Scalable and Efficient Learning from Crowds with Gaussian Processes

Pablo Morales-Álvarez[a], Pablo Ruiz[b], Raúl Santos-Rodríguez[c], Rafael Molina[a], Aggelos K. Katsaggelos[b]

[a] *Computer Science and Artificial Intelligence Department, University of Granada, Spain.*
[b] *Electrical Engineering and Computer Science Department, Northwestern University, USA.*
[c] *Intelligent Systems Laboratory, University of Bristol, UK.*

Abstract

Over the last few years, multiply-annotated data has become a very popular source of information. Online platforms such as Amazon Mechanical Turk have revolutionized the labelling process needed for any classification task, sharing the effort between a number of annotators (instead of the classical single expert). This *crowdsourcing* approach has introduced new challenging problems, such as handling disagreements on the annotated samples or combining the unknown expertise of the annotators. Probabilistic methods, such as Gaussian Processes (GP), have proven successful to model this new crowdsourcing scenario. However, GPs do not scale up well with the training set size, which makes them prohibitive for medium-to-large datasets (beyond 10K training instances). This constitutes a serious limitation for current real-world applications. In this work, we introduce two scalable and efficient GP-based crowdsourcing methods that allow for processing previously-prohibitive datasets. The first one is an efficient and fast approximation to GP with squared exponential (SE) kernel. The second allows for learning a more flexible kernel at the expense of a heavier training (but still scalable to large datasets). Since the latter is not a GP-SE approximation, it can be also considered as a whole new scalable and efficient crowdsourcing method, useful for any dataset size. Both methods use Fourier features and variational inference, can predict the class of new samples, and estimate the expertise of the involved annotators. A complete experimentation compares them with state-of-the-art probabilistic approaches in synthetic and real crowdsourcing datasets of different sizes. They stand out as the best performing approach for large scale problems. Moreover, the second method is competitive with the current state-of-the-art for small datasets.

⋆This work was supported by the Spanish Ministry of Economy and Competitiveness under project DPI2016-77869-C2-2-R, the US Department of Energy (DE-NA0002520) and the Visiting Scholar Program at the University of Granada. PMA received financial support through *La Caixa* Fellowship for Doctoral Studies (*La Caixa* Banking Foundation, Barcelona, Spain).

Email addresses: pablomorales@decsai.ugr.es (Pablo Morales-Álvarez), mataran@northwestern.edu (Pablo Ruiz), enrsr@bristol.ac.uk (Raúl Santos-Rodríguez), rms@decsai.ugr.es (Rafael Molina), aggk@eecs.northwestern.edu (Aggelos K. Katsaggelos)

Keywords: Scalable Crowdsourcing, Classification, Gaussian Processes, Fourier Features, Bayesian Modeling, Variational Inference.

1. Introduction

The term *crowdsourcing* was coined in 2006 by J. Howe [1] to refer to "the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined generally large group of people in the form of an open call". In the last decade, many crowdsourcing services have proliferated in the Internet, where a dataset can be published and millions of people around the world can provide labels in exchange for a reward [2]. Amazon Mechanical Turk (`www.amt.com`), Galaxy Zoo (`www.galaxyzoo.org`), Zooniverse (`www.zooniverse.org`), Crowdflowers (`www.crowdflower.com`) or Clickworker (`www.clickworker.com`) are among the most popular ones. Due to the great number of potential annotators, large data sets can be labeled in a very short time, overcoming one of the main limitations of the classical expert-alone labelling process. However, this crowdsourcing approach has introduced new challenging problems, such as combining the unknown expertise of annotators, dealing with disagreements on the annotated samples, or detecting the existence of spammer and adversarial annotators [2]. All these problems have required probabilistic sound solutions, beyond the naive use of majority voting plus classical classification methods.

Crowdsourcing applications are growing rapidly. Since the early innovative use to detect small volcanoes in Magellan SAR images of Venus [3], crowdsourcing techniques have been applied to a wide range of modern problems such as mitosis detection in breast cancer histology images [4], topic modelling from crowds [5], and detection of glitches in signals acquired by the laureate Laser Interferometer Gravitational-Wave Observatory (LIGO) [6]. There also exist some recent attempts to combine crowdsourcing with Deep Learning approaches [4, 7]. Interestingly, the growth of social websites based on user-generated content (TripAdvisor, Twitter, YouTube) has turned multiple-annotation into a very natural way of labeling reviews, opinions, or videos. This relates crowdsourcing to the emerging *explainable-AI* [8] which, in addition to predict a label for a given sample, explains the decision process in a human understandable and reconstructable way.

The first paper on crowdsourcing dates back to 1979 [9]. Early contributions addressed the estimation of the underlying true labels and the reliability of the annotators, but were not conceived to learn a classifier. This idea was explored by Raykar *et al.* [10], who proposed to jointly estimate the coefficients of a logistic regression (LR) classifier and the annotators' expertise. The latter is modelled through the *sensitivity* and *specificity* concepts, which refer to the accuracy of the annotator when labelling instances from each class. Yan *et al.* [11] (see also the subsequent journal version [12]), introduced a crowdsourcing classifier (also based on LR) which considers a feature-dependent model for the

annotators' expertise. The main limitation of these two approaches is the simple LR classification model, which can only deal with linearly separable data. Rodrigues *et al.* [13] overcame this problem by introducing a crowdsourcing classifier based on Gaussian Processes (GP) [14, 15, 16]. GP is a probabilistic state-of-the-art model for functions, which uses the so-called *"kernel trick"* [17, Chapter 6] to deal with complex non-linear decision boundaries. Moreover, its Bayesian formulation excels at uncertainty quantification [14]. Expectation Propagation (EP) [18] (see also [14, Section 3.6]) was used as inference procedure for GP in [13]. Recently, Variational Inference (VI) [19, 20] was used as an alternative to EP in crowdsourcing, outperforming it in both predictive performance and computational cost [21, 22]. These probabilistic GP-based methods have proven very successful in the crowdsourcing literature. However, the poor scalability of standard GP models hampers their applicability to current medium-to-large scale real-world problems. Therefore, the development of scalable and efficient methods is one of the main research lines in crowdsourcing.

More specifically, classical[1] GPs operate with $N \times N$ kernel matrices, where N is the training set size. This implies a $\mathcal{O}(N^2)$ cost in RAM memory, a $\mathcal{O}(N^3)$ computational complexity at the training step (since the kernel matrix must be inverted), and $\mathcal{O}(N^2)$ cost in the test step. As a consequence, $N = 10^4$ instances is generally considered the practical limit of standard GPs [14]. Since current real-world problems usually involve larger datasets, many *sparse GP approximations* have been developed in the Machine Learning community during the last years. The first approaches focused on selecting a convenient subset of the training set and applying standard GP there [23], see also [14, Chapter 8]. Later on, *pseudo-inputs* and *inducing points* were proposed as a smarter way to reduce the computational cost of classical GP without completely loosing the information provided by the discarded points [24]. This approach has become very popular, and many works have been devoted to analyze it in depth and advance it further [25, 26, 27, 28]. Another recent promising approach is based on the random Fourier features approximation to the kernel matrix [29], which was proposed for GP-regression in [30] and further improved in [31]. Moreover, it was recently extended to GP-classification in [16].

