Apprentissage d'ensemble basé sur des points de repère avec des caractéristiques de Fourier aléatoires et un renforcement du gradient

Léo Gautheron[1], Pascal Germain[2], Amaury Habrard[1], Guillaume Metzler[1], Emilie Morvant[1], Marc Sebban[1], and Valentina Zantedeschi

[1] Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, Saint-Etienne, France
firstname.name@univ-st-etienne.fr
[2] Département d'Informatique et de Génie Logiciel, Université Laval, Québec, Canada
pascal.germain@ift.ulaval.ca
[3] Inria Lille - Nord Europe, Modal Project-Team, 59650 Villeneuve d'Ascq, France

Abstract. Cet article s'appuie sur deux stratégies d'apprentissage de pointe, le renforcement du gradient (gradient boosting (GB)) et les caractéristiques de Fourier aléatoires (random Fourier features (RFF)), pour résoudre le problème de l'apprentissage de noyau. Notre étude s'appuie sur un résultat récent montrant que l'on peut apprendre une distribution par le biais des RFF pour produire un nouveau noyau adapté à la tâche à accomplir. Pour apprendre cette distribution, nous exploitons un schéma de GB exprimé sous forme d'ensembles d'apprenants faibles RFF, chacun d'entre eux étant une fonction du noyau conçue pour apprendre les résidus. Contrairement aux techniques d'apprentissage de noyaux multiples qui utilisent un dictionnaire pré-calculé de fonctions du noyau, à chaque itération nous apprenons un noyau à partir des données d'apprentissage comme une somme pondérée de RFF. Cette stratégie permet de construire un classificateur basé sur un petit ensemble de noyaux "repères", mieux adaptés à l'application en question. Nous effectuons une analyse expérimentale approfondie pour mettre en évidence les avantages de notre méthode par rapport aux méthodes de pointe basées sur le boosting et l'apprentissage de noyaux.

Keywords: Gradient boosting · Random Fourier features · Kernel learning

1 Introduction

Kernel methods are among the most popular approaches in machine learning due to their capability to address non-linear problems, their robustness and their simplicity. However, they exhibit two main flaws in terms of memory usage and time complexity. Landmark-based kernel approaches [2, 18] can be used to drastically reduce the number of instances involved in the comparisons, but they heavily depend on the choice and the parameterization of the kernel. Multiple

Kernel Learning [16] and Matching Pursuit methods [14] can provide alternative solutions to this problem but these require the use of a pre-defined dictionary of base functions. Another strategy to improve the scalability of kernel methods is to use approximation techniques such as the Nyström method [3, 15] or Random Fourier Features (RFF) based approaches [6, 12, 17]. The latter are probably the most used thanks to their simplicity and ease of computation. They allow the approximation of any shift-invariant kernel with random features based on the Fourier transform of the kernel. Several works have extended this family of methods by allowing one to adapt the RFF approximation directly from the training data [1, 7, 10, 13, 17]. Among them, the recent work of Letarte *et al.* [7] introduces a method to obtain a weighting distribution over the random features by a single pass over them. This strategy is derived from a statistical learning analysis, starting from the observation that each random feature can be interpreted as a weak hypothesis in the form of trigonometric functions obtained by the Fourier decomposition. Thus, a classifier can be seen as a weighted majority vote over the random features. This decomposition is then considered as a *prior* distribution over the space of weak hypotheses/random features; they propose to learn a *posterior* distribution by optimizing a PAC-Bayesian bound *w.r.t* a kernel alignment generalization loss over the training data. In other words, this corresponds to automatically learning a representation of the samples through the approximation of a kernel suited for the task at hand. However, in practice, this method requires the use of a fixed set of landmarks selected beforehand and independently from the task. It is only once these landmarks are selected that the method can learn a representation based on the PAC-Bayesian bound. This leads to three important limitations: *(i)* the need for a heuristic strategy for selecting relevant landmarks, *(ii)* these latter and the associated representation might not be adapted for the underlying task, and *(iii)* the number of landmarks might not be minimal *w.r.t.* that task, inducing higher computational and memory costs.

We propose in this paper to tackle the aforementioned issues with a gradient boosting approach [4]. Our aim is to learn iteratively the classifier and a compact and efficient representation at the same time. Our greedy optimization method is similar to Oglic & Gärtner's one [9], which at each iteration of the functional gradient descent [8] refines the representation by adding the base function minimizing a residual-based loss function. But unlike our approach, their method does not allow to learn a classifier at the same time. Instead, we propose to jointly optimize the classifier and the base functions in the form of kernels by leveraging both gradient boosting and RFF. Interestingly, we further show that we can benefit from a significant performance boost by *(i)* considering each weak learner as a single trigonometric feature, and *(ii)* learning the random part of the RFF.

Organization of the paper. Section 2 describes the notations and the necessary background knowledge. We present our method in Section 3 as well as two efficient refinements before presenting an extensive experimental study in Section 4, comparing our strategy with boosting-based and kernel learning methods.

2 Notations and Related Work

We consider binary classification tasks from a d-dimensional input space $\mathbb{R}^d$ to a label set $Y=\{-1,+1\}$. Let $S=\{(\mathbf{x}_i,y_i)\}_{i=1}^n \sim \mathcal{D}^n$ be a training set of n points sampled *i.i.d.* from $\mathcal{D}$, a fixed and unknown data-generating distribution over $\mathbb{R}^d\times Y$. We focus on kernel-based algorithms that rely on pre-defined kernel functions $k:\mathbb{R}^d\times\mathbb{R}^d\to\mathbb{R}$ assessing the similarity between any two points of the input space. These methods present a good performance when the parameters of the kernels are learned and the chosen kernels are able to fit the distribution of the data. However, selecting the right kernel and tuning its parameters is computationally expensive, in general. To reduce this overhead, one can resort to Multiple Kernel Learning techniques [16] which boil down to selecting the combination of kernels that fits the best the training data: a dictionary of T base functions $\{k^t\}_{t=1}^T$ is composed by various kernels associated with some fixed parameters, and a combination is learned, defined as

$$H(\mathbf{x},\mathbf{x}') \;=\; \sum_{t=1}^{T} \alpha^t\, k^t(\mathbf{x},\mathbf{x}'), \tag{1}$$

with $\alpha^t\in\mathbb{R}$ the weight of the kernel $k^t(\mathbf{x},\mathbf{x}')$. As shown in Section 3, our main contribution is to address this issue of optimizing a linear combination of kernels by leveraging RFF and gradient boosting (we recall basics on it in Section 3.1). To avoid the dictionary of kernel functions in Equation (1) from being pre-computed, we propose a method inspired from Letarte et *al.* [7] to learn a set of approximations of kernels tailored to the underlying classification task. Unlike Letarte et *al.*, we learn such functions so that the representation and the classifier are jointly optimized. We consider landmark-based shift-invariant kernels relying on the value $\boldsymbol{\delta}=\mathbf{x}^t-\mathbf{x}\in\mathbb{R}^d$ and usually denoted by abuse of notation by $k(\boldsymbol{\delta}) = k(\mathbf{x}^t-\mathbf{x}) = k(\mathbf{x}^t,\mathbf{x})$, where $\mathbf{x}^t\in\mathbb{R}^d$ is a point—called landmark—lying on the input space which all the instances are compared to, and that strongly characterizes the kernel. At each iteration of our gradient boosting procedure, we optimize the kernel function itself, exploiting the flexibility of the framework of Letarte *et al.*, where a kernel is a weighted sum of RFF [12] defined as

