



# Landmark-Based Ensemble Learning with Random Fourier Features and Gradient Boosting

Léo Gautheron<sup>1</sup>(✉), Pascal Germain<sup>2</sup>, Amaury Habrard<sup>1</sup>, Guillaume Metzler<sup>1</sup>,  
Emilie Morvant<sup>1</sup>, Marc Sebban<sup>1</sup>, and Valentina Zantedeschi<sup>3</sup>

<sup>1</sup> Univ Lyon, UJM-Saint-Etienne, CNRS, Institut d'Optique Graduate School,  
Laboratoire Hubert Curien UMR 5516, 42023 Saint-Etienne, France  
{leo.gautheron, amaury.habrard, guillaume.metzler, emilie.morvant,  
marc.sebban}@univ-st-etienne.fr, leo.gautheron@hotmail.fr

<sup>2</sup> Département d'informatique et de génie logiciel, Université Laval, Québec, Canada  
pascal.germain@ift.ulaval.ca

<sup>3</sup> GE - Global Research, 1 Research Circle, Niskayuna, NY 12309, USA  
vzantedeschi@gmail.com

**Abstract.** This paper jointly leverages two state-of-the-art learning strategies—gradient boosting (GB) and kernel Random Fourier Features (RFF)—to address the problem of kernel learning. Our study builds on a recent result showing that one can learn a distribution over the RFF to produce a new kernel suited for the task at hand. For learning this distribution, we exploit a GB scheme expressed as ensembles of RFF weak learners, each of them being a kernel function designed to fit the residual. Unlike Multiple Kernel Learning techniques that make use of a pre-computed dictionary of kernel functions to select from, at each iteration we fit a kernel by approximating it from the training data as a weighted sum of RFF. This strategy allows one to build a classifier based on a small ensemble of learned kernel “landmarks” better suited for the underlying application. We conduct a thorough experimental analysis to highlight the advantages of our method compared to both boosting-based and kernel-learning state-of-the-art methods.

**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

---

**Electronic supplementary material** The online version of this chapter ([https://doi.org/10.1007/978-3-030-67664-3\\_9](https://doi.org/10.1007/978-3-030-67664-3_9)) contains supplementary material, which is available to authorized users.

simplicity. However, they exhibit two main flaws in terms of memory usage and time complexity. Landmark-based kernel approaches [2] 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 [13] and Matching Pursuit methods [12] 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 [3] or Random Fourier Features (RFF) [10]. The latter is probably the most used thanks to its simplicity and ease of computation. It allows the approximation of any shift-invariant kernel based on the Fourier transform of the kernel. Several works have extended this technique by allowing one to adapt the RFF approximation directly from the training data [1, 6, 11]. Among them, the recent work of Letarte *et al.* [6] 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. However, in practice, this method requires the use of a fixed set of landmarks selected beforehand and independently from the task before being able to learn the representation in a second step. 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 these 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 [8], which at each iteration of the functional gradient descent [7] 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 Sect. 3 as well as two efficient refinements before presenting an extensive experimental study in Sect. 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  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$  be a training set of  $n$  points.

We focus on kernel-based algorithms that rely on pre-defined kernel functions  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  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 [13] 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 of various kernels associated with some fixed parameters, and a combination is learned, defined as

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

with  $\alpha^t \in \mathbb{R}$  the weight of the kernel  $k^t(\mathbf{x}, \mathbf{x}')$ . As shown in Sect. 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 Sect. 3.1). To avoid the dictionary of kernel functions in Eq. (1) from being pre-computed, we propose a method inspired from Letarte *et al.* [6] 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 [10] defined as

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

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

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

When  $q^t$  is uniform, we retrieve the setting of RFF and we have  $k(\boldsymbol{\delta}) \simeq k_{q^t}(\boldsymbol{\delta})$  where larger number of random features  $K$  give better approximations [10]. Letarte *et al.* [6] aim to learn the weights of the random Fourier features  $q^t$ . To do so, they consider a loss function  $\ell$  that measures the quality of the similarities computed using the kernel  $k_{q^t}$ . Their theoretical study on  $\ell$  leads to a closed-form solution for  $q^t$  computed as

