consider d-dimensional datasets in which the data in class i is sampled from a distribution p_i supported by a subset $D_i \subset \mathbb{R}^d$. In most cases, we consider two classes with $D_1 \cap D_2 = \emptyset$ so that the data is perfectly separable.

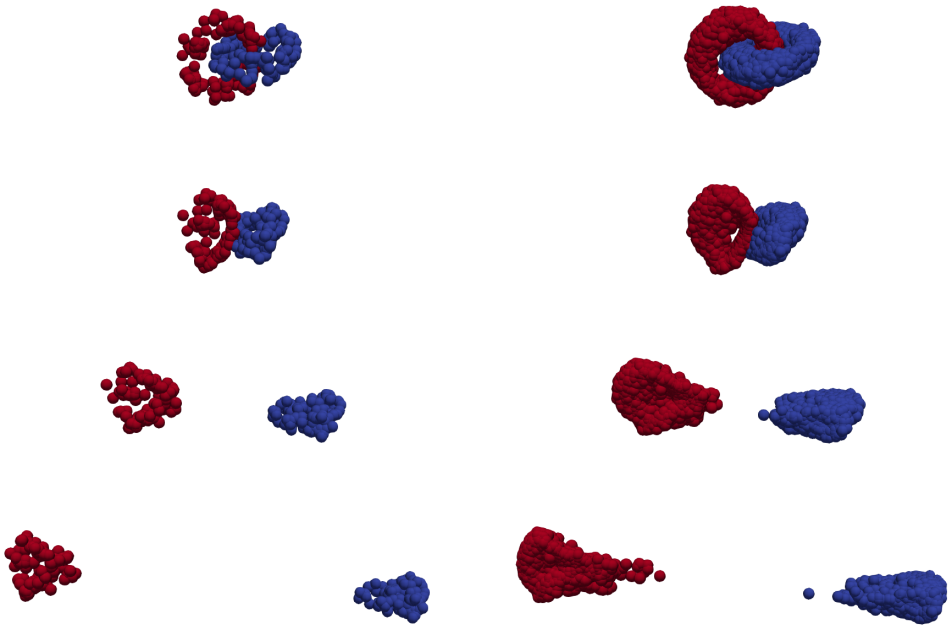

FIGURE 1. Visualisation of the diffeomorphic flow applied to the tori dataset. Left: training data; Right: test data. Top to bottom: $t = 0$, $t = 0.3$, $t = 0.7$, $t = 1$. The data is visualized in a frame formed by the discriminant direction followed by the two principal components in the perpendicular space.

8.1. **Tori datasets.** In our first set of experiments, we let $D_i = R(\mathbb{T}_i \times \mathbb{R}^{d-3})$ where $\mathbb{T}_1$, $\mathbb{T}_2$ are non-intersecting tori in $\mathbb{R}^3$ and R is a random d-dimensional rotation. The tori are positioned as illustrated in the first panels of figure 1, so that, even though they have an empty intersection, they are not linearly separable. The distribution of training and test data is (before rotation) the product of a uniform distribution on the torus and of a standard Gaussian in the $d-3$ remaining dimensions. In all cases, we added a dummy dimension to the data as described in section 7.1.

Classification results for this dataset are summarized in Table 1. Here, we let the number of noise dimensions vary from 0 to 7 and 17 (so that the total number of dimensions is 3, 10 and 20) and the number of training samples vary from 200 to 500, 1000 and 2000. The problem becomes very challenging for most classifiers when the number of noisy dimensions is large. The only exception is the MLP, which still performs quite well. All other classifiers see their performance improved after diffeomorphic transformation of the data. One can also notice that, after transformation, all classifiers perform approximately at the same level. Figure 1 illustrates how the data is transformed by the diffeomorphism and is typical of the other results.