$$k_{q^t}(\mathbf{x}^t-\mathbf{x}) = \sum_{j=1}^{K} q_j^t \cos\big(\boldsymbol{\omega}_j\cdot(\mathbf{x}^t-\mathbf{x})\big), \tag{2}$$

where the $\boldsymbol{\omega}_j$ are drawn from the Fourier transform of a shift invariant kernel k denoted by $p(\boldsymbol{\omega})$ and defined as

$$p(\boldsymbol{\omega}) = \frac{1}{(2\pi)^d}\int_{\mathbb{R}^d} k(\boldsymbol{\delta})e^{-i\boldsymbol{\omega}\cdot\boldsymbol{\delta}}d\boldsymbol{\delta}. \tag{3}$$

When $q^t=p$, we retrieve the classical setting of RFF and we have $k(\boldsymbol{\delta})\simeq k_{q^t}(\boldsymbol{\delta})$. It is known that the larger the number of random features K in Equation (2), the better the resulting approximation [12]. Letarte *et al.* [7] aim to learn the weights

Algorithm 1: Gradient boosting [4]

Inputs : Training set $S = \left\{(\mathbf{x}_i, y_i)\right\}_{i=1}^{n}$; Loss ℓ; Number of iterations T

Output: sign $\left(H^0(\mathbf{x}) + \sum_{t=1}^{T} \alpha^t h_{a^t}(\mathbf{x})\right)$

1: $\forall i = 1, \ldots, n, \quad H^0(\mathbf{x}_i) = \operatorname{argmin}_\rho \sum_{i=1}^{n} \ell(y_i, \rho)$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad \forall i = 1, \ldots, n, \quad \tilde{y}_i = -\dfrac{\partial \ell\big(y_i, H^{t-1}(\mathbf{x}_i)\big)}{\partial H^{t-1}(\mathbf{x}_i)}$
4: $\quad a^t = \operatorname{argmin}_a \sum_{i=1}^{n} \big(\tilde{y}_i - h_a(\mathbf{x}_i)\big)^2$
5: $\quad \alpha^t = \operatorname{argmin}_\alpha \sum_{i=1}^{n} \ell\big(y_i, H^{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i)\big)$
6: $\quad \forall i = 1, \ldots, n, \quad H^t(\mathbf{x}_i) = H^{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$
7: **end for**

of the random Fourier features q^t. To do so, they consider a loss function ℓ that measures the quality of the similarities computed using the kernel k_{q^t}. Their theoretical study on ℓ leads to a closed-form solution for q^t computed as

$$\forall j \in \{1, \ldots, K\}, \quad q_j^t = \frac{1}{Z^t} \exp\Big(\frac{-\beta\sqrt{n}}{n} \sum_{i=1}^{n} \ell(h_{\boldsymbol{\omega}^j}^t(\mathbf{x}_i))\Big), \tag{4}$$

with $\beta \geq 0$ a parameter to tune, $h_{\boldsymbol{\omega}}^t(\mathbf{x}) = \cos\left(\boldsymbol{\omega} \cdot (\mathbf{x}^t - \mathbf{x})\right)$, and Z^t a normalization constant such that $\sum_{j=1}^{K} q_j^t = 1$. They learn a representation of the input space of n_L features where each of them is computed using k_{q^t} with the landmark $(\mathbf{x}^t, y^t)$ selected randomly from the training set. Once the new representation is computed, a (linear) predictor is learned from it, in a second step.

It is worth noticing that this kind of procedure exhibits two limitations. First, the model can be optimized only after having learned the representation. Second, the landmarks have to be fixed before learning the representation. Thus, the constructed representation is not guaranteed to be compact and relevant for the learning algorithm considered. To tackle these issues, we propose in the following a strategy that performs both steps at the same time through a gradient boosting process that allows to jointly learn the set of landmarks and the final predictor.

3 Gradient Boosting Random Fourier Features

The approach we propose follows the widely used gradient boosting framework first introduced by Friedman [4]. We briefly recall it below.

3.1 Gradient Boosting in a Nutshell

Gradient boosting is an ensemble method that aims at learning a weighted majority vote over an ensemble of T weak predictors in a greedy way by learning one classifier per iteration. The final majority vote is of the form

$$\forall \mathbf{x} \in \mathbb{R}^d, \ \operatorname{sign}\left(H^0(\mathbf{x}) + \sum_{t=1}^{T} \alpha^t h_{a^t}(\mathbf{x})\right),$$

Algorithm 2: GBRFF1

Inputs : Training set $S = \left\{(\mathbf{x}_i, y_i)\right\}_{i=1}^{n}$; Number of iterations T;
K number of random features; Parameters γ and β

Output: $\text{sign}\left(H^0(\mathbf{x}) + \sum_{t=1}^{T} \alpha^t \sum_{j=1}^{K} q_j^t \cos\left(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x})\right)\right)$

1: $H^0 \leftarrow H^0(\mathbf{x}_i) = \frac{1}{2} \ln \frac{1+\frac{1}{n}\sum_{j=1}^{n} y_j}{1-\frac{1}{n}\sum_{j=1}^{n} y_j}$

2: **for** $t = 1, \ldots, T$ **do**

3: $\quad \forall i = 1, \ldots, n, \quad w_i = \exp(-y_i H^{t-1}(\mathbf{x}_i))$

4: $\quad \forall i = 1, \ldots, n, \quad \tilde{y}_i = y_i w_i$

5: $\quad$ Draw $\{\boldsymbol{\omega}_j^t\}_{j=1}^{K} \sim \mathcal{N}(0, 2\gamma)^{K \times d}$

6: $\quad \mathbf{x}^t = \underset{\mathbf{x} \in \mathbb{R}^d}{\text{argmin}} \; \frac{1}{n} \sum_{i=1}^{n} \exp\Big(- \tilde{y}_i \frac{1}{K} \sum_{j=1}^{K} \cos(\boldsymbol{\omega}_j^t \cdot (\mathbf{x} - \mathbf{x}_i))\Big)$.