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

---

**Algorithm 1:** Gradient boosting [4]

---

**Inputs :** Training set  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ ; Loss  $\ell$ ; Number of iterations  $T$ **Output:**  $\text{sign}\left(H^0(\mathbf{x}) + \sum_{t=1}^T \alpha^t h_{a^t}(\mathbf{x})\right)$ 1:  $\forall i = 1, \dots, n, \quad H^0(\mathbf{x}_i) = \underset{\rho}{\text{argmin}} \sum_{i=1}^n \ell(y_i, \rho)$ 2: **for**  $t = 1, \dots, T$  **do**3:  $\forall i = 1, \dots, n, \quad \tilde{y}_i = -\frac{\partial \ell(y_i, H^{t-1}(\mathbf{x}_i))}{\partial H^{t-1}(\mathbf{x}_i)}$ 4:  $a^t = \underset{a}{\text{argmin}} \sum_{i=1}^n (\tilde{y}_i - h_a(\mathbf{x}_i))^2$ 5:  $\alpha^t = \underset{\alpha}{\text{argmin}} \sum_{i=1}^n \ell(y_i, H^{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$ 6:  $\forall i = 1, \dots, n, \quad H^t(\mathbf{x}_i) = H^{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$ 7: **end for**

---

with  $\beta \geq 0$  a parameter to tune,  $h_{\omega}^t(\mathbf{x}) = \cos(\omega \cdot (\mathbf{x}^t - \mathbf{x}))$ , 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

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

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  $\alpha^t$  is the weight associated

**Algorithm 2: GBRFF1**


---

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

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

- 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**  $t = 1, \dots, T$  **do**
- 3:  $\forall i = 1, \dots, n, \quad w_i = \exp(-y_i H^{t-1}(\mathbf{x}_i))$
- 4:  $\forall i = 1, \dots, n, \quad \tilde{y}_i = y_i w_i$
- 5:  $\forall j = 1, \dots, K$ , draw  $\boldsymbol{\omega}_j^t \sim \mathcal{N}(0, 2\gamma)^d$
- 6:  $\mathbf{x}^t = \underset{\mathbf{x} \in \mathbb{R}^d}{\text{argmin}} \frac{1}{n} \sum_{i=1}^n \exp\left(-\tilde{y}_i \frac{1}{K} \sum_{j=1}^K \cos(\boldsymbol{\omega}_j^t \cdot (\mathbf{x} - \mathbf{x}_i))\right)$
- 7:  $\forall j = 1, \dots, K, \quad q_j^t = \frac{1}{2^t} \exp\left(\frac{-\beta\sqrt{n}}{n} \sum_{i=1}^n \exp\left(-\tilde{y}_i \cos(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i))\right)\right)$
- 8:  $\alpha^t = \frac{1}{2} \ln \frac{\sum_{i=1}^n \left(1 + y_i \sum_{j=1}^K q_j^t \cos(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i))\right) w_i}{\sum_{i=1}^n \left(1 - y_i \sum_{j=1}^K q_j^t \cos(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i))\right) w_i}$
- 9:  $\forall i = 1, \dots, n, \quad H^t(\mathbf{x}_i) = H^{t-1}(\mathbf{x}_i) + \alpha^t \sum_{j=1}^K q_j^t \cos(\boldsymbol{\omega}_j^t \cdot (\mathbf{x}^t - \mathbf{x}_i))$
- 10: **end for**

---

to the predictor  $h_{\alpha^t}$  and is learned at the same time as the parameters  $\alpha^t$  of that classifier. Given a differentiable loss  $\ell$ , 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  $h_{\alpha^t}$  that fits the best the residuals (line 4), before learning the optimal step size  $\alpha^t$  that minimizes the loss by adding  $h_{\alpha^t}$ , weighted by  $\alpha^t$ , to the current vote (line 5). Finally, the model is updated by adding  $\alpha^t h_{\alpha^t}(\cdot)$  (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 which follows the steps of Algorithm 1. The loss function  $\ell$  at the core of our algorithm is the exponential loss:

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

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

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

The residuals of line **3** are defined as  $\tilde{y}_i = -\frac{\partial \ell(y_i, H^{t-1}(\mathbf{x}_i))}{\partial H^{t-1}(\mathbf{x}_i)} = y_i e^{-y_i H^{t-1}(\mathbf{x}_i)}$ .