We also point out that all classifiers except RF are invariant by rotation of the data, so that making a random rotation when generating it does not change their performance. The estimation of the diffeomorphism using a radial kernel is also rotation invariant. The RF classifier, which is based on comparison along coordinate axes, is highly affected, however. Without a random rotation, it

	Log. Regression	Lin. SVM	SVM	RF	kNN	MLP
	3 Dimensions, 200 Training samples					
Original Data	0.322	0.321	0.0	0.029	0.0	0.026
Transformed Data	0.0	0.0	0.0	0.0	0.0	0.0
	10 Dimensions, 200 Training samples					
Original Data	0.315	0.315	0.285	0.314	0.328	0.182
Transformed Data	0.228	0.228	0.228	0.230	0.228	0.229
	10 Dimensions, 400 Training samples					
Original Data	0.318	0.316	0.256	0.277	0.269	0.093
Transformed Data	0.146	0.146	0.146	0.148	0.146	146
	10 Dimensions, 1000 Training samples					
Original Data	0.334	0.332	0.234	0.282	0.220	0.039
Transformed Data	0.038	0.037	0.037	0.042	0.037	0.38
	10 Dimensions, 2000 Training samples					
Original Data	0.337	0.337	0.198	0.254	0.181	0.002
Transformed Data	0.006	0.006	0.006	0.006	0.006	0.007
	20 Dimensions, 200 Training samples					
Original Data	0.305	0.307	0.312	0.350	0.375	0.301
Transformed Data	0.293	0.293	0.293	0.295	0.293	0.294
	20 Dimensions, 400 Training samples					
Original Data	0.312	0.312	0.287	0.332	0.366	0.208
Transformed Data	0.267	0.267	0.267	0.271	0.267	0.269
	20 Dimensions, 1000 Training samples					
Original Data	0.329	0.330	0.289	0.304	0.334	0.117
Transformed Data	0.221	0.221	0.221	0.224	0.221	0.223
	20 Dimensions, 2000 Training samples					
Original Data	0.326	0.326	0.280	0.319	0.321	0.032
Transformed Data	0.161	0.160	0.159	0.160	0.160	0.161

TABLE 1. Comparative performance of classifiers on "Tori" data

actually performs extremely well, with an error rate of only 0.02 with 17 noisy dimensions and 2000 examples. Interestingly, if one uses a convolutional kernel to train the diffeomorphism (which

	Log. Reg.	Lin. SVM	SVM	RF	kNN	MLP
	10 Dimensions, 200 Training samples					
Original Data	0.479	0.478	0.119	0.284	0.425	0.276
Transformed Data	0.161	0.160	0.161	0.156	0.159	0.164
	10 Dimensions, 400 Training samples					
Original Data	0.494	0.493	0.090	0.253	0.417	0.220
Transformed Data	0.094	0.093	0.094	0.097	0.093	0.092
	10 Dimensions, 1000 Training samples					
Original Data	0.482	0.482	0.059	0.214	0.394	0.148
Transformed Data	0.055	0.055	0.056	0.059	0.055	0.057
	10 Dimensions, 2000 Training samples					
Original Data	0.469	0.470	0.040	0.184	0.346	0.071
Transformed Data	0.044	0.043	0.043	0.043	0.042	0.042

TABLE 2. Comparative performance of classifiers on "RBF" data

exploits a similar bias in the organization of the data), the error rate of linear classifiers after transformation also drop to around 0.003.

8.2. **RBF datasets.** The second dataset generates classes using radial basis functions. More precisely, we let $\rho(z) = \exp(-z^2)$ and generate classes based on the sign of the function

$$\sin\left(\sum_{j=1}^{L} \rho(|X - c_j|)a_j\right) - \mu$$

where X is a d-dimensional standard Gaussian and μ is estimated so that both positive and negative classes are balanced. The centers, $c_1, \ldots, c_L$ are chosen as $c_j = (j/2L)e_{(j \bmod d)+1}$ where $j \bmod d$ is the remainder of the division of j by d and $e_1, \ldots, e_d$ is the canonical basis of $\mathbb{R}^d$. The coefficients are $a_j = 2(j \bmod 1.5) - 0.75$. In the results summarized in Table 2, we took $d = 10$, $L = 20$.

(Nonlinear) SVM performs best on this dataset, which is not too surprising, because it is essentially tailored for this kind of problem. Linear classifiers are barely better than chance before diffeomorphic transformation. If one excepts the smallest dataset (200 training samples) in which a 5% gap can be observed, the performance of the linear classifiers after transformations are quite similar to those of non-linear SVM before transformation.

8.3. **Spherical Layers.** We here deduce the class from the sign of $\cos(4|X|)$ where X is a d-dimensional standard Gaussian, and we here provide results with $d = 3$. Here, we see that linear classifiers cannot do better than chance (this is by design), but that after transformation, all classifiers outperform, with a large margin, the best non-linear classifiers trained on the original data.