In this work, we start by applying the aforementioned Fourier features methodology to approximate the squared exponential (SE) kernel of the GP-based crowdsourcing method proposed in [21, 22]. This approach is referred to as *RFF* (Random Fourier Features). Then, we also propose *VFF* (Variational Fourier Features), which does not approximate a SE kernel but learns a new one well-suited for the data at hand. The training cost and RAM memory requirements for both, including the computation of the Fourier features, scale linearly with N, and their test cost is independent on N. These are very significant reductions with respect to previous approaches. Whereas *RFF* is a large-scale approximation of the previous approach in [21, 22], *VFF* is a whole new scal-

[1]Throughout this work, we will refer to *classical* and *standard* GP interchangeably to denote the typical and well-known formulation in [14].

able crowdsourcing method, whose additional flexibility allows one to capture new relevant patterns (even in previously-reachable small datasets). However, *VFF* is more prone to overfitting, and slower in practice. A complete experimentation with real and synthetic crowdsourcing datasets of different sizes will show that i) the proposed methods can handle much larger training sets than previous approaches, ii) they have better generalization capability with a faster training step, iii) the test computational cost is extraordinarily reduced, iv) the estimations of annotators' sensitivity and specificity are very accurate, and v) *VFF* is competitive with other state-of-the-art methods in small datasets.

The rest of the paper is organized as follows. Section 2 introduces the probabilistic modelling of the proposed methods. Section 3 presents the variational inference scheme used to estimate the posterior distributions and all the parameters of the model. Section 4 shows the predictive distribution to be used in the test step. Section 5 includes a complete experimentation evaluating the proposed methods. Finally, the main conclusions and some future outlook are provided in Section 6.

2. Probabilistic modelling

Formally, a crowdsourcing classification problem involves a training dataset $\{\mathbf{X}, \mathbf{Y}\}$, where $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^\intercal \in \mathbb{R}^{N \times D}$ is the set of features, and $\mathbf{Y} = \{y_n^r \in \{0,1\} | n = 1, \dots, N,\ r \in R_n\}$ is the set annotations. N, D, and R denote, respectively, the number of training instances, their dimension (i.e. the number of features), and the number of annotators. $R_n \subseteq \{1, \dots, R\}$ denotes the set of annotators that labelled the n-th instance. Analogously, we define $N_r \subseteq \{1, \dots, N\}$ as the set of instances annotated by the r-th annotator.

The most successful probabilistic crowdsourcing approaches model the set of annotations $\mathbf{Y}$ by introducing a set of underlying unknown *real* labels $\mathbf{z} = (z_1, \dots, z_N)^\intercal \in \{0,1\}^N$. Given z_n and $r \in R_n$, the r-th annotator's label is modelled with the conditional Bernoulli distributions

$$\mathrm{p}(y_n^r = 1 | z_n = 1) = \alpha_r, \quad \mathrm{p}(y_n^r = 0 | z_n = 0) = \beta_r, \tag{1}$$

where $\alpha_r, \beta_r \in [0,1]$ are called *sensitivity* and *specificity* for the r-th annotator, respectively. These numbers represent the reliability of that annotator when labelling instances in each class. Assuming independence between annotators and across their annotations, we have

$$\mathrm{p}(\mathbf{Y} | \mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \prod_{r=1}^{R} \prod_{n \in N_r} \left[\alpha_r^{y_n^r} (1 - \alpha_r)^{1 - y_n^r} \right]^{z_n} \left[(1 - \beta_r)^{y_n^r} \beta_r^{1 - y_n^r} \right]^{1 - z_n}, \tag{2}$$

where we denote $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_R)^\intercal$, and $\boldsymbol{\beta} = (\beta_1, \dots, \beta_R)^\intercal$.

In this work, as in [22], all the α_r and β_r are treated in a Bayesian way, i.e. they are assumed to be stochastic variables. More specifically, they are assigned prior beta distributions $\alpha_r \to \mathrm{Beta}(a_\alpha^r, b_\alpha^r)$ and $\beta_r \to \mathrm{Beta}(a_\beta^r, b_\beta^r)$. Recall that

$\text{Beta}(x|a,b) \propto x^{a-1}(1-x)^{b-1}$ for $0 < x < 1$, with $\mathbb{E}(x) = a/(a+b)$. During inference, the following expectations of a beta distribution will be also required

$$\mathbb{E}(\log x) = \psi(a) - \psi(a+b), \quad \mathbb{E}(\log(1-x)) = \psi(b) - \psi(a+b), \tag{3}$$

where ψ denotes the digamma function (see [17, Exercise 2.11]). In a beta distribution, the hyper-parameters a and b can be set to introduce prior knowledge about the variable (in our case, the reliability of each annotator labelling instances in each class). When no prior knowledge is available, $a = b = 1$ produces an uniform prior distribution. Since the *specificity* and *sensitivity* of the different annotators are assumed independent, we have the joint priors:

$$\text{p}(\boldsymbol{\alpha}) = \prod_{r=1}^{R} \text{Beta}(\alpha_r|a_\alpha^r, b_\alpha^r), \quad \text{p}(\boldsymbol{\beta}) = \prod_{r=1}^{R} \text{Beta}(\beta_r|a_\beta^r, b_\beta^r). \tag{4}$$

Finally, to model the underlying real labels $\mathbf{z}$ given the features $\mathbf{X}$, Gaussian Processes (GP) has proven to be the most successful probabilistic approach, mainly because of its great flexibility and excellent uncertainty quantification [13, 21, 22]. A GP introduces N latent variables $(f_1 = f(\mathbf{x}_1), \ldots, f_N = f(\mathbf{x}_N)) =: \mathbf{f}$ that jointly follow a multivariate normal distribution whose covariance matrix (the *kernel matrix*) depends on $\mathbf{X}$, i.e. the distribution of $\mathbf{f}$ is $\mathcal{N}(\mathbf{0}, \mathbf{K} = (k(\mathbf{x}_n, \mathbf{x}_m))_{1\leq n,m\leq N})$. The kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ encodes the properties (like smoothness) of the functions $f(\mathbf{x})$ considered. Then, given each latent variable f_n, the underlying real label z_n is modelled with the sigmoid function σ, $\text{p}(z_n = 1|f_n) = \sigma(f_n) = (1 + \exp(-f_n))^{-1}$. Under this common classical model, the main difference between the previous approaches [13] and [21, 22] is the inference procedure used: Expectation Propagation [32] in the former and Variational Inference [19, 20] in the latter (recall the second paragraph of Section 1). Figure 1a) shows a graphical representation of this GP-based classical model, which is in the basis of our proposal.

Although standard GP is well-known for modelling very complex data and accurately quantifying and propagating uncertainty, it does not scale up well to large datasets (recall the fourth paragraph in Section 1). Therefore, different sparse GP approximations have been proposed over the last years in the Machine Learning community [27, 28, 30, 31, 16]. Here, as it is done for regression in [30, 31] and for classification in [16], we will resort to the interesting Fourier features approximation [29] and will apply it to crowdsourcing.

2.1. Fourier features

The work [29] presents a general methodology to approximate any positive-definite shift-invariant kernel k by a linear one. This is achieved by projecting the original D-dimensional data $\mathbf{x}$ into $2D_f$ Fourier features $\boldsymbol{\phi}(\mathbf{x})$, whose linear kernel k_L approximates the original k. In the case of GP, this linearity enables one to *undo* the so-called kernel trick and work in the primal space of features [17, Chapter 6]. With this, $N \times N$ matrix inversions are substituted by

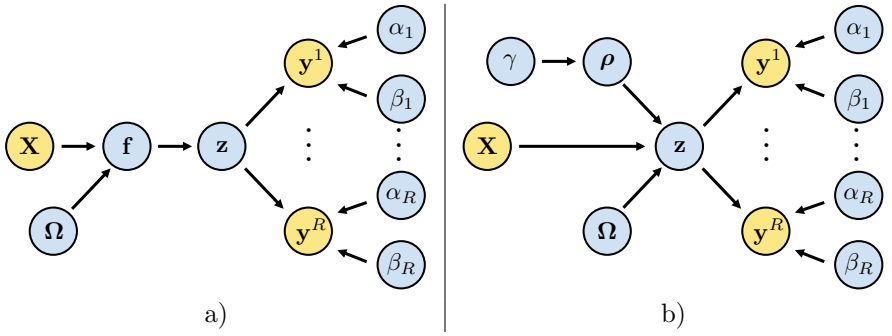

Figure 1: Graphical representations of the classical GP-based probabilistic model for crowdsourcing (left) and the new one proposed here (right). Yellow nodes represent the observed variables, and blue nodes represent the variables to be estimated. Notice that the only difference is in the connection between the features $\mathbf{X}$ and the underlying real labels $\mathbf{z}$ (a GP is used on the left and a Bayesian logistic-regression model based on Fourier features on the right). In the latter, we have $\mathbf{\Omega} = \omega$ for *RFF* and $\mathbf{\Omega} = \mathbf{W}$ for *VFF*.