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 (Eq. (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:

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

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

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

where its parameters are given by  $a^t = (\{\boldsymbol{\omega}_j^t\}_{j=1}^K, \mathbf{x}^t, q^t)$ . Instead of using a pre-defined set of landmarks [6], we build this set iteratively, *i.e.*, we learn one landmark per iteration. To benefit from the closed form of Eq. (4), we propose the following greedy approach to learn the parameters  $a^t$ . At each iteration  $t$ , we draw  $K$  vectors  $\{\boldsymbol{\omega}_j^t\}_{j=1}^K \sim p^K$  with  $p$  the Fourier transform of a given kernel (as defined in Eq. (3)); then we look for the optimal landmark  $\mathbf{x}^t$ . Plugging Eq. (8) into Eq. (7) and assuming a uniform prior distribution over the random features,  $\mathbf{x}^t$  is learned to minimize

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

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 Eq. (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.* [6], given the landmark  $\mathbf{x}^t$  found by gradient descent, we can now compute the weights of the random features  $q^t$  as

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

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

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

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

where  $w_i = \exp(-y_i H^{t-1}(\mathbf{x}_i))$ . In order to have a closed-form solution of  $\alpha$ , 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

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

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

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

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

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

As usually done in the RFF literature [1, 10, 11] we use the RBF kernel  $k_{\gamma}(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2}$  with as Fourier transform vectors of  $d$  numbers each drawn from the normal law with zero mean and variance  $2\gamma$  that we denote  $\mathcal{N}(0, 2\gamma)^d$ .

### 3.3 Refining GBRFF1

In **GBRFF1**, the number of random features  $K$  used at each iteration has a direct impact on the computation time of the algorithm. Moreover  $\omega^t$  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  $\omega^t$  after a random initialization from the Fourier transform. We show that a simple gradient descent with respect 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

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

**Algorithm 3: GBRFF2**

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

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

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

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

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

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

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

We leverage the periodicity of the cosine function along each direction to find the optimal  $c$ -th coordinate of the landmark  $\mathbf{x}_c^t \in \left[\frac{-\pi}{\boldsymbol{\omega}_c^t}, \frac{\pi}{\boldsymbol{\omega}_c^t}\right]$  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 loss). There is an infinite number of landmarks giving such a minimal loss at the middle of the yellow stripes. Thus,



by setting one coordinate of the landmark to an arbitrary value, 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 dataset after 10 iterations. By generalizing this, instead of learning a landmark vector  $\mathbf{x}^t \in \mathbb{R}^d$ , we fix all but one coordinate of the landmark to 0, and then learn a single scalar  $b^t \in [-\pi, \pi]$  that minimizes

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

**Learning  $\omega^t$  for Faster Convergence.** The second refinement concerns the randomness of the RFF due to vector  $\omega^t$ . So far, the latter was drawn according  $p$  and then used to learn  $b^t$ . We suggest instead to fine-tune  $\omega^t$  by doing a gradient descent with as initialization the vector drawn from  $p$ . 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:

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

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

## 4 Experimental Evaluation

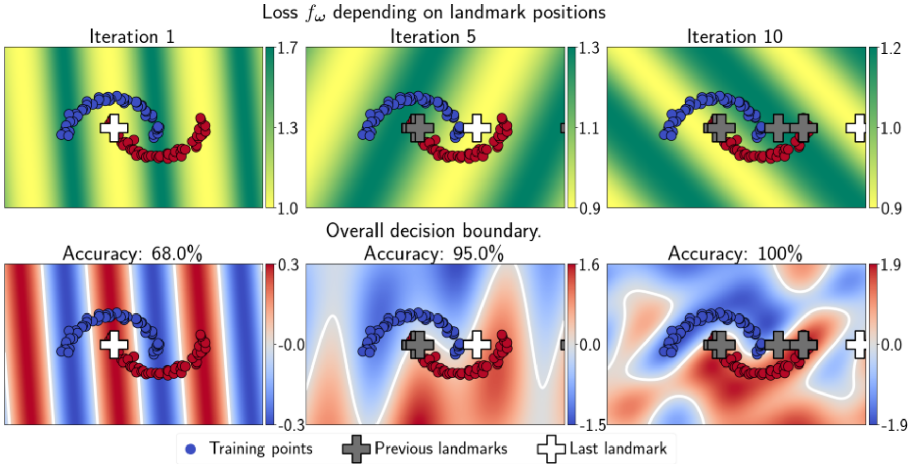
The objective of this section is three-fold: first, we aim to bring to light the interest of learning the landmarks rather than fixing them as done in Letarte *et al.* [6]; second we study the impact of the number  $K$  of random features; lastly, we perform an extensive experimental comparison of our algorithms. The Python code of all experiments and the data used are publicly available<sup>1</sup>.