	Log. Reg.	Lin. SVM	SVM	RF	kNN	MLP
	3 Dimensions, 200 Training samples					
Original Data	0.498	0.498	0.439	0.394	0.390	0.351
Transformed Data	0.276	0.276	0.276	0.278	0.277	0.277
	3 Dimensions, 400 Training samples					
Original Data	0.482	0.483	0.364	0.288	0.307	0.298
Transformed Data	0.205	0.205	0.206	0.206	0.206	0.206
	3 Dimensions, 1000 Training samples					
Original Data	0.501	0.501	0.289	0.203	0.228	0.434
Transformed Data	0.131	0.132	0.132	0.133	0.132	0.133
	3 Dimensions, 2000 Training samples					
Original Data	0.477	0.477	0.246	0.196	0.189	0.279
Transformed Data	0.108	0.108	0.109	0.110	0.107	0.107

TABLE 3. Comparative performance of classifiers on 3D spherical layers.

8.4. **Segment Lengths.** Our next synthetic example is a simple pattern recognition problem in $d = 100$ dimensions. Samples from class 0 take the form $(\rho_1 U_1, \ldots, \rho_d, U_d)$ where ρ is uniformly distributed between 0.75 and 1.25, and $U = (U_1, \ldots U_d)$ is a binary vector with exactly $l_1 = 10$ ones, which are consecutive at a random starting location in the vector. For class 1, simply replace l_1 by $l_2 = 11$. So the two classes are perfectly separable by counting the number of non-zero entries. Discovering this simple rule is however quite challenging for many classifiers, especially with small datasets. Here again, classifiers –MLPs excepted– perform significantly better when run on the transformed data.

8.5. **Segment pairs.** This section describes a more challenging version of the former in which each data point consists in two sequences of ones in the unit circle (discretized over 100 points), these two sequences being both of length five in class 1, and of length 4 and 6 (in any order) in class 2, which are not linearly separable. The approach is also challenging for metric-based methods (such as kNN, SVM and diffeomorphic learning) because each data point has close neighbors in the other class: the nearest non-identical neighbor in the same class is obtained by shifting one of the two segments, and would be at distance $\sqrt{2}$, which is identical to the distance of the closest element from the other class which is obtained by replacing a point in one segment by 0 and adding a 1 to the other segment. As shown in Table 5, MLP's perform best on this data, with some margin. Diffeomorphic classification does however improve significantly on the classification rates of all other classifiers.

8.6. **MNIST (Subset).** To conclude this section we provide (in Table 6) classification results on a subset of the MNIST dataset [12], reduced to three classes ("3", "5" and "8") with 1500 examples for training and 2000 for testing. We used the raw data directly as input, with $d = 784$. Diffeomorphic learning slightly improve the original results.

	Log. Reg.	Lin. SVM	SVM	RF	kNN	MLP
	100 Dimensions, 200 Training samples					
Original Data	0.451	0.448	0.470	0.474	0.492	0.312
Transformed Data	0.312	0.312	0.312	0.324	0.312	0.312
	100 Dimensions, 400 Training samples					
Original Data	0.439	0.443	0.477	0.471	0.502	0.216
Transformed Data	0.187	0.187	0.186	0.201	0.187	0.186
	100 Dimensions, 1000 Training samples					
Original Data	0.426	0.433	0.469	0.462	0.321	0.020
Transformed Data	0.025	0.025	0.025	0.0367	0.025	0.025
	100 Dimensions, 2000 Training samples					
Original Data	0.444	0.442	0.433	0.396	0.074	0.0
Transformed Data	0.004	0.004	0.004	0.009	0.004	0.004

TABLE 4. Comparative performance of classifiers on segment lengths

	Log. Reg.	Lin. SVM	SVM	RF	kNN	MLP
	100 Dimensions, 200 Training samples					
Original Data	0.513	0.515	0.460	0.505	0.532	0.411
Transformed Data	0.421	0.421	0.421	0.422	0.421	0.421
	100 Dimensions, 400 Training samples					
Original Data	0.465	0.467	0.498	0.497	0.488	0.144
Transformed Data	0.303	0.303	0.303	0.303	0.303	0.302
	100 Dimensions, 1000 Training samples					
Original Data	0.543	0.549	0.450	0.499	0.403	0.024
Transformed Data	0.146	0.146	0.145	0.145	0.143	0.145
	100 Dimensions, 2000 Training samples					
Original Data	0.514	0.512	0.442	0.510	0.283	0.013
Transformed Data	0.069	0.069	0.069	0.070	0.068	0.068