$2D_f \times 2D_f$ ones, yielding a total $\mathcal{O}(ND_f^2 + D_f^3)$ training cost. In large-scale applications we can set $D_f \ll N$, and the resulting $\mathcal{O}(ND_f^2)$ complexity, which is *linear* in N, constitutes an important reduction over the original $\mathcal{O}(N^3)$. Moreover, both the test complexity and the memory cost reduce to $\mathcal{O}(D_f^2)$, which is *independent* on N. Of course, the main drawback of this process is that we work with an *approximation* to the original kernel.

More specifically, let us consider the well-known SE kernel $k(\mathbf{x}, \mathbf{y}) = \gamma \cdot \exp(-||\mathbf{x}-\mathbf{y}||^2/(2\omega^2))$, where the hyper-parameters γ and ω are called *variance* and *length-scale*, respectively. Following [29], this kernel can be approximated as $k(\mathbf{x}, \mathbf{y}) \approx k_L(\mathbf{x}, \mathbf{y}) := \gamma \cdot \boldsymbol{\phi}(\mathbf{x})^\intercal \boldsymbol{\phi}(\mathbf{y})$, where the *Fourier features* $\boldsymbol{\phi}$ are given by

$$\boldsymbol{\phi}(\mathbf{x})^\intercal = D_f^{-1/2} \cdot \Big(\cos(\mathbf{w}_1^\intercal \mathbf{x}), \sin(\mathbf{w}_1^\intercal \mathbf{x}), \ldots, \cos(\mathbf{w}_{D_f}^\intercal \mathbf{x}), \sin(\mathbf{w}_{D_f}^\intercal \mathbf{x})\Big) \in \mathbb{R}^{2D_f}, \tag{5}$$

and the D_f *Fourier frequencies* $\mathbf{w}_i$ must be sampled from a normal distribution $\mathcal{N}(\mathbf{0}, \omega^{-2}\mathbf{I})$. This approximation exponentially improves with the number D_f of Fourier frequencies used [29, Claim 1]. However, increasing D_f will go at the expense of increasing train and test computational cost and memory requirements in our methods. Other kernels could also be used, but that would involve sampling from a different distribution.

2.2. The proposed models

Our first proposal consists of introducing this Fourier features approximation for the SE kernel in the variational GP-based crowdsourcing method *VGPCR* [21, 22]. Notice that, as explained above, the Fourier frequencies $\mathbf{w}_i$ must be sampled from $\mathcal{N}(\mathbf{0}, \omega^{-2}\mathbf{I})$ and fixed, whereas the length-scale hyper-parameter ω must be estimated during training (just as for standard GPs). To uncouple $\mathbf{w}_i$

and ω, we resort to the following equivalent expression for the Fourier features, which makes explicit the dependence on ω

$$\boldsymbol{\phi}(\mathbf{x}|\omega)^\intercal = D_f^{-1/2}\times \tag{6}$$
$$\times \left(\cos(\omega^{-1}\mathbf{w}_1^\intercal\mathbf{x}), \sin(\omega^{-1}\mathbf{w}_1^\intercal\mathbf{x}), \ldots, \cos(\omega^{-1}\mathbf{w}_{D_f}^\intercal\mathbf{x}), \sin(\omega^{-1}\mathbf{w}_{D_f}^\intercal\mathbf{x})\right),$$

where now $\mathbf{w}_i$ must be sampled now from $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Then, undoing the kernel trick and passing to the primal space of features, we change the GP for the equivalent[2] Bayesian logistic-regression model $\mathrm{p}(z_n = 1|\mathbf{x}_n, \omega, \boldsymbol{\rho}) = (1 + \exp(-\boldsymbol{\phi}(\mathbf{x}_n|\omega)^\intercal\boldsymbol{\rho}))^{-1}$, where the logistic-regression weights $\boldsymbol{\rho}$ follow a normal prior $\mathcal{N}(\mathbf{0}, \gamma\mathbf{I})$ (more details about the kernel trick in [17, Chapter 6]). Finally, assuming independence between the different instances given $\boldsymbol{\rho}$, we have

$$\mathrm{p}(\mathbf{z}|\boldsymbol{\rho}, \omega, \mathbf{X}) = \prod_{n=1}^{N}\left(\frac{1}{1+e^{-\boldsymbol{\rho}^\intercal\boldsymbol{\phi}(\mathbf{x}_n|\omega)}}\right)^{z_n}\left(\frac{e^{-\boldsymbol{\rho}^\intercal\boldsymbol{\phi}(\mathbf{x}_n|\omega)}}{1+e^{-\boldsymbol{\rho}^\intercal\boldsymbol{\phi}(\mathbf{x}_n|\omega)}}\right)^{1-z_n}. \tag{7}$$

This model will be refered to as *RFFGPCR* (Random Fourier Features Gaussian Processes for Crowdsourcing), or *RFF* for short. In *RFF*, the Fourier frequencies $\mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_{D_f})^\intercal \in \mathbb{R}^{D_f\times D}$ are *randomly* sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ and fixed from the beginning, whereas ω is estimated during training (to maximize the marginal likelihood, see Section 3).

Our second proposal follows the same rationale as *RFF*, but *optimizes* the Fourier frequencies $\mathbf{w}_i$ in eq. (5). Since they are estimated to maximize the marginal likelihood within a variational scheme (see Section 3), this approach is refered to as *VFFGPCR* (Variational Fourier Features Gaussian Processes for Crowdsourcing), or *VFF* for short. Therefore, the *VFF* model for $\mathbf{z}$ is identical to that of *RFF*, eq. (7), but with $\mathbf{W}$ playing the role of ω and with the original Fourier features expression in eq. (5) instead of the modified eq. (6). To unify the notation, we will indistinctly write $\boldsymbol{\Omega}$ for ω (*RFF*) or $\mathbf{W}$ (*VFF*), and therefore

$$\mathrm{p}(\mathbf{z}|\boldsymbol{\rho}, \boldsymbol{\Omega}, \mathbf{X}) = \prod_{n=1}^{N}\left(\frac{1}{1+e^{-\boldsymbol{\rho}^\intercal\boldsymbol{\phi}(\mathbf{x}_n|\boldsymbol{\Omega})}}\right)^{z_n}\left(\frac{e^{-\boldsymbol{\rho}^\intercal\boldsymbol{\phi}(\mathbf{x}_n|\boldsymbol{\Omega})}}{1+e^{-\boldsymbol{\rho}^\intercal\boldsymbol{\phi}(\mathbf{x}_n|\boldsymbol{\Omega})}}\right)^{1-z_n}, \tag{8}$$

with $\boldsymbol{\phi}(\mathbf{x}_n|\boldsymbol{\Omega})$ as in eq. (6) (*RFF*) or eq. (5) (*VFF*).