7: $\quad \forall j = 1, \ldots, K, \; q_j^t = \frac{1}{Z^t} \exp\left(\frac{-\beta\sqrt{n}}{n} \sum_{i=1}^{n} \exp\Big(- \tilde{y}_i \cos\left(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i)\right)\Big)\right)$

8: $\quad \alpha^t = \frac{1}{2} \ln \frac{\sum_{i=1}^{n} \left(1 + y_i \sum_{j=1}^{K} q_j^t \cos\left(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i)\right)\right) w_i}{\sum_{i=1}^{n} \left(1 - y_i \sum_{j=1}^{K} q_j^t \cos\left(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i)\right)\right) w_i}$

9: $\quad \forall i = 1, \ldots, n, \; H^t(\mathbf{x}_i) = H^{t-1}(\mathbf{x}_i) + \alpha^t \sum_{j=1}^{K} q_j^t \cos\left(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i)\right)$

10: **end for**

where H^0 is an initial classifier fixed before the iterative process (usually set such that it returns the same value for every sample), and α^t is the weight associated to the predictor h_{a^t} and is learned at the same time as the parameters a^t of that classifier. Given a differentiable loss ℓ, the objective of the gradient boosting algorithm is to perform a gradient descent where the variable to be optimized is the ensemble and the function to be minimized is the empirical loss. The pseudo-code of gradient boosting is reported in Algorithm 1. First, the ensemble is constituted by only one predictor: the one that outputs a constant value minimizing the loss over the whole training set (line **1**). Then at each iteration, the algorithm computes for each training example the negative gradient of the loss (line **3**), also called the residual and denoted by $\tilde{y}_i$. The next step consists in optimizing the parameters of the predictor h_{a^t} that fits the best the residuals (line **4**), before learning the optimal step size α^t that minimizes the loss by adding h_{a^t}, weighted by α^t, to the current vote (line **5**). Finally, the model is updated by adding $\alpha^t h_{a^t}(\cdot)$ (line **6**) to the vote.

3.2 Gradient Boosting with Random Fourier Features

Our main contribution takes the form of a learning algorithm which jointly optimizes a compact representation of the data and the model. Our method, called **GBRFF1**, leverages both Gradient Boosting and RFF. We describe its pseudo-code in Algorithm 2. We present below its main constitutive elements by following the steps of the gradient boosting method of Algorithm 1.

The loss function ℓ at the core of our algorithm is the exponential loss:

$$\ell\left(H^T\right) = \frac{1}{n}\sum_{i=1}^{n}\exp\Big(-y_i\, H^T(\mathbf{x}_i)\Big). \tag{5}$$

Given $\ell\left(H^T\right)$, line **1** of Algorithm 1 amounts to setting the initial learner as

$$\forall i \in \{1,\ldots,n\}, \quad H^0(\mathbf{x}_i) = \frac{1}{2}\ln\frac{1+\frac{1}{n}\sum_{j=1}^{n} y_j}{1-\frac{1}{n}\sum_{j=1}^{n} y_j}. \tag{6}$$

The residuals of line **3** are defined as $\tilde{y}_i = -\dfrac{\partial\ell\big(y_i, H^{t-1}(\mathbf{x}_i)\big)}{\partial H^{t-1}(\mathbf{x}_i)} = y_i\, \mathrm{e}^{-y_i H^{t-1}(\mathbf{x}_i)}$. Line **4** of Algorithm 1 tends to learn a weak learner that outputs exactly the residuals' values by minimizing the squared loss; but, this is not well-suited in our setting with the exponential loss (Equation (5)). To benefit from the exponential decrease of the loss, we are rather interested in weak learners that output predictions having a large absolute value and being of the same sign as the residuals. Thus, we aim at favoring parameter values minimizing the exponential loss between the residuals and the predictions of the weak learner as follows:

$$a^t = \operatorname*{argmin}_{a} \frac{1}{n}\sum_{i=1}^{n}\exp\big(-\tilde{y}_i h_a(\mathbf{x}_i)\big). \tag{7}$$

Following the RFF principle, we can now define our weak learner as

$$h_{a^t}(\mathbf{x}_i) = \sum_{j=1}^{K} q_j^t \cos\left(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i)\right), \tag{8}$$

where its parameters are given by $a^t{=}(\{\boldsymbol{\omega}_j^t\}_{j=1}^K, \mathbf{x}^t, q^t)$. Instead of using a predefined set of landmarks [7], we build this set iteratively, *i.e.*, we learn one landmark per iteration. To benefit from the closed form of Equation (4), we propose the following greedy approach to learn the parameters a^t. At each iteration t, we draw a set of K random features $\{\boldsymbol{\omega}_j^t\}_{j=1}^K{\sim}p^K$ with p the Fourier transform of a given kernel (as defined in Equation (3)); then we are looking for the optimal landmark $\mathbf{x}^t$. Plugging Equation (8) into Equation (7) and assuming a uniform prior distribution over the random features, $\mathbf{x}^t$ is learned to minimize

$$\mathbf{x}^t = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\exp\Big(-\tilde{y}_i\frac{1}{K}\sum_{j=1}^{K}\cos(\boldsymbol{\omega}_j^t\cdot(\mathbf{x}-\mathbf{x}_i))\Big). \tag{9}$$

Even if this problem is non-convex due to the cosine function, we can still compute its derivative and perform a gradient descent to find a possible solution. The partial derivative of Equation (9) with respect to $\mathbf{x}$ is given by

$$\frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}){=}\frac{1}{Kn}\sum_{i=1}^{n}\left[\frac{\tilde{y}_i}{K}\sum_{j=1}^{K}\sin(\boldsymbol{\omega}_j^t\cdot(\mathbf{x}{-}\mathbf{x}_i))\right]\exp\left[-\frac{\tilde{y}_i}{K}\sum_{j=1}^{K}\cos(\boldsymbol{\omega}_j^t\cdot(\mathbf{x}{-}\mathbf{x}_i))\right]\sum_{j=1}^{K}\boldsymbol{\omega}_j^t.$$

According to Letarte *et al.* [7], given the landmark $\mathbf{x}^t$ found by gradient descent, we can now compute the weights of the random features q^t as

$$\forall j \in \{1,\dots,K\},\quad q_j^t = \frac{1}{Z^t}\exp\left[\frac{-\beta\sqrt{n}}{n}\sum_{i=1}^{n}\exp\Big(-\tilde{y}_i\cos\big(\boldsymbol{\omega}_j^t\cdot(\mathbf{x}^t-\mathbf{x}_i)\big)\Big)\right], \quad (10)$$

with $\beta \geq 0$ a parameter to tune and Z^t the normalization constant.