### 4.1 Setting

For **GBRFF1** and **GBRFF2**, we select by cross-validation (CV) the hyperparameter  $\gamma \in \frac{2^{\{-2, \dots, 2\}}}{d}$ . For **GBRFF2**, we also tune  $\lambda \in \{0, 2^{\{-5, \dots, -2\}}\}$ . 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 tree depth in  $\{1, \dots, 10\}$  and the L2 regularization parameter  $\lambda \in \{0, 2^{\{-5, \dots, -2\}}\}$ .

<sup>1</sup> The code is available here: <https://leogautheron.github.io>.



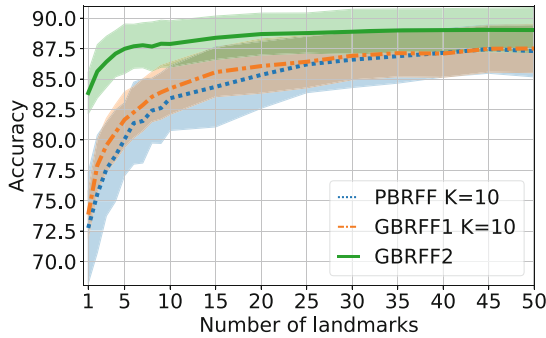
**Fig. 1.** GBRFF1 with  $K=1$  on the two-moons dataset at different iterations. Top row shows the periodicity of the loss (light yellow indicates the minimal loss). Bottom row shows the resulting decision boundaries between the classes (blue & red) by fixing arbitrarily one coordinate of the landmark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

- **BMKR** [13] is a Multiple Kernel Learning method based on gradient boosting with least square loss. It selects at each iteration the best kernel plugged inside an SVR to fit the residuals among 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** [8] 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** [6] that (1) draws with replacement  $n_L$  landmarks from the training set; (2) learns a representation of  $n_L$  features where each feature is computed using Eq. (2) based on  $K=10$  vectors drawn like our methods from  $\mathcal{N}(0, 2\gamma)^d$ ; (3) learns a linear SVM on the new representation. We select by CV its parameters  $\gamma \in \frac{2^{\{-2, \dots, 2\}}}{d}$ ,  $\beta \in 10^{\{-2, \dots, 2\}}$  and the SVM parameter  $C \in 10^{\{-2, \dots, 2\}}$ .

We consider 16 datasets coming mainly from the UCI repository that we binarized as described in Table 1. We generate for each dataset 20 random 70%/30% train/test splits. Datasets are pre-processed such that each feature in the training set has 0 mean and unit variance; the factors computed on the training set are then used to scale each feature in the test set. All parameters are tuned by 5-fold CV on the training set by performing a grid search.

**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

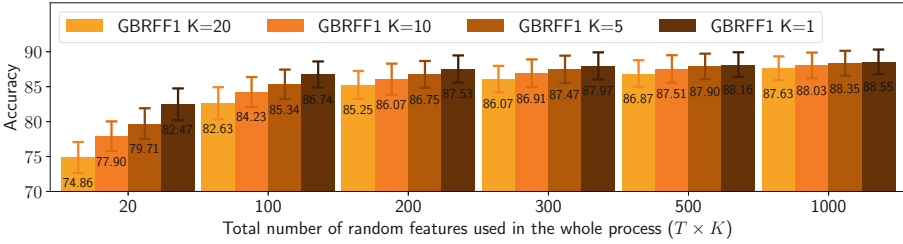
**Fig. 2.** Mean test accuracy over 20 train/test splits over the 16 datasets. We train the three methods using from 1 to 50 landmarks.