Unlike *RFF*, notice that *VFF* is no longer an approximation to *VGPCR* (for which the Fourier frequencies must be sampled from $\mathcal{N}(\mathbf{0}, \omega^{-2}\mathbf{I})$), but a whole new probabilistic crowdsourcing method that *learns* an appropriate kernel. Moreover, its computational cost is similar to *RFF*'s. More specifically, we will see that the theoretical training complexity for *VFF* is $\mathcal{O}(ND_fD + ND_f^2)$ (whereas it is $\mathcal{O}(ND_f^2)$ for *RFF*). This is linear in N (like for *RFF*), and therefore much more scalable than the original *VGPCR* ($\mathcal{O}(N^3)$). Nonetheless, the

[2] Again, we stress that this new model is equivalent to GP with the Fourier features *approximation* for the SE kernel.

experimentation will show that the Fourier frequencies optimization significantly slows down *VFF* when compared to *RFF* in practice. Moreover, whereas D_f has a clear influence in *RFF* performance (the higher, the better it is the kernel approximation), it is related to the complexity of the model (the degrees of freedom) in *VFF*. Therefore, in *VFF*, large values of D_f may lead to overfitting to the training set.

In summary, the proposed probabilistic crowdsourcing model is

$$\mathrm{p}(\mathbf{Y}, \mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta} | \boldsymbol{\Omega}, \gamma) = \mathrm{p}(\mathbf{Y}|\mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})\mathrm{p}(\mathbf{z}|\boldsymbol{\rho}, \boldsymbol{\Omega})\mathrm{p}(\boldsymbol{\rho}|\gamma)\mathrm{p}(\boldsymbol{\alpha})\mathrm{p}(\boldsymbol{\beta}), \tag{9}$$

with $\mathrm{p}(\mathbf{Y}|\mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ as in eq. (2), $\mathrm{p}(\mathbf{z}|\boldsymbol{\rho}, \boldsymbol{\Omega})$ as in eq. (8), $\mathrm{p}(\boldsymbol{\rho}|\gamma) = \mathcal{N}(\boldsymbol{\rho}|\mathbf{0}, \gamma\mathbf{I})$, and $\mathrm{p}(\boldsymbol{\alpha}), \mathrm{p}(\boldsymbol{\beta})$ as in eq. (4). Notice that, for clarity, we have omitted $\mathbf{X}$ from the notation. Figure 1b) shows a graphical representation of the proposed model.

3. Variational Bayes inference

Once the training set $\{\mathbf{X}, \mathbf{Y}\}$ is observed, Bayesian inference seeks to calculate the maximum-likelihood hyperparameters $(\hat{\boldsymbol{\Omega}}, \hat{\gamma}) = \arg\max_{\boldsymbol{\Omega}, \gamma} \mathrm{p}(\mathbf{Y}|\boldsymbol{\Omega}, \gamma)$, and the posterior distribution $\mathrm{p}(\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta}|\mathbf{Y}, \hat{\boldsymbol{\Omega}}, \hat{\gamma})$. However, in our case, the marginal likelihood $\mathrm{p}(\mathbf{Y}|\boldsymbol{\Omega}, \gamma) = \int_{\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta}} \mathrm{p}(\mathbf{Y}, \mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta}|\boldsymbol{\Omega}, \gamma)$ cannot be obtained in closed form (for simplicity, the sum in the discrete variable $\mathbf{z}$ is denoted with integration). Variational inference [19, 20], see also [17, Section 10.1], is a very popular approach to obtain an approximation to the posterior distribution in Bayesian inference. It consists of finding, inside a predefined family $\mathcal{Q}$, the distribution $\mathrm{q} \in \mathcal{Q}$ that minimizes the Kullback-Leibler divergence (KL) from q to the real posterior. Recall that the KL divergence from a distribution $\mathrm{q}(\mathbf{x})$ to another $\mathrm{p}(\mathbf{x})$ is defined as $\mathrm{KL}(\mathrm{q}||\mathrm{p}) = \int \mathrm{q}(\mathbf{x}) \log(\mathrm{q}(\mathbf{x})/\mathrm{p}(\mathbf{x}))\mathrm{d}\mathbf{x}$, which is always greater or equal to zero, and vanishes if and only if $\mathrm{q} = \mathrm{p}$. A different popular approach to approximate the posterior distribution is called *Expectation Propagation* [18]. However, to the best of our knowledge, variational inference has achieved better results in classical GP-based probabilistic crowdsourcing methods, being also significantly more efficient (which is specially relevant in large-scale scenarios like ours) [21, 22].

Here, for $\boldsymbol{\Omega}$ and γ fixed, we propose an approximate posterior of the form

$$\mathrm{q}(\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathrm{q}(\boldsymbol{\rho})\mathrm{q}(\boldsymbol{\alpha})\mathrm{q}(\boldsymbol{\beta})\mathrm{q}(\mathbf{z})^{3}. \tag{10}$$

The reason for "uncoupling" $\mathbf{z}$ is that integrating it out in the true posterior $\mathrm{p}(\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta}|\mathbf{Y}, \boldsymbol{\Omega}, \gamma)$ is analytically intractable. Notice that this also applies to $\boldsymbol{\rho}$ due to the sigmoids in eq. (8), for which we will additionally resort to the local variational bound of the sigmoid [17, Section 10.5]. Using the factorization

[3] This is equivalent to the more general form $\mathrm{q}(\boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta})\mathrm{q}(z)$, since the variables $\boldsymbol{\rho}$, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$ are coupled in the joint model of eq. (9) only through $\mathbf{z}$.

proposed in eq. (10), the well-known mean-field formula [17, Section 10.1.1, eq. (10.9)] yields the following update for $\mathrm{q}(\mathbf{z})$ (which factorizes along data points):

$$\begin{aligned}
\mathrm{q}(z_n = 0) &\propto \prod_{r \in R_n} \exp\left\{ y_n^r \mathbb{E}_\mathrm{q}(\log(1-\beta_r)) + (1-y_n^r)\mathbb{E}_\mathrm{q}(\log \beta_r) \right\}, \\
\mathrm{q}(z_n = 1) &\propto \exp(\boldsymbol{\phi}(\mathbf{x}_n|\boldsymbol{\Omega})^\intercal \mathbb{E}_\mathrm{q}(\boldsymbol{\rho})) \\
&\quad \times \prod_{r \in R_n} \exp\left\{ y_n^r \mathbb{E}_\mathrm{q}(\log \alpha_r) + (1-y_n^r)\mathbb{E}_\mathrm{q}(\log(1-\alpha_r)) \right\},
\end{aligned} \tag{11}$$

where the expectations are with respect to the current values of $\mathrm{q}(\boldsymbol{\alpha})$, $\mathrm{q}(\boldsymbol{\beta})$ and $\mathrm{q}(\boldsymbol{\rho})$. For the terms of the form $\mathbb{E}_\mathrm{q}(\log(\cdot))$, recall eq. (3). Analogously, the updates for $\mathrm{q}(\boldsymbol{\alpha})$ and $\mathrm{q}(\boldsymbol{\beta})$ factorize along annotators and are given by:

$$\mathrm{q}(\alpha_r) = \mathrm{Beta}\left(\alpha_r \middle| a_\alpha^r + \sum_{n \in N_r} \mathbb{E}_\mathrm{q}(z_n) y_n^r, b_\alpha^r + \sum_{n \in N_r} \mathbb{E}_\mathrm{q}(z_n)(1-y_n^r) \right), \tag{12}$$

$$\mathrm{q}(\beta_r) = \mathrm{Beta}\left(\beta_r \middle| a_\beta^r + \sum_{n \in N_r} (1-\mathbb{E}_\mathrm{q}(z_n))(1-y_n^r), b_\beta^r + \sum_{n \in N_r} (1-\mathbb{E}_\mathrm{q}(z_n)) y_n^r \right), \tag{13}$$

where the expectations are with respect to the current distribution $\mathrm{q}(\mathbf{z})$.