The last step concerns the step size α^t. It is computed so as to minimize the combination of the current model H^{t-1} with the weak learner h^t, *i.e.*,

$$\alpha^t = \underset{\alpha}{\operatorname{argmin}}\sum_{i=1}^{n}\exp\big[-y_i(H^{t-1}(\mathbf{x}_i)+\alpha h^t(\mathbf{x}_i))\big] = \underset{\alpha}{\operatorname{argmin}}\sum_{i=1}^{n} w_i\exp\big[-y_i\alpha h^t(\mathbf{x}_i)\big],$$

where $w_i = \exp(-y_i H^{t-1}(\mathbf{x}_i))$. In order to have a closed-form solution of α, we use the convexity of the above quantity and the fact that $h^t(\mathbf{x}_i) \in [-1,1]$ to bound the loss function to optimize. Indeed, we get

$$\sum_{i=1}^{n} w_i \mathrm{e}^{-y_i\alpha h^t(\mathbf{x}_i)} \leq \sum_{i=1}^{n}\left[\frac{1-y_i h^t(\mathbf{x}_i)}{2}\right] w_i \mathrm{e}^{\alpha} + \sum_{i=1}^{n}\left[\frac{1+y_i h^t(\mathbf{x}_i)}{2}\right] w_i \mathrm{e}^{-\alpha}.$$

This upper bound is strictly convex. Its minimum α^t can be found by setting to 0 the derivative *w.r.t.* α of the right-hand side of the previous equation. We get

$$\sum_{i=1}^{n}\left(\frac{1-y_i h^t(\mathbf{x}_i)}{2}\right) w_i\,\mathrm{e}^{\alpha} = \sum_{i=1}^{n}\left(\frac{1+y_i h^t(\mathbf{x}_i)}{2}\right) w_i\,\mathrm{e}^{-\alpha},$$

for which the solution is given by $\alpha^t = \dfrac{1}{2}\ln\left(\dfrac{\sum_{i=1}^{n}(1-y_i h^t(\mathbf{x}_i))w_i}{\sum_{i=1}^{n}(1+y_i h^t(\mathbf{x}_i))w_i}\right)$.

The same derivation can be used to find the initial predictor H^0.

Note that, as usually done in the RFF literature [1, 6, 12, 13, 17], in **GBRFF1** we make use of the RBF kernel $k_\gamma(\mathbf{x},\mathbf{x}') = \mathrm{e}^{-\gamma\|\mathbf{x}-\mathbf{x}'\|^2}$ with corresponding Fourier transform the normal law $\mathcal{N}(0,2\gamma)^d$.

3.3 Refining GBRFF1

In **GBRFF1**, the number of random features used at each iteration K has a direct impact on the computation time of the algorithm. Moreover $\boldsymbol{\omega}^t$ is drawn according to the Fourier transform of the RBF kernel and thus is not learned. The second part of our contribution is to propose two refinements. First, we bring to light the fact that one can drastically reduce the complexity of **GBRFF1** by learning a rough approximation of the kernel, yet much simpler and still very effective, using K=1. In this scenario, we show that learning the landmarks boils down to finding a single real number in $[-\pi,\pi]$. Then, to speed up the convergence of the algorithm, we suggest to optimize $\boldsymbol{\omega}^t$ after a random initialization from the Fourier transform. We show that a simple gradient descent with respect

Algorithm 3: GBRFF2

Inputs : Training set $S = \left\{(\mathbf{x}_i, y_i)\right\}_{i=1}^{n}$; Number of iterations T;
Parameters γ and λ

Output: sign $\Big(H^0(\mathbf{x}) + \sum_{t=1}^{T} \alpha^t \cos\left(\boldsymbol{\omega}^t \cdot \mathbf{x}_i - b^t\right)\Big)$

1: $H^0 \leftarrow H^0(\mathbf{x}_i) = \frac{1}{2} \ln \frac{\sum_{j=1}^{n} \left(1+y_j\right)}{\sum_{j=1}^{n} \left(1-y_j\right)}$
2: **for** $t = 1, \ldots, T$ **do**
3: $\quad \forall i = 1, \ldots, n, \quad w_i = \exp(-y_i H^{t-1}(\mathbf{x}_i))$
4: $\quad \forall i = 1, \ldots, n, \quad \tilde{y}_i = y_i w_i$
5: $\quad$ Draw $\boldsymbol{\omega} \sim \mathcal{N}(0, 2\gamma)^d$
6: $\quad b^t = \underset{b \in [-\pi, \pi]}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \exp\Big(- \tilde{y}_i \cos\Big(\boldsymbol{\omega} \cdot \mathbf{x}_i - b)\Big)\Big)$
7: $\quad \boldsymbol{\omega}^t = \underset{\boldsymbol{\omega} \in \mathbb{R}^d}{\operatorname{argmin}} \lambda \|\boldsymbol{\omega}\|_2^2 + \frac{1}{n} \sum_{i=1}^{n} \exp\Big(- \tilde{y}_i \cos\Big(\boldsymbol{\omega} \cdot \mathbf{x}_i - b^t)\Big)\Big)$.
8: $\quad \alpha^t = \frac{1}{2} \ln \frac{\sum_{i=1}^{n} \Big(1+y_i \cos\left(\boldsymbol{\omega}^t \cdot \mathbf{x}_i - b^t\right)\Big) w_i}{\sum_{i=1}^{n} \Big(1-y_i \cos\left(\boldsymbol{\omega}^t \cdot \mathbf{x}_i - b^t\right)\Big) w_i}$
9: $\quad \forall i = 1, \ldots, n, \; H^t(\mathbf{x}_i) = H^{t-1}(\mathbf{x}_i) + \alpha^t \cos\left(\boldsymbol{\omega}^t \cdot \mathbf{x}_i - b^t\right)$
10: **end for**

to this parameter allows a faster convergence with better performance. These two improvements lead to a variant of our original algorithm, called **GBRFF2** and presented in Algorithm 3.

Cheaper landmark learning using the periodicity of the cosine. As we set K=1, the weak learner $h_{a^t}(\mathbf{x})$ is now simply defined as

$$h_{a^t}(\mathbf{x}) = \cos\left(\boldsymbol{\omega}^t \cdot (\mathbf{x}^t - \mathbf{x}_i)\right),$$

where its parameters are given by $a^t = (\boldsymbol{\omega}^t, \mathbf{x}^t)$. As said previously, this formulation allows us to eliminate the dependence on the hyper-parameter K. Moreover, one can also get rid of β, because learning the weights q_j^t of the random features (line **7** of Algorithm 2) is no more necessary. Instead, since K=1, we can see α^t learned at each iteration as a surrogate of the weight of each random feature. As our weak learner is based on a single random feature, the objective function (line **6**) to learn the landmark at iteration t becomes

$$\mathbf{x}^t = \underset{\mathbf{x} \in \mathbb{R}^d}{\operatorname{argmin}} \, f_{\boldsymbol{\omega}^t}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \exp\Big(- \tilde{y}_i \cos(\boldsymbol{\omega}^t \cdot (\mathbf{x} - \mathbf{x}_i))\Big).$$

Let $c \in [\![1, d]\!]$ be the index of the c-th coordinate of the landmark $\mathbf{x}^t$. We rewrite the objective function as

$$f_{\boldsymbol{\omega}^t}(\mathbf{x}^t) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{e}^{-\tilde{y}_i \cos\left(\boldsymbol{\omega}^t \cdot \mathbf{x}^t - \boldsymbol{\omega}^t \cdot \mathbf{x}_i\right)} = \frac{1}{n} \sum_{i=1}^{n} \mathrm{e}^{-\tilde{y}_i \cos\left(\boldsymbol{\omega}_c^t \mathbf{x}_c^t + \sum_{j \neq c} \boldsymbol{\omega}_j^t \mathbf{x}_j^t - \boldsymbol{\omega}^t \cdot \mathbf{x}_i\right)}.$$