TABLE 5. Comparative performance of classifiers on segment pair data

	Log. Reg.	Lin. SVM	SVM	RF	kNN	MLP
	100 Dimensions, 1500 Training samples					
Original Data	0.100	0.121	0.093	0.070	0.056	0.065
Transformed Data	0.050	0.051	0.050	0.046	0.044	0.067

TABLE 6. Comparative performance of classifiers on a subset of the MNIST dataset.

9. Discussion

In this paper, we have introduced the concept of diffeomorphic learning and provided a few illustrations of its performance on simple, but often challenging, classification problems. On this class of problems, the proposed approach appeared quite competitive among other classifiers used as comparison (the method always ended up either first or second). Some limitations also appeared, that we will discuss in more details. We however point out that all classifiers that were run in our experiments were used "off-the-shelves", as provided from a software package. There is no doubt that, after some tuning up specific to each dataset, each of these classifiers could perform better. Such an analysis was not our goal here, because this paper must be considered as a proof of concept for the method we introduce, and not an attempt at a fair ranking of existing methods. So, the classification rates that we obtained must be analyzed with this in mind. We just note that we also used the exact same version of diffeomorphic learning for all datasets, with fixed values of the parameters and their dependency on the size of the training set.

We first remark that diffeomorphic learning is, because of the use of an RKHS as baseline space for vector fields, a kernel method, which means that it relies on a proper choice of metric in the ambient space. Even though diffeomorphisms enable extremely large transformations, the solution, because of the smoothness penalty, is still a C^1 transformation of the original data and tries to minimize the distortion while classifying the training data (we note that the training set error was zero for all our examples). It would not be difficult, however, to extend the proposed approach to a setting in which an affine transformation is estimated in addition to the diffeomorphism, through an additional finite dimensional control. This addition would not be redundant with the estimated diffeomorphism, because groups of diffeomorphisms of $\mathbb{R}^d$ generated by RKHS's with radial kernels only contain mappings that tend to the identity at infinity. Such methods are already used in shape analysis [21], but their applications with large dimensional data will require additional work in order to reduce the complexity of the estimation of the d^2 parameters describing an affine transformation.

The proposed method shares at least one similarity with deep learning [8], in that the prediction is based on a dynamical system, with a gradient computed with a back-propagation algorithm. But there are notable differences, starting with the fact that diffeomorphic learning estimates an invertible transformation of the training data rather than a set of discriminating features. Each point in the final reference frame, observed or not, can be traced back to a unique point in the original frame, which could quite naturally lead to generative data models. Estimating a diffeomorphic transformation is however much more numerically ambitious, as we will soon discuss, resulting, in theory, in the estimation of d d-dimensional displacement vectors per data point, where d is the

dimension. (The reduction discussed in section 3 "only" requires computing a d-dimensional vector per training point.) The number of features per dimension (e.g., per pixel) estimated in deep networks is generally much lower.

Unsurprisingly, computation time is currently the main limitation of the method. The largest synthetic examples had 2000 training samples in 100 dimensions, for which training required about one day one our current implementation (on a four-core Intel i7 laptop), and the MNIST example ran over several days. While we have no illusion on the optimality of our own implementation, and similar implementations on more powerful platforms, notably GPU arrays, are available in shape analysis, more thoughts need to be given in trying to reduce the computational load. The computational bottleneck is the evaluation of system (10), which is linear in the number of time steps, T, linear in the dimension, d (assuming a scalar radial kernel) and quadratic in the sample size N. One of the most direct approaches in order to reduce the cost is to replace the optimal expression of v in equation (13) by a sum over a subset of the training points, i.e., to take

$$v(t,\cdot) = \sum_{j=1}^{n} K(\cdot, z_{k_j}(t))a_j(t) \tag{13}$$

with $\partial_t z_{k_j} = v(t, z_{k_j})$ and $z_{k_j}(0) = x_{k_j}$ where $\{k_1, \ldots, k_n\}$ is a subset of $\{1, \ldots, N\}$. The dependency of the computational cost per iteration in the data size would now be of order nN instead of N^2 (because the evolution of all points is still needed to evaluate the objective function and its gradient), but this would also allow for randomized evaluations of the training data (stochastic gradient descent) that would replace nN by nN' where N' is the size a random subsample used at each iteration. This general scheme will be explored in future work, where optimal strategies in selecting an n-dimensional subset of the training data must be analyzed, including in particular the trade-off it implies between computation time and sub-optimality of solutions. Even though they appeared in a very different context, some inspiration may be obtained from similar approaches that were explored in shape analysis [22, 5, 23, 6, 9].