In order to update $\mathrm{q}(\boldsymbol{\rho})$, we find analytic intractability in $\boldsymbol{\rho}$ due to the sigmoids in $\mathrm{p}(\mathbf{z}|\boldsymbol{\rho}, \boldsymbol{\Omega})$, recall eq. (8). To overcome this, we use the local variational bound of the sigmoid [17, Section 10.5, eq. (10.144)], which yields

$$\mathrm{p}(\mathbf{z}|\boldsymbol{\rho}, \boldsymbol{\Omega}) \geq \exp\left(\mathbf{v}^\intercal \boldsymbol{\Phi} \boldsymbol{\rho} - \boldsymbol{\rho}^\intercal \boldsymbol{\Phi}^\intercal \boldsymbol{\Lambda} \boldsymbol{\Phi} \boldsymbol{\rho} + C(\boldsymbol{\xi}) \right) =: H(\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\Omega}, \boldsymbol{\xi}). \tag{14}$$

Notice that this lower bound is exponentially-quadratic in $\boldsymbol{\rho}$, which will allow us to identify a Gaussian distribution in $\boldsymbol{\rho}$. In exchange, we are introducing N additional hyper-parameters $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_N)$ to be estimated. Here we are writing $\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_N)^\intercal \in \mathbb{R}^{N \times (2D_f)}$ for the matrix of Fourier features, $\mathbf{v} = \mathbf{z} - (1/2)\mathbf{1}$, $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda(\xi_1), \ldots, \lambda(\xi_N))$, and $\lambda(\xi) = (2\xi)^{-1}\left((1+\exp(-\xi))^{-1} - 1/2\right)$. The term $C(\boldsymbol{\xi}) = \sum_{n=1}^N \left(\lambda(\xi_n)\xi_n^2 + \xi_n/2 - \log\left(1 + e^{\xi_n}\right)\right)$ only depends on $\boldsymbol{\xi}$.

Using eq. (14) we have, up to a constant, the following upper bound for the KL divergence (which must be minimized, instead of the intractable KL itself, in $\mathrm{q}(\boldsymbol{\rho})$, with $\mathrm{q}(\mathbf{z})$, $\mathrm{q}(\boldsymbol{\alpha})$ and $\mathrm{q}(\boldsymbol{\beta})$ fixed):

$$\begin{aligned}
&\mathrm{KL}(\mathrm{q}(\boldsymbol{\rho})\mathrm{q}(\boldsymbol{\alpha})\mathrm{q}(\boldsymbol{\beta})\mathrm{q}(\mathbf{z}) || \mathrm{p}(\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta}|\mathbf{Y}, \boldsymbol{\Omega}, \gamma)) \leq \\
&\int_{\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\Omega}, \gamma} \mathrm{q}(\boldsymbol{\rho})\mathrm{q}(\boldsymbol{\alpha})\mathrm{q}(\boldsymbol{\beta})\mathrm{q}(\mathbf{z}) \log \frac{\mathrm{q}(\boldsymbol{\rho})\mathrm{q}(\boldsymbol{\alpha})\mathrm{q}(\boldsymbol{\beta})\mathrm{q}(\mathbf{z})}{\mathrm{p}(\mathbf{Y}|\mathbf{z}, \boldsymbol{\alpha}, \boldsymbol{\beta}) H(\mathbf{z}, \boldsymbol{\rho}, \boldsymbol{\Omega}, \boldsymbol{\xi}) \mathrm{p}(\boldsymbol{\rho}|\gamma) \mathrm{p}(\boldsymbol{\alpha}) \mathrm{p}(\boldsymbol{\beta})}.
\end{aligned} \tag{15}$$

Following the standard mean-field procedure [17, Section 10.1.1], this minimization yields $\mathrm{q}(\boldsymbol{\rho}) \propto H(\mathbb{E}_\mathrm{q}(\mathbf{z}), \boldsymbol{\rho}, \boldsymbol{\Omega}, \boldsymbol{\xi}) \mathrm{p}(\boldsymbol{\rho}|\gamma)$. Since H is exponentially-quadratic in $\boldsymbol{\rho}$, we have $\mathrm{q}(\boldsymbol{\rho}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, with

$$\boldsymbol{\Sigma} = \left(\gamma^{-1}\mathbf{I} + \boldsymbol{\Phi}^\intercal (2\boldsymbol{\Lambda}) \boldsymbol{\Phi} \right)^{-1}, \quad \boldsymbol{\mu} = \boldsymbol{\Sigma} \boldsymbol{\Phi}^\intercal \mathbb{E}_\mathrm{q}(\mathbf{v}). \tag{16}$$

Then, approximating $\mathrm{p}(\mathbf{z}|\boldsymbol{\rho},\boldsymbol{\Omega})$ by its lower bound $H(\mathbf{z},\boldsymbol{\rho},\boldsymbol{\Omega},\boldsymbol{\xi})$ in the full model $\mathrm{p}(\mathbf{Y},\mathbf{z},\boldsymbol{\rho},\boldsymbol{\alpha},\boldsymbol{\beta}|\boldsymbol{\Omega},\gamma)$, we find that $\boldsymbol{\rho}$ can be marginalized out (again, H being exponentially-quadratic in $\boldsymbol{\rho}$ is essential here). Using the current distribution $\mathrm{q}(\mathbf{z})$, the maximum-likelihood estimators for $\boldsymbol{\Omega}$ and γ are

$$(\hat{\boldsymbol{\Omega}},\hat{\gamma}) = \tag{17}$$
$$\arg\max_{\boldsymbol{\Omega},\gamma} \left(-\log|2\gamma\boldsymbol{\Phi}^\intercal\boldsymbol{\Lambda}\boldsymbol{\Phi}+\mathbf{I}| + \mathbb{E}_{\mathrm{q}}(\mathbf{v})^\intercal\boldsymbol{\Phi}\left(\gamma^{-1}\mathbf{I}+2\boldsymbol{\Phi}^\intercal\boldsymbol{\Lambda}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^\intercal\mathbb{E}_{\mathrm{q}}(\mathbf{v})\right).$$

Finally, the hyper-parameters $\boldsymbol{\xi}$ are estimated to minimize the right-hand side of eq. (15), which yields (notice that the square is element-wise)

$$\boldsymbol{\xi} = \sqrt{\mathrm{diag}\left(\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\intercal\right)+\left(\boldsymbol{\Phi}\boldsymbol{\mu}\right)^2}. \tag{18}$$

In summary, the proposed methods calculate sequences $\{\boldsymbol{\xi}^k\}$, $\{\boldsymbol{\Omega}^k,\gamma^k\}$, $\{\mathrm{q}^k(\boldsymbol{\rho})\}$, $\{\mathrm{q}^k(\boldsymbol{\alpha})\}$, $\{\mathrm{q}^k(\boldsymbol{\beta})\}$, $\{\mathrm{q}^k(\mathbf{z})\}$ until convergence, following the formulas derived in this section. The training process is summarized in Algorithm 1. The computational cost of the algorithms is dominated by $2D_f \times 2D_f$ matrix inversions (e.g. eq. (17)) and $(2D_f \times N)\cdot(N\times 2D_f)$ matrix multiplications (e.g. $\boldsymbol{\Sigma}$ in eq. (16)). This yields a theoretical complexity of $\mathcal{O}(D_f^3+ND_f^2)$ which, in large scale scenarios (where D_f will be taken $D_f \ll N$), is $\mathcal{O}(ND_f^2)$. In the case of VFF, the optimization with respect to the $D_f \cdot D$ components of $\mathbf{W}$ introduces an additional dependence on D, and yields $\mathcal{O}(ND_f^2+ND_fD)$ cost.