Note that we can leverage the periodicity of the cosine function along each direction to find the optimal c-th coordinate of the landmark $\mathbf{x}_c^t \in [\frac{-\pi}{\boldsymbol{\omega}_c^t}, \frac{\pi}{\boldsymbol{\omega}_c^t}]$ that minimizes $f_{\boldsymbol{\omega}^t}(\mathbf{x}^t)$ by fixing all the other coordinates. Figure 1 illustrates this phenomenon on the two-moons dataset when applying **GBRFF1** with K=1. The plots in the first row show the periodicity of the loss represented as repeating diagonal green/yellow stripes (light yellow is associated to the smallest value taken by the loss). Note that there is an infinite number of landmarks giving such a minimal loss at the middle of the yellow stripes. Thus, it suffices to set one coordinate of the landmark to an arbitrary value, and then the algorithm is still able at any iteration to find along the second coordinate a value that minimizes the loss (the resulting landmark at the current iteration is depicted by a white cross). The second row shows that such a strategy allows us to get an accuracy of 100% on this toy 2D dataset after 10 iterations.

By generalizing this principle, instead of learning a landmark vector $\mathbf{x}^t \in \mathbb{R}^d$, we fix all but one coordinate of the landmark, *e.g.* to 0, and then learn a single scalar $b^t \in [-\pi, \pi]$ that minimizes

$$f_{\boldsymbol{\omega}^t}(b^t) = \frac{1}{n}\sum_{i=1}^{n} \exp\left(-\tilde{y}_i \cos\left(\boldsymbol{\omega}^t \cdot \mathbf{x}_i - b^t\right)\right).$$

Learning $\boldsymbol{\omega}^t$ for faster convergence. The second refinement concerns the randomness of the RFF due to vector $\boldsymbol{\omega}^t$. So far, the latter was drawn according to the Fourier transform (line **5** of Algorithm 2) and then used to learn b^t. We suggest instead to fine-tune $\boldsymbol{\omega}^t$ by doing a simple gradient descent with as initialization the vector drawn according to the Fourier transform. Supported by the experiments performed in the following, we claim that such a strategy allows us to both speed up the convergence of the algorithm and boost the accuracy. This update requires to add a line of code, just after line **6** of Algorithm 2, expressed as a regularized optimization problem:

$$\boldsymbol{\omega}^t = \underset{\boldsymbol{\omega}\in\mathbb{R}^d}{\operatorname{argmin}}\ \lambda\|\boldsymbol{\omega}\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \exp\Big(-\tilde{y}_i \cos\Big(\boldsymbol{\omega}\cdot\mathbf{x}_i - b^t)\Big)\Big),$$

$$\text{its derivative being}\quad \frac{\partial f_{\boldsymbol{\omega}}}{\partial \boldsymbol{\omega}}(\boldsymbol{\omega}) = 2\lambda\boldsymbol{\omega} + \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\tilde{y}_i \sin(\boldsymbol{\omega}\cdot\mathbf{x}_i - b^t)\, \mathrm{e}^{-\tilde{y}_i \cos(\boldsymbol{\omega}\cdot\mathbf{x}_i - b^t)}.$$

4 Experimental Evaluation

The objective of this section is three-fold: first, we aim to bring to light the interest of learning the landmarks rather than a priori fixing them as done in Letarte *et al.* [7]; second we study the impact of the number K of random features; lastly, we perform an extensive experimental comparison of our algorithms[4] with some boosting-based and kernel-learning state-of-the-art methods.

[4] In case of acceptance, the code of the methods and experiments will be published.

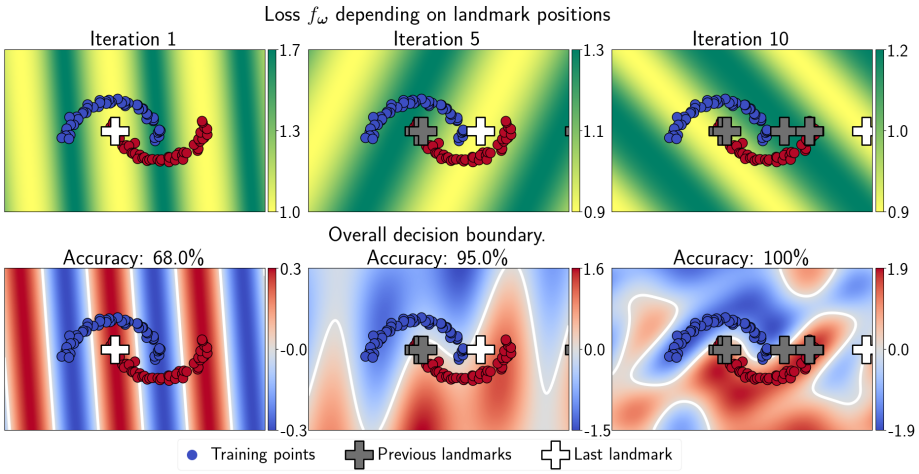

Fig. 1. GBRFF1 with K=1 on the two-moons dataset at different iterations. Top row shows the periodicity of the loss (light yellow indicates the minimal loss). Bottom row shows the resulting decision boundaries between the classes (blue & red) by fixing arbitrarily one coordinate of the landmark and minimizing the loss along the other one.

4.1 Setting

For **GBRFF1** (Algorithm 2), we select by cross-validation (CV) the hyper-parameter $\gamma \in \frac{2^{\{-2,\dots,2\}}}{d}$ with d the number of features of the training data. For **GBRFF2** (Algorithm 3), we tune $\lambda \in \{0, 2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}\}$. We compare our two methods with the following algorithms.

• **LGBM** [5] is a state-of-the-art gradient boosting method using trees as base predictors. We select by CV the maximum depth of the trees in $\{1, \dots, 5\}$.

• **BMKR** [16] is a Multiple Kernel Learning method based on gradient boosting with the least square loss. It selects at each iteration the best performing kernel plugged inside an SVR to learn the residuals. It considers at each iteration 10 RBF kernels with $\gamma \in 2^{\{-4,\dots,5\}}$ and the linear kernel $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$. We select by CV the SVR parameter $C \in 10^{\{-2,\dots,2\}}$.

• **GFC** [9] is a greedy feature construction method based on functional gradient descent. It iteratively refines the representation learned by adding a feature that matches the residual function defined for the least squared loss. We use the final representation to learn a linear SVM where $C \in 10^{\{-2,\dots,2\}}$ is selected by CV.