We have only considered, in this paper, applications of diffeomorphic learning to classification problems. Extensions to other contexts will be considered in the future. Some, such as regression, may be relatively straightforward, while others, such as clustering, or dimension reduction should require additional thoughts, as the obtained results will be highly dependent on the amount of metric distortion allowed in the diffeomorphic transformation.

References

[1] N. Aronszajn. Theory of reproducing kernels. *Trans. Am. Math. Soc.*, 68:337–404, 1950.

[2] M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *Int J. Comp. Vis.*, 61(2):139–157, 2005.

[3] MD Buhmann. *Radial basis functions: theory and implementations, volume 12 of Cambridge Monographs on Applied and Computational Mathematics*. Cambridge University Press, Cambridge, 2003.

[4] J. Duchon. Interpolation des fonctions de deux variables suivant le principe de la flexion des plaques minces. *R.A.I.R.O. Analyse Numerique*, 10:5–12, 1977.

[5] Stanley Durrleman, Stéphanie Allassonnière, and Sarang Joshi. Sparse adaptive parameterization of variability in image ensembles. *International Journal of Computer Vision*, 101(1):161–183, 2013.

[6] Stanley Durrleman, Marcel Prastawa, Nicolas Charon, Julie R Korenberg, Sarang Joshi, Guido Gerig, and Alain Trouvé. Morphometry of anatomical shape complexes with dense deformations and sparse parameters. *NeuroImage*, 101:35–49, 2014.

[7] N. Dyn. Interpolation and approximation by radial and related functions. In C. K. Chui, L. L. Shumaker, and J. D. Ward, editors, *Approximation Theory VI: vol. 1*, pages 211–234. Academic Press, 1989.
[8] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
[9] Barbara Gris, Stanley Durrleman, and Alain Trouvé. A sub-riemannian modular framework for diffeomorphism-based analysis of shape ensembles. *SIAM Journal on Imaging Sciences*, 11(1):802–833, 2018.
[10] Leslie M Hocking. *Optimal control: an introduction to the theory with applications*. Oxford University Press, 1991.
[11] S. Joshi and M. Miller. Landmark matching via large deformation diffeomorphisms. *IEEE transactions in Image Processing*, 9(8):1357–1370, 2000.
[12] Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.
[13] Jack Macki and Aaron Strauss. *Introduction to optimal control theory*. Springer Science & Business Media, 2012.
[14] J. Meinguet. Multivariate interpolation at arbitrary points made simple. *J. Applied Math. and Physics*, 30:292–304, 1979.
[15] Mario Micheli and Joan A. Glaunès. Matrix-valued kernels for shape deformation analysis. *Geom. Imaging Comput.*, 1(1):57–139, 2014.
[16] Michael I Miller, Alain Trouvé, and Laurent Younes. Hamiltonian systems and optimal control in computational anatomy: 100 years since d'arcy thompson. *Annual review of biomedical engineering*, 17:447–509, 2015.
[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
[18] B Schölkopf and J Smola, A. *Learning with kernels*. MIT Press, 2002.
[19] T. L. Vincent and W. J. Grantham. *Nonlinear and Optimal Control Systems*. Wiley, 1997.
[20] G. Wahba. *Spline Models for Observational Data*. SIAM, 1990.
[21] Laurent Younes. *Shapes and diffeomorphisms*, volume 171. Springer Science & Business Media, 2010.
[22] Laurent Younes. Constrained diffeomorphic shape evolution. *Foundations of Computational Mathematics*, 12(3):295–325, 2012.
[23] Laurent Younes. Gaussian diffeons for surface and image matching within a lagrangian framework. *Geometry, Imaging and Computing*, 1(1):141–171, 2014.

Department of Applied Mathematics and Statistics and Center for Imaging Science, Johns Hopkins University.

E-mail address: laurent.younes@jhu.edu