Algorithm 1 Training of RFF and VFF

Require: $\mathbf{X}$, $\mathbf{Y}$, $\boldsymbol{\Omega}^0$, $\mathrm{q}^0(\boldsymbol{\rho})$, $\mathrm{q}^0(\mathbf{z})$, $k=0$.
  repeat
    Update $\boldsymbol{\xi}^{k+1}$ with eq. (18) using $\mathrm{q}^k(\boldsymbol{\rho})$ and $\boldsymbol{\Omega}^k$;
    Update γ^{k+1} and $\boldsymbol{\Omega}^{k+1}$ with eq. (17) using $\boldsymbol{\xi}^{k+1}$ and $\mathrm{q}^k(\mathbf{z})$;
    Update $\mathrm{q}^{k+1}(\boldsymbol{\rho})$ with eq. (16) using $\boldsymbol{\xi}^{k+1}$, $\boldsymbol{\Omega}^{k+1}$, γ^{k+1} and $\mathrm{q}^k(\mathbf{z})$;
    Update $\mathrm{q}^{k+1}(\boldsymbol{\alpha})$ and $\mathrm{q}^{k+1}(\boldsymbol{\beta})$ with eqs. (12)-(13) using $\mathrm{q}^k(\mathbf{z})$;
    Update $\mathrm{q}^{k+1}(\mathbf{z})$ with eq. (11) using $\boldsymbol{\Omega}^{k+1}$, $\mathrm{q}^{k+1}(\boldsymbol{\rho})$, $\mathrm{q}^{k+1}(\boldsymbol{\alpha})$ and $\mathrm{q}^{k+1}(\boldsymbol{\beta})$.
    $k=k+1$;
  until convergence
  Output: Final values $\hat{\boldsymbol{\xi}}$, $\hat{\boldsymbol{\Omega}}$, $\hat{\gamma}$, $\hat{\mathrm{q}}(\boldsymbol{\rho})$, $\hat{\mathrm{q}}(\boldsymbol{\alpha})$, $\hat{\mathrm{q}}(\boldsymbol{\beta})$, $\hat{\mathrm{q}}(\mathbf{z})$.

It is interesting to examine and understand how the proposed methodology mitigates the effect of weak annotators (i.e. those who may provide unreliable labels). Recall from eq. (1) that each annotator reliability is modelled through sensitivity and specificity parameters α and β. These parameters are estimated during the training step, see eqs. (12) and (13). Then, these estimations of α and β are used in eq. (11) in order to update the distribution of the underlying real label z for each training instance. Importantly, note that the influence of each annotation y_n^r is appropriately modulated by the estimations of α and β for the corresponding annotator.

This becomes even clearer when degenerate posterior distributions are assumed for α_r and β_r. In this case, the posterior distribution approximation $\mathrm{q}(z_n)$ in eq. (11) is proportional to $\prod_r (1-\beta_r)^{y_n^r}\beta_r^{(1-y_n^r)}$ and $\prod_r \alpha_r^{y_n^r}(1-\alpha_r)^{(1-y_n^r)}$ for $z_n = 0$ and $z_n = 1$, respectively. Suppose that an annotator labels an instance as $y_n^r = 1$. Then, this implies a factor (which can be understood as a "multiplicative" weight) of $(1-\beta_r)$ for the probability of $z_n = 0$, and a factor of α_r for $z_n = 1$. If the annotator is a reliable one, then α_r and β_r are close to 1, which implies a much greater weight for $z_n = 1$ than for $z_n = 0$. However, if the annotator is a weak one (for both classes), then α_r and β_r will be close to 0, and the weight for $z_n = 0$ will be much greater than for $z_n = 1$, making it very likely to correctly switch the (very likely) wrong label provided by this weak annotator. The weaker the annotator is, the more likely it is to switch the annotation. An analogous interpretation applies when the annotator labels $y_n^r = 0$, or when the annotator is weak only for one of the two classes. Observe also that a spammer annotator (i.e. $\alpha_r = \beta_r = 0.5$), will not influence the probability of z_n, as both weights will be identical.

Finally, notice that the Bayesian modelling allows for naturally specifying the available prior knowledge on the annotators. For instance, if a particular annotator is known to be weak (even only for one of the two classes), the corresponding Beta prior distribution (recall the paragraph before eq. (3)) can be conveniently set to integrate in the model this valuable information.

4. The predictive distribution

Once the model is trained, the final distributions $\hat{\mathrm{q}}(\boldsymbol{\alpha})$ and $\hat{\mathrm{q}}(\boldsymbol{\beta})$ represent the estimated sensitivity and specificity for the annotators (as well as their uncertainty). Analogously, $\hat{\mathrm{q}}(\mathbf{z})$ describes the estimated uncertainty for the underlying real labels of the training instances. The most common problem is to, based on the training data, obtain the *predictive distribution* for the real class of a new instance $\mathbf{x}_* \in \mathbb{R}^D$, i.e. compute $\mathrm{p}(z_* = 1|\mathbf{Y})$ (obviously, $\mathrm{p}(z_* = 0|\mathbf{Y}) = 1 - \mathrm{p}(z_* = 1|\mathbf{Y})$). Using the standard approximation for the expectation of the sigmoid under a Gaussian [17, Section 4.5.2, eq. (4.153)], we have

$$\mathrm{p}(z_* = 1|\mathbf{Y}) = \mathbb{E}_{\hat{\mathrm{q}}(\boldsymbol{\rho})}\mathrm{p}(z_* = 1|\boldsymbol{\rho}, \hat{\boldsymbol{\Omega}}) \approx \sigma\left(\frac{\hat{\boldsymbol{\phi}}_*^\intercal \hat{\boldsymbol{\mu}}}{\sqrt{1 + (\pi/8)\hat{\boldsymbol{\phi}}_*^\intercal \hat{\boldsymbol{\Sigma}} \hat{\boldsymbol{\phi}}_*}}\right), \tag{19}$$

where $\sigma(x) = (1 + \exp(-x))^{-1}$ is the sigmoid, $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ are the mean and covariance of the posterior $\hat{\mathrm{q}}(\boldsymbol{\rho})$ (which are obtained in the training step), and $\hat{\boldsymbol{\phi}}_* = \boldsymbol{\phi}(\mathbf{x}_*|\hat{\boldsymbol{\Omega}})$ (using eq. (6) in the case of *RFF* and eq. (5) for *VFF*).

The theoretical computational complexity for the test step is dominated by the computation $\hat{\boldsymbol{\phi}}_*^\intercal \boldsymbol{\Sigma} \hat{\boldsymbol{\phi}}_*$. This implies a $\mathcal{O}(D_f^2)$ cost per test instance. Unlike classical GP, whose corresponding complexity is $\mathcal{O}(N^2)$, this is independent on the number of training instances N. In large scale scenarios (where N is large), this will translate into an overwhelming superiority of the proposed methods in

terms of "production" time (i.e. time needed for prediction), which is essential in real-world applications.

Finally, since eq. (19) is one of the key ingredients for Active Learning (AL) techniques, let us conclude this section by commenting on the use of AL for our RFF and VFF models. In section 1 we motivated the use of crowdsourcing in labeling tasks as a very efficient way to annotate large datasets. In order to further speed up this process, crowdsourcing can be combined with AL. For a classic (non-crowdsourcing) classifier, AL selects the most informative instance from a set of unlabeled samples, and the expert provides the corresponding label. The new labeled sample is included in the training set, and the classifier is retrained (updated). It has been shown that AL significantly reduces the number of samples to be labeled in order to train an accurate classifier (see, for instance, [33]).