## 4.2 Influence of Learning the Landmarks

We present in Fig. 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 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 amount 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 Fig. 3 the mean test accuracy of

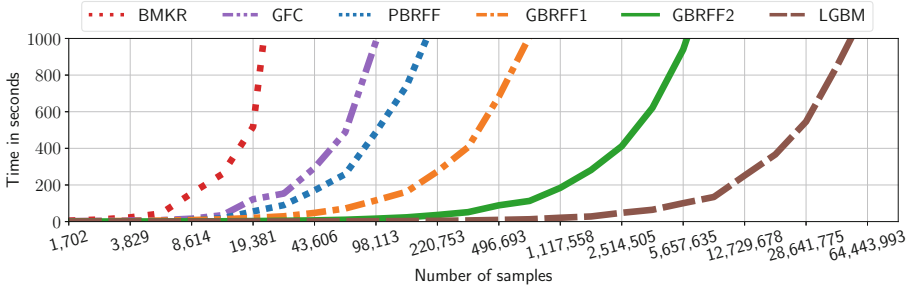


**Fig. 3.** Mean results over the 16 datasets *w.r.t.* the same total number of random features  $T \times K$  for  $K \in \{1, 5, 10, 20\}$ , with  $T$  the number of boosting iterations.

**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  $T \times K$ , 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  $\alpha^t$  (line 8 of Algorithm 3) rather than using the closed-form solution of Eq. (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 allows 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  $\omega^t$ . We have already shown in Fig. 2 that this allows us to get better results. We study in this section how **GBRFF2** scales compared to the other methods. To do so, we consider artificial datasets with an increasing number of samples (generated with scikit-learn [9] library’s `make_classification` function). The initial size is set to 150 samples, and we successively generate datasets with a size equal to the previous dataset size multiplied by 1.5. Here, we do not split the datasets in train and test as we are not interested in the accuracy. We report the time in seconds necessary to train the models and to predict the labels on the whole datasets. 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**. All the methods are run with 100 iterations (or landmarks) and are not run on datasets requiring more than 1000 s of execution time (because larger datasets requiring more than 1000 s by the fastest method



**Fig. 4.** Computation time in seconds required to train and test the six methods with fixed parameters on an artificial dataset having an increasing number of samples. The whole dataset is used for training and testing, and a method requiring more than 1000 seconds at a given step is not trained on the larger datasets.

do not fit in the RAM memory of the computer used for the experiments). We report the results in Fig. 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, 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 1000 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 1000 seconds, it is able to deal with datasets that are 10 times larger than **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.