• **PBRFF** [7] first draws randomly with replacement a set of n_L landmarks from the training set, then learns the new representation where each new feature is computed using Equation (2) and finally learns a linear SVM on the mapped training set. We fix the number of random features to $K = 10$ and we draw them like for our two methods **GBRFF1** and **GBRFF2** from the normal law $\mathcal{N}(0, 2\gamma)^d$. We select by CV its parameters $\gamma \in \frac{2^{\{-2,\dots,2\}}}{d}$, $\beta \in 10^{\{-2,\dots,2\}}$ and the SVM parameter $C \in 10^{\{-2,\dots,2\}}$.

Table 1. Description of the datasets (n: number of examples, d: number of features, c: number of classes) and the classes chosen as negative (-1) and positive (+1).

Name	n	d	c	Label -1	Label +1	Name	n	d	c	Label -1	Label +1
wine	178	13	3	2, 3	1	australian	690	14	2	0	1
sonar	208	60	2	M	R	pima	768	8	2	0	1
newthyroid	215	5	3	1	2, 3	vehicule	846	18	4	van	bus, opel, saab
heart	270	13	2	1	2	german	1000	23	2	1	2
bupa	345	6	2	2	1	splice	3175	60	2	+1	-1
iono	351	34	2	g	b	spambase	4597	57	2	0	1
wdbc	569	30	2	B	M	occupancy	20560	5	2	0	1
balance	625	4	3	B, R	L	bankmarketing	45211	51	2	no	yes

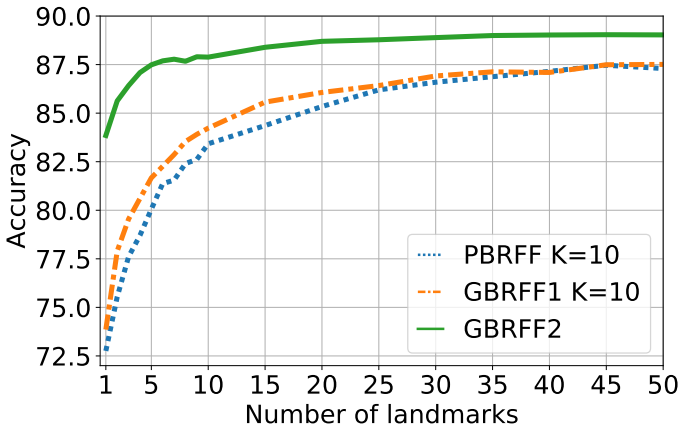

Fig. 2. Mean test accuracy over 20 train/test splits over the 16 datasets. We train the three methods using from 1 to 50 landmarks.

We consider 16 datasets coming mainly from the UCI repository. We binarized them as described in Table 1 where we specify the classes that are considered respectively as label '-1' and as label '+1'. For each dataset, we generate 20 random splits of 70% training examples and 30% testing samples. All datasets are scaled such that each feature in the training set has a mean of 0 and a variance of 1; the factors computed on the training set are then used to scale each feature in the test set. The hyper-parameters of the methods are tuned by a 5-fold CV on the training set by performing a grid search.

4.2 Influence of learning the landmarks

We present in Figure 2 the behavior of the three methods that make use of landmarks and RFF, that is **PBRFF**, **GBRFF1** and **GBRFF2**. With more than 25 landmarks, **PBRFF** and **GBRFF1** show similar mean accuracy and reach about 87.5% after 50 iterations. However, for a small set of landmarks (in particular smaller than 25) **GBRFF1** is consistently superior by about 1

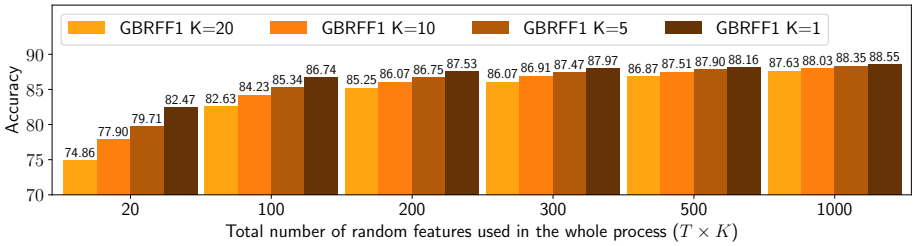

Fig. 3. Mean results over the 16 datasets *w.r.t.* the same total number of random features $T \times K$ for $K \in \{1, 5, 10, 20\}$, with T the number of boosting iterations.

point higher than **PBRFF**, showing the interest of learning the landmarks. But the certainly most striking result comes from the performance of our variant **GBRFF2** which outperforms the two competing methods. This is particularly true for a small number of landmarks. Notice that **GBRFF2** is able to reach its maximum with about 20 landmarks, while **GBRFF1** and **PBRFF** require more iterations without reaching the same performance. This definitely shows the benefit of learning the random features compared to drawing them randomly.

4.3 Influence of the number of random features

A key parameter of **GBRFF1** is K, the number of random features used at each iteration. To highlight its impact, we report in Figure 3 the mean test accuracy of **GBRFF1** with $K \in \{1, 5, 10, 20\}$ across all datasets and over the 20 train/test splits. To have a fair study, the comparison is performed according to the same total number of random features after the whole boosting process, that is $T \times K$ with T the number of iterations. First of all, we observe that with a total of $1,000$ random features, K does not have a big impact on the performance. However, when decreasing the value of $T \times K$, we can note that it becomes much more interesting in terms of accuracy to set K to a small value. This shows that the more we want a compact final representation, the more we need to refine the random features: it is better to weight each of the features greedily with α^t (line **8** of Algorithm 3) rather than using the closed-form solution of Equation (10) (line **7** of Algorithm 2) to weight them all at once. Even if in the usual context of RFF it is desirable to have a large K value to approximate a kernel, this series of experiments shows that a simple rough approximation with K=1 along with a sufficient number of iterations allow the final ensemble to mimic the approximation of a new kernel suited for the task at hand.

4.4 Influence of the number of samples on the computation time

The specificities of **GBRFF2** come from the number of random features K set to 1 at each iteration and the learning of $\boldsymbol{\omega}^t$. We already shown in Figure 2 that this allows us to get better results. We study in this section how **GBRFF2**

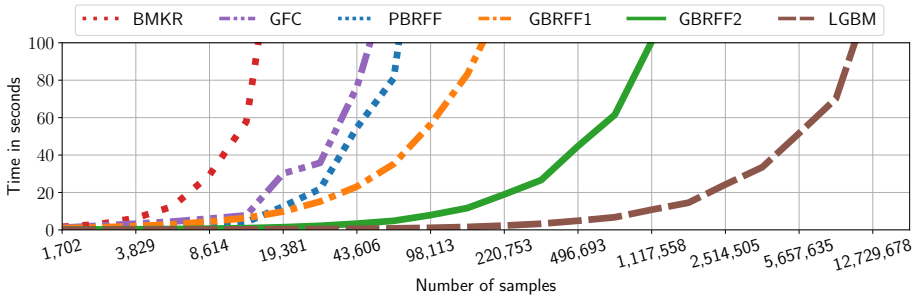

Fig. 4. Computation time in seconds required to train and test the six compared methods with fixed parameters on an artificial dataset having an increasing number of samples. Half of the dataset is used for training and half for testing and a method requiring more than 100 seconds at a given step is not trained on the larger datasets.