In crowdsourcing labeling problems, AL becomes an even more interesting (and challenging) problem, since the best annotator/s to provide the label must also be selected. Interestingly, the majority of probabilistic crowdsourcing AL methods in the literature are based on different combinations of the same two key ingredients: the uncertainty of the model when labeling a new instance (in our case given by the predictive distribution in eq. (19)), and the estimated expertise for each annotator (in our case the sensitivity and specificity posteriors given in eqs. (12) and (13)). Rodrigues *et al.* [13] first select the closest sample to the decision boundary and then the annotator who maximizes the expected probability of success. Yan *et al.* [34] minimize an objective function to simultaneously find the closest sample to the decision boundary and the annotator who minimizes the probability of mistake. More recently, Yang *et al.* [35] select the sample that maximizes the Shannon entropy of the predictive distribution and the annotator who maximizes the probability of success. All these approaches can be naturally used with our predictive distribution in eq. (19) and our estimated sensitivities and specificities in eqs. (12) and (13). However, since the use of AL in crowdsourcing is not the goal of this work, the comparison and development of AL techniques will not be explored here.

5. Experiments

In this section, we evaluate the performance of our methods and compare them with current state-of-the-art probabilistic crowdsourcing approaches. These include the GP-based *VGPCR* [22] and *Rodrigues* [13]. We also include the most straightforward manner to apply a GP to the crowdsourcing setting, *GP-MV*, which consists of a standard GP classifier trained with the majority voting (MV) labels. Finally, to obtain a more thorough comparison, the classical LR-based methods *Raykar* [10] and *Yan* [11] are also considered (recall the second paragraph in Section 1).

Since the main goal is to illustrate the scalability and performance of *RFF* and *VFF* in previously-prohibitive settings, we include two such datasets (where classical approaches must be trained with a subset). The first one, with 28000 training instances, comes from a real health-care activity-recognition problem.

The second one, synthetic and with 100000 training samples, shows the potential of the proposed methods in even larger scale problems. Finally, two real datasets with 700 and 4999 training instances, respectively, are included to illustrate the performance of the proposed methods on small-scale problems. They cover different application domains such as audio recognition and sentiment analysis.

The predictive performance of the methods is compared using the area under the ROC curve (AUC). This metric deals well with imbalance scenarios (it penalizes errors in the minority class), and is independent on the threshold used for the final prediction. In order to compare the computational cost, the CPU time needed for both train and test steps will be reported. Please notice that the train CPU time includes the optimization of all the model parameters, including the Fourier frequencies for *VFF* (recall that *RFF* does not estimate them).

We implemented *RFF*, *VFF*, *VGPCR*, *GP-MV*, *Raykar*, and *Yan* in Matlab©, whereas a Matlab© implementation for *Rodrigues* can be downloaded from his website `http://www.fprodrigues.com`. All the code and datasets will be made available at `http://decsai.ugr.es/vip/software.html` upon acceptance of the paper. The experiments were run on the same machine Intel© Xeon© E5-2630 v4 @ 2.20GHz.

5.1. The sphere dataset

Sphere (Sensor Platform for HEalthcare in Residential Environment) is a recognition dataset where activity predictions are made based on RBG-D video, a tri-axial accelerometer, and environmental sensors [36]. Data was collected from 10 people on two different occasions. There were 8 males and 2 females, with 8 between the ages of 18 to 29 and 2 within the ages of 30 to 39. Each participant was wearing a wrist-worn accelerometer and was asked to perform a series of scripted activities, taking around 25 to 30 minutes in total. These activities are categorized into ambulation actions (e.g. walking), posture actions (e.g. standing), and transitional actions (e.g. sit to stand). The script was carried out twice in full by each participant on different days.

Labeling this data is challenging, since the annotations are inherently noisy. For instance, the precise selection of start and end time is inherently ambiguous, as is the distinction among closely related actions (e.g., "bending" and "kneeling"). In order to mitigate these issues, the full dataset was annotated at least twice by a team of $R = 12$ annotators that were recruited and trained to annotate the set of activities. Our experiments consider the binary task of classifying between ambulatory and sedentary activities based on $D = 12$ statistical features (mean, minimum, maximum, standard deviation, variance) extracted from the acceleration data. This yields a final dataset with 31050 instances.

A set with 3050 instances was left for test[4], yielding a maximum number of 28000 training instances. In order to study the scalability of the compared methods, increasing training set sizes were considered, namely $N \in$

[4]Since true underlying labels were not available in this real problem, test instances were selected among those not having discrepancies between different annotators.

$\{1000, 5000, 10000, 15000, 20000, 28000\}$. As classical GP-based methods are limited in practice to 10000-15000 training points, *VGPCR* and *GP-MV* could not be trained beyond $N = 15000$[5]. A special grid $N \in \{100, 500, 1000, 2500\}$ was used for *Rodrigues*, since it did not manage to converge properly and therefore its computational training cost exploded as N increased (as we will see in Figure 4). Different values of D_f (number of Fourier frequencies) were also considered for *RFF*, $D_f \in \{10, 50, 100, 200, 300, 400, 500, 600, 700\}$, and for *VFF*, $D_f \in \{1, 5, 10, 30, 50, 70, 90, 110, 130, 150\}$. Notice that, since *VFF* optimizes over the Fourier frequencies, it is natural to train it with smaller values of D_f.

The main ideas and interpretations will be provided in this section together with the most relevant figures. For completeness, additional information is included in the tables in Appendix A. Namely, Table A.1 contains the test AUC for all the compared methods (except for *Rodrigues*, see its own Table A.2). Mean and covariance over five independent runs[6] are shown. Analogously, Table A.3 (Table A.4 for *Rodrigues*) shows the CPU time needed for train, and Table A.5 (Table A.6 for *Rodrigues*) the CPU time needed for test.

First, let us examine the trade-off between generalization capability and training CPU time for the compared methods, see Figure 2. Notice that *Rodrigues* does not appear in the figure, since its predictive performance in this problem is around 0.5 in AUC, see Table A.2. Among the rest of methods, (the x-axis of) Figure 2 shows a clear distinction between LR-based ones (*Raykar* and *Yan*, which are below 0.7 in AUC) and GP-based ones (the other four, which reach around 0.79). Of course, this is to be expected due to the more complex non-linear boundaries provided by GP-based methods, and reveals an underlying non-linear structure for the *sphere* dataset (otherwise, the gap between LR- and GP-based methods would be smaller).

Now, among the four outstanding methods in terms of test AUC, the y-axis of Figure 2 shows a clear difference in the CPU time needed to train each one (recall the logarithmic scale in this axis). Namely, the proposed *RFF* and *VFF* are around three and fifty times faster than *GP-MV* / *VGPCR*, respectively. Notice also that, in terms of predictive performance, *RFF* is slightly below *GP-MV* / *VGPCR*, whereas *VFF* is slightly above them. This is the natural and logical behavior of the proposed pair of methods: since *VFF* optimizes over the Fourier frequencies, it is more computationally demanding than *RFF*; on the other hand, it manages to achieve more accurate results. This latter advantage will be more noticeable in the next experiment, where many more training points will be available to learn from.

Second, an essential aspect in real-world applications is the test CPU time, also known as *production time*. This amounts to the actual time that the system needs to make a prediction once it is trained. Depending on the problem at hand,

[5] When trying with $N = 20000$ for any of these methods, the RAM memory requirements exceeded the possibilities of the considered machine.

[6] These independent runs differ in the training subset if $N < 28000$, and also in the Fourier frequencies initialization for *RFF* and *VFF*.