#### 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 leaving apart

**Table 2.** Mean test accuracy  $\pm$  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 first 15 datasets.

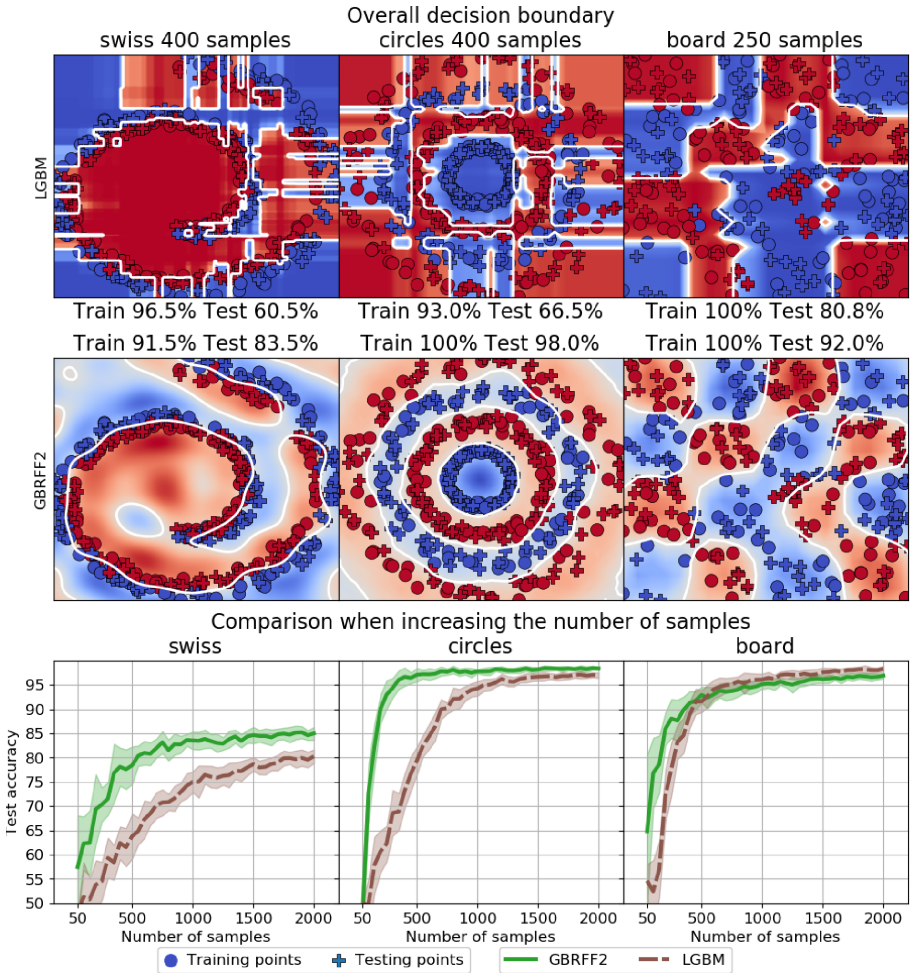
Dataset	BMKR	GFC	PBRFF	GBRFF1	LGBM	GBRFF2
Wine	<b>99.5</b> $\pm$ 1.0	99.3 $\pm$ 1.1	98.1 $\pm$ 2.1	98.3 $\pm$ 1.5	96.6 $\pm$ 3.2	98.5 $\pm$ 1.6
Sonar	78.8 $\pm$ 7.2	76.6 $\pm$ 3.2	76.7 $\pm$ 5.2	81.8 $\pm$ 3.5	82.4 $\pm$ 4.3	<b>83.0</b> $\pm$ 5.0
Newthyroid	96.5 $\pm$ 1.7	96.5 $\pm$ 2.1	96.5 $\pm$ 1.5	95.3 $\pm$ 2.2	94.8 $\pm$ 2.9	<b>96.9</b> $\pm$ 2.1
Heart	<b>85.6</b> $\pm$ 4.0	79.4 $\pm$ 4.5	85.4 $\pm$ 3.5	83.6 $\pm$ 4.0	83.0 $\pm$ 3.5	83.1 $\pm$ 4.0
Bupa	68.1 $\pm$ 4.9	64.7 $\pm$ 3.2	69.0 $\pm$ 4.2	70.3 $\pm$ 4.9	<b>72.0</b> $\pm$ 3.3	71.2 $\pm$ 4.5
Iono	<b>94.2</b> $\pm$ 1.4	91.5 $\pm$ 2.3	94.2 $\pm$ 1.8	88.2 $\pm$ 2.3	93.3 $\pm$ 2.5	89.2 $\pm$ 2.1
Wdbc	96.1 $\pm$ 1.2	95.8 $\pm$ 1.3	96.5 $\pm$ 1.1	96.8 $\pm$ 1.1	95.8 $\pm$ 1.5	<b>97.3</b> $\pm$ 1.2
Balance	96.0 $\pm$ 1.2	95.1 $\pm$ 2.0	<b>98.9</b> $\pm$ 1.1	97.7 $\pm$ 0.7	93.5 $\pm$ 2.6	97.7 $\pm$ 0.6
Australian	85.9 $\pm$ 2.0	80.9 $\pm$ 2.4	84.6 $\pm$ 2.3	86.7 $\pm$ 1.7	85.5 $\pm$ 1.9	<b>86.9</b> $\pm$ 1.9
Pima	76.4 $\pm$ 2.0	68.7 $\pm$ 2.6	76.1 $\pm$ 2.5	76.5 $\pm$ 2.7	75.5 $\pm$ 2.7	<b>77.1</b> $\pm$ 2.5
Vehicle	96.6 $\pm$ 1.3	95.9 $\pm$ 0.8	96.5 $\pm$ 1.4	96.3 $\pm$ 1.2	96.7 $\pm$ 1.0	<b>97.1</b> $\pm$ 1.0
German	72.3 $\pm$ 1.8	64.3 $\pm$ 2.8	72.4 $\pm$ 1.4	73.7 $\pm$ 1.6	73.5 $\pm$ 1.7	<b>74.0</b> $\pm$ 1.3
Splice	87.5 $\pm$ 1.0	87.0 $\pm$ 1.0	83.5 $\pm$ 0.7	83.9 $\pm$ 1.1	<b>97.0</b> $\pm$ 0.5	92.4 $\pm$ 0.8
Spambase	93.5 $\pm$ 0.4	91.3 $\pm$ 0.6	91.6 $\pm$ 0.7	90.7 $\pm$ 0.7	<b>95.6</b> $\pm$ 0.4	92.8 $\pm$ 0.6
Occupancy	<b>99.3</b> $\pm$ 0.1	98.9 $\pm$ 0.7	98.9 $\pm$ 0.1	98.8 $\pm$ 0.1	99.3 $\pm$ 0.1	98.9 $\pm$ 0.1
Mean	88.4 $\pm$ 2.1	85.7 $\pm$ 2.0	87.9 $\pm$ 2.0	87.9 $\pm$ 2.0	89.0 $\pm$ 2.1	<b>89.1</b> $\pm$ 2.0
Average Rank	2.88	4.94	3.75	3.81	3.44	<b>2.19</b>
Bankmarketing	-	-	-	89.7 $\pm$ 0.2	<b>90.8</b> $\pm$ 0.2	90.0 $\pm$ 0.2