scales compared to the other methods. To do so, we consider artificial datasets with an increasing number of samples (generated with scikit-learn [11] library's `make_classification` function). The initial size is set to 150 samples, and we successively generate datasets with a size 50% larger than the previous one. Here, we randomly split the datasets into 50% train / 50% test and we report the time in seconds necessary to train the models and to predict the labels of the test examples. The parameters are fixed as follows: $C = 1$ for the methods using SVM or SVR; the tree depth is set to 5 for **LGBM**; $K = 10$, $\gamma = \frac{1}{d}$, and $\beta = 1$ for **PBRFF** and **GBRFF1**; $\gamma = \frac{1}{d}$ and $\lambda = 0$ for **GBRFF2**. Note that all the methods are run with 100 iterations (or landmarks) and for a maximum execution time of 100 seconds. We report the results in Figure 4.

We first recall that **GBRFF2** learns at each iteration a random feature and a landmark while **GBRFF1** only learns the landmark and **PBRFF** draws them randomly. Thus, **GBRFF1** should present higher computation times compared to **PBRFF**. However, we can note that for datasets with a number of samples larger than $20,000$, **GBRFF1** becomes cheaper than **PBRFF**. This is due to the fact that the SVM classifier learned by **PBRFF** does not scale as well as gradient boosting-based methods. The two-step method **GFC** is in addition also slower than **GBRFF1**. This shows the computational advantage of having a one-step procedure to learn both the representation and the final classifier. When looking at the time limit of 100 seconds, both **GBRFF1** and **GBRFF2** are the fastest kernel-based methods compared to **BMKR**, **GFC** and **PBRFF**. This shows the efficiency of learning kernels in a greedy fashion. We also see that **GBRFF2** performs faster than **GBRFF1** for any number of samples. At the limit of 100 seconds, it is able to deal with datasets that are 10 times larger that **GBRFF1**, due to the lower complexity of the learned weak learner used in **GBRFF2**. Finally, **GBRFF2** is globally the second fastest method behind the gradient boosting method **LGBM** that uses trees as base classifiers.

Table 2. Mean test accuracy ± standard deviation over 20 random train/test splits. A '-' in the last row indicates that the algorithm did not converge in time on this dataset. Average ranks and mean results are computed over the 15 first datasets.

Dataset	BMKR	GFC	PBRFF	GBRFF1	LGBM	GBRFF2
wine	**99.5** ± 1.0	99.3 ± 1.1	98.1 ± 2.1	98.3 ± 1.5	96.5 ± 3.0	98.5 ± 1.6
sonar	78.8 ± 7.2	76.6 ± 3.2	76.7 ± 5.2	81.8 ± 3.5	**83.0** ± 4.2	83.0 ± 5.0
newthyroid	96.5 ± 1.7	96.5 ± 2.1	96.5 ± 1.5	95.3 ± 2.2	94.8 ± 3.0	**96.9** ± 2.1
heart	**85.6** ± 4.0	79.4 ± 4.5	85.4 ± 3.5	83.6 ± 4.0	84.0 ± 3.2	83.1 ± 4.0
bupa	68.1 ± 4.9	64.7 ± 3.2	69.0 ± 4.2	70.3 ± 4.9	**71.9** ± 4.0	71.2 ± 4.5
iono	**94.2** ± 1.4	91.5 ± 2.3	94.2 ± 1.8	88.2 ± 2.3	93.2 ± 2.8	89.2 ± 2.1
wdbc	96.1 ± 1.2	95.8 ± 1.3	96.5 ± 1.1	96.8 ± 1.1	95.7 ± 1.4	**97.3** ± 1.2
balance	96.0 ± 1.2	95.1 ± 2.0	**98.9** ± 1.1	97.7 ± 0.7	93.5 ± 2.6	97.7 ± 0.6
australian	85.9 ± 2.0	80.9 ± 2.4	84.6 ± 2.3	86.7 ± 1.7	85.9 ± 1.8	**86.9** ± 1.9
pima	76.4 ± 2.0	68.7 ± 2.6	76.1 ± 2.5	76.5 ± 2.7	76.0 ± 2.7	**77.1** ± 2.5
vehicle	96.6 ± 1.3	95.9 ± 0.8	96.5 ± 1.4	96.3 ± 1.2	96.6 ± 1.0	**97.1** ± 1.0
german	72.3 ± 1.8	64.3 ± 2.8	72.4 ± 1.4	73.7 ± 1.6	73.4 ± 1.6	**74.0** ± 1.3
splice	87.5 ± 1.0	87.0 ± 1.0	83.5 ± 0.7	83.9 ± 1.1	**96.9** ± 0.5	92.4 ± 0.8
spambase	93.5 ± 0.4	91.3 ± 0.6	91.6 ± 0.7	90.7 ± 0.7	**95.2** ± 0.7	92.8 ± 0.6
occupancy	**99.3** ± 0.1	98.9 ± 0.7	98.9 ± 0.1	98.8 ± 0.1	99.2 ± 0.1	98.9 ± 0.1
Mean	88.4 ± 2.1	85.7 ± 2.0	87.9 ± 2.0	87.9 ± 2.0	**89.1** ± 2.2	**89.1** ± 2.0
Average Rank	2.88	4.94	3.75	3.88	3.31	2.25
bankmarketing	-	-	-	89.7 ± 0.2	**90.8** ± 0.2	90.0 ± 0.2

4.5 Performance comparison between all methods

Table 2 presents for each dataset the mean results over the 20 splits using 100 iterations/landmarks for each method. Due to the size of the dataset "bankmarketing", we do not report the results of the algorithms that do not converge in time for this dataset, and we compute the average ranks and mean results over the other 15 datasets. In terms of accuracy, **GBRFF2** shows very good results compared with the state-of-the-art as it obtains the best average rank among the six methods and on average the best mean accuracy (with **LGBM**) leaving apart "bankmarketing". Interestingly, our method is the only kernel-based one that scales well enough to be applied to this latter dataset.

4.6 Comparison of LGBM and GBRFF2 on toy datasets

In this last experiment, we focus on **LGBM** and **GBRFF2** which have been shown to be the two best performing methods. We consider three synthetics 2D datasets with non-linearly separable classes. The first one, called "swiss", represents two spirals of two classes side by side. The second one, namely "circles", consists of four circles with the same center and an increasing radius by alternating the class of each circle. The third dataset, called "board", consists of a four by four checkerboard with alternating classes in each cell. Note that both **LGBM** and **GBRFF2** are run for 1000 iterations allowing them to converge.

Figure 5 gives evidence that **GBRFF2** is able to achieve better results than **LGBM** using only a small number of training examples, *i.e.*, 500 or less. The performances are asymptotically similar for both methods on the board and circle datasets with a faster rate of convergence for **GBRFF2**. Furthermore, if we

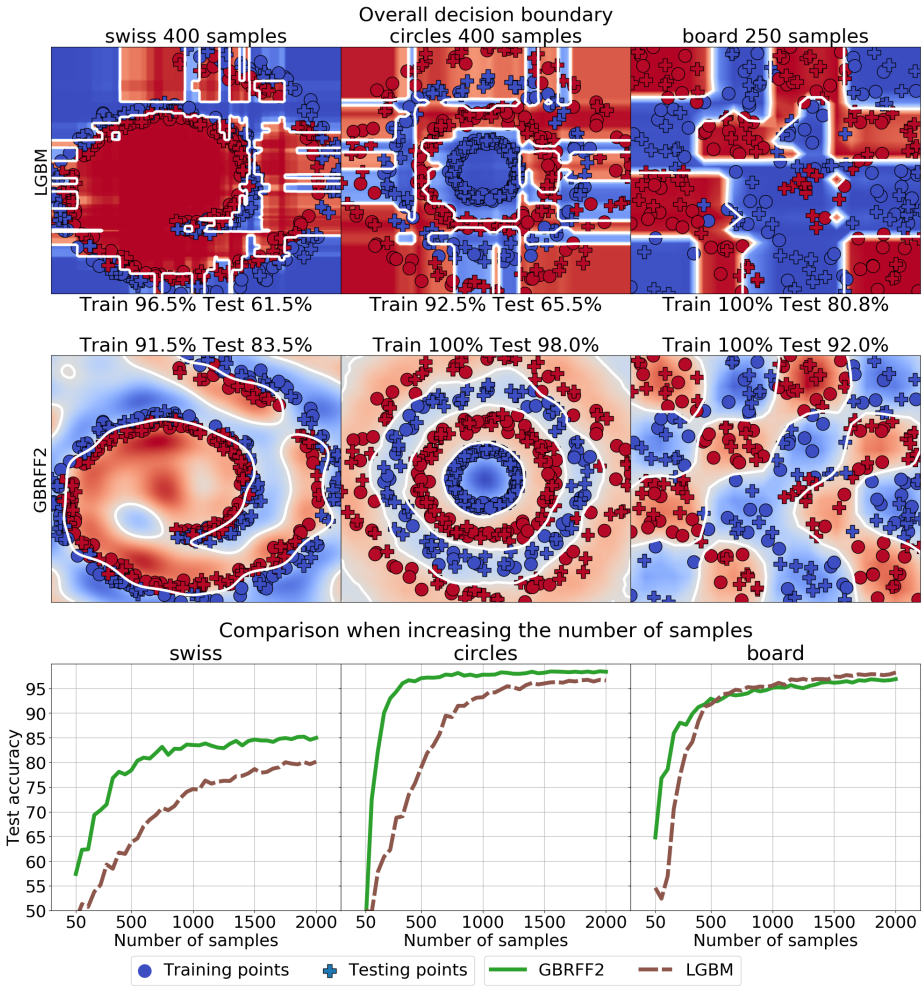

Fig. 5. Comparison of **LGBM** and **GBRFF2** on three synthetic datasets in terms of classification accuracy and decision boundaries (upper part of the figure) and in terms of performance *w.r.t.* the number of examples (last row of plots).

look at the decision boundaries and their associated performances at train and test time, we can see that **LGBM** is prone to overfit the training data compared to our approach, showing a drastic drop in performance between learning and testing. The learned decision boundaries are also smoother with **GBRRF2** than with **LGBM**. These experiments show the advantage of having a non linear weak learner in a gradient boosting approach.

5 Conclusion and Perspectives

In this paper, we take advantages of two machine learning approaches, gradient boosting and random Fourier features, to derive a novel algorithm that jointly learns a compact representation and a model based on random features. Building on a recent work [7], we learn a kernel by approximating it as a weighted sum of RFF [12]. The originality is that we learn such kernels so that the representation and the classifier are jointly optimized. We show that we can benefit from a performance boost in terms of accuracy and computation time by considering each weak learner as a single trigonometric feature and learning the random part of the RFF. The experimental study shows the competitiveness of our method with state-of-the-art boosting and kernel learning methods.

So far, the random features have been learned with a simple L2 regularization. A promising future line of research is to add a regularization on the random feature to foster diversity. In addition, the optimization of the random feature and of the landmark at each iteration can be computationally expensive when the number of iterations is large. A possibility to speed-up the learning procedure is to derive other kernel approximations where these two parameters can be computed with a closed-form solution. Other perspectives regarding the scalability include the use of standard gradient boosting tricks [5] such as sampling or learning the kernels in parallel.

References

1. Agrawal, R., Campbell, T., Huggins, J., Broderick, T.: Data-dependent compression of random features for large-scale kernel approximation. In: AISTATS (2019)
2. Balcan, M., Blum, A., Srebro, N.: Improved guarantees for learning via similarity functions. In: COLT (2008)
3. Drineas, P., Mahoney, M.: On the nyström method for approximating a gram matrix for improved kernel-based learning. JMLR **6**, 2153–2175 (2005)
4. Friedman, J.: Greedy function approximation: a gradient boosting machine. Ann. Statist. pp. 1189–1232 (2001)
5. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: NeurIPS (2017)
6. Le, Q., Sarlós, T., Smola, A.: Fastfood-computing hilbert space expansions in loglinear time. In: ICML (2013)
7. Letarte, G., Morvant, E., Germain, P.: Pseudo-bayesian learning with kernel fourier transform as prior. In: AISTATS (2019)
8. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Functional gradient techniques for combining hypotheses. In: NeurIPS (1999)
9. Oglic, D., Gärtner, T.: Greedy feature construction. In: NeurIPS (2016)
10. Oliva, J., Dubey, A., Wilson, A., Póczos, B., Schneider, J., Xing, E.: Bayesian nonparametric kernel-learning. In: AISTATS (2016)
11. Pedregosa, F.*et al.*.: Scikit-learn: Machine learning in Python. JMLR **12**, 2825–2830 (2011)
12. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: NeurIPS (2008)

13. Sinha, A., Duchi, J.: Learning kernels with random features. In: NeurIPS (2016)
14. Vincent, P., Bengio, Y.: Kernel matching pursuit. Mach. Learn. **48**(1-3), 165–187 (2002)
15. Williams, C., Seeger, M.: Using the nyström method to speed up kernel machines. In: NeurIPS (2001)
16. Wu, D., Wang, B., Precup, D., Boulet, B.: Boosting based multiple kernel learning and transfer regression for electricity load forecasting. In: ECML-PKDD (2017)
17. Yang, Z., Wilson, A., Smola, A., Song, L.: A la carte–learning fast kernels. In: AISTATS (2015)
18. Zantedeschi, V., Emonet, R., Sebban, M.: Fast and provably effective multi-view classification with landmark-based svm. In: ECML-PKDD (2018)