“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 in terms of accuracy and execution time. Even if **BMKR** is among the three best methods in terms of accuracy, we do not consider it for this experiment due to its poor execution time. Learning a classifier based on non-linear kernels through **GBRFF2** has the advantage of being able to capture non-linear decision surfaces, whereas **LGBM** is not well suited for this because it uses trees as base learner. To illustrate this advantage, 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. Here, both **LGBM** and **GBRFF2** are run for 1000 iterations to ensure their convergence and parameters are tuned by CV as previously.

Figure 5 gives evidence that **GBRFF2** is able to achieve better results than **LGBM** using only a small amount 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 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 **GBRFF2** than with **LGBM**. These experiments show the advantage of having a non-linear weak learner in a gradient boosting approach.



**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).

## 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 [6], we learn a kernel by approximating it as a weighted sum of RFF [10]. 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.

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 promising future line of research to speed-up the learning 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.

**Acknowledgements.** Work supported in part by French projects APRIORI ANR-18-CE23-0015, LIVES ANR-15-CE23-0026 and IDEXLYON ACADEMICS ANR-16-IDEX-0005, and in part by the Canada CIFAR AI Chair Program.

## References

1. Agrawal, R., Campbell, T., Huggins, J., Broderick, T.: Data-dependent compression of random features for large-scale kernel approximation. In: the 22nd International Conference on Artificial Intelligence and Statistics, pp. 1822–1831 (2019)
2. Balcan, M., Blum, A., Srebro, N.: Improved guarantees for learning via similarity functions. In: the 21st Annual Conference on Learning Theory, pp. 287–298 (2008)
3. Drineas, P., Mahoney, M.W.: On the Nyström method for approximating a gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.* **6**, 2153–2175 (2005)
4. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. *Ann. Stat.* **29**, 1189–1232 (2001)
5. Ke, G., et al.: Lightgbm: a highly efficient gradient boosting decision tree. In: *Advances in Neural Information Processing Systems*, pp. 3146–3154 (2017)
6. Letarte, G., Morvant, E., Germain, P.: Pseudo-Bayesian learning with kernel fourier transform as prior. In: *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 768–776 (2019)
7. Mason, L., Baxter, J., Bartlett, P.L., Frean, M.: Functional gradient techniques for combining hypotheses. In: *Advances in Neural Information Processing Systems*, pp. 221–246 (1999)
8. Oglic, D., Gärtner, T.: Greedy feature construction. In: *Advances in Neural Information Processing Systems*, pp. 3945–3953 (2016)
9. Pedregosa, F., et al.: Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)



10. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: *Advances in Neural Information Processing Systems*, pp. 1177–1184 (2008)
11. Sinha, A., Duchi, J.C.: Learning kernels with random features. In: *Advances in Neural Information Processing Systems*, pp. 1298–1306 (2016)
12. Vincent, P., Bengio, Y.: Kernel matching pursuit. *Mach. Learn.* **48**(1–3), 165–187 (2002)
13. Wu, D., Wang, B., Precup, D., Boulet, B.: Boosting based multiple kernel learning and transfer regression for electricity load forecasting. In: Altun, Y. (ed.) *ECML PKDD 2017. LNCS (LNAI)*, vol. 10536, pp. 39–51. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-71273-4\\_4](https://doi.org/10.1007/978-3-319-71273-4_4)