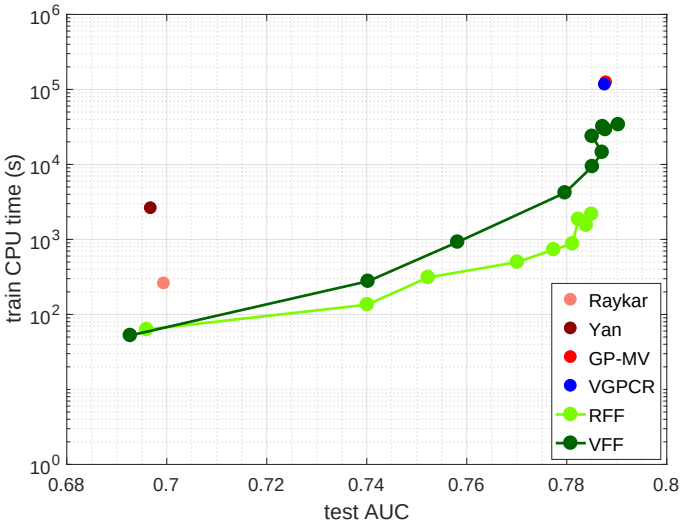

Figure 2: Trade-off between predictive performance (test AUC) and training cost (training CPU time) in the *sphere* dataset. Each method is trained with its maximum possible number of training points. For *RFF* and *VFF*, the full D_f-grids specified in the text are used. We observe that both *RFF* and *VFF* are significantly (more than three times) faster than the other competitive methods (*GP-MV* and *VGPCR*). Indeed, *VFF* manages to slightly outperform them, whereas *RFF* is around 50 times faster (and very close in predictive performance). Notice the logarithmic scale in the y-axis.

test CPU time might be more relevant than train one, since the latter affects only once whereas the former is involved in any new prediction. In our case, a fast prediction is essential to develop a practical health-care activity-recognition system that can be deployed in real nursing or retirement homes. Thus, let us now analyze the compared methods in terms of test CPU time, see Figure 3.

The difference between the proposed *RFF*/*VFF* and *GP-MV*/*VGPCR* (the only competitive methods in predictive performance, recall Figure 2) is overwhelming. Whereas the latter need more than 350 seconds to provide a prediction (for all the 3050 test instances), the former take less than 1 second. But, actually, the difference goes beyond these "absolute" numbers in this particular problem: whereas the test CPU time for the classical GP-based methods grows as $\mathcal{O}(N^2)$ with the training set size N, the novel *RFF*/*VFF* are independent on N (as expected from their theoretical formulation, recall Section 4). This fact makes classical GP-based probabilistic crowdsourcing methods prohibitive in practice for any medium-size real-world application where the production time plays an important role. Indeed, the new *RFF*/*VFF* might be the only choice for these scenarios (of course, as shown in the figure, test CPU time for LR-based methods is also independent on N, but their linear boundaries usually limit their applicability to real-world problems).

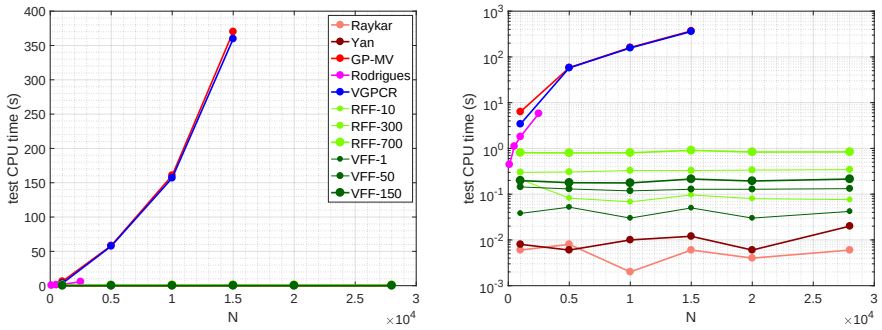

Figure 3: CPU time needed at test step (production time) as a function of the training set size in the *sphere* dataset. The linear (standard) scale in the left plot allows for a more intuitive perception of the methods scalability. The logarithmic scale in the right plot shows the differences between the fastest ones. Different representative values of D_f are shown for *RFF* and *VFF*. These are more than 350 times faster than *GP-MV* and *VGPCR* (the only competitive methods in terms of predictive performance). Moreover, as theoretically expected, their test cost is independent on N.

Among the proposed methods, notice that test CPU time grows with D_f (indeed, recall from Section 4 that their theoretical complexity is $\mathcal{O}(D_f^2)$). Therefore, and since *RFF* usually works with larger values of D_f, it is usually slightly slower than *VFF* in production time. Nonetheless, the difference is normally insignificant.

We have just seen that our methods are scalable in terms of test CPU time (in fact, they are independent on N). Let us now analyze the scalability with N in terms of training CPU time. Figure 2 already showed that *RFF*/*VFF* can be trained with $N = 28000$ instances significantly faster than *GP-MV*/*VGPCR* with $N = 15000$ (their maximum possible N). Now we examine more carefully the explicit dependence on N, see Figure 4.

This figure confirms in practice the theoretical linear-in-N training cost of the novel *RFF* and *VFF*, as well as the cubic of the classical GP-based methods (*GP-MV*, *Rodrigues*, *VGPCR*). This means that our methods can still scale up to pretty larger datasets (in fact, in the next experiment they will reach $N = 10^5$), whereas classical ones are not suitable for such scenarios. Moreover, this training CPU time explosion is not the only limitation of classical approaches. Even if we did not have training time restrictions (which, of course, is not realistic in practical applications), classical methods need to deal with $N \times N$ matrices, which implies a $\mathcal{O}(N^2)$ RAM memory cost. However, *RFF* and *VFF* substitute these matrices with $2D_f \times 2D_f$ ones, removing the quadratic dependence on N.

Figure 4 also shows that, although both *RFF* and *VFF* are linear in N, the latter is computationally more expensive than the former (because of the Fourier frequencies optimization). Finally, the extraordinary long training CPU time of *Rodrigues* is explained because the convergence process oscillates and the maximum number of iterations is reached. This might be related to the

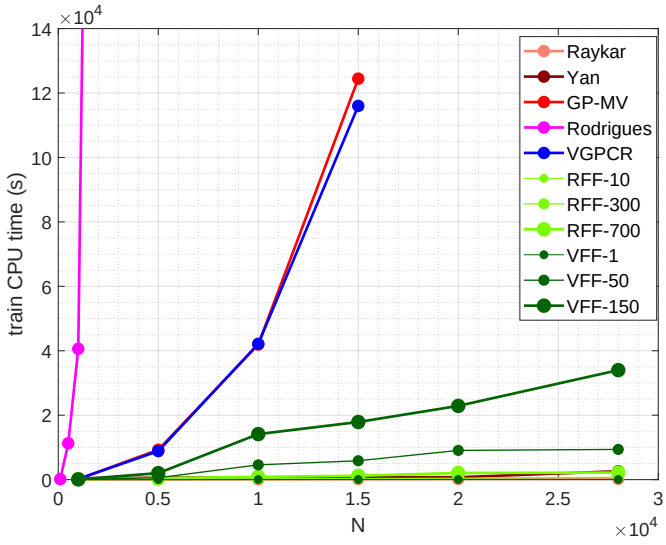

Figure 4: Training computational cost as a function of the training set size in the *sphere* dataset. Different representative values of D_f are shown for *RFF* and *VFF*. As theoretically expected, we observe a linear growth with N for the proposed methods, which makes them suitable for large-scale applications. On the contrary, classical GP-based methods cubic growth is prohibitive for that setting.

different inference procedure.

Finally, it is interesting to analyze the role of D_f in *RFF* and *VFF*, that is, how it influences their predictive performance in practice. Figure 5 addresses this question. According to their theoretical formulation (recall second-to-last paragraph in Section 2.2), increasing D_f in *RFF* improves its approximation to a GP with SE kernel. However, in *VFF* it regulates the complexity of the model and, therefore, large values might lead to overfitting to the training set. The left plot in Figure 5 confirms the simple behavior of *RFF*. Analogously, the right plot shows a more complicated behavior for *VFF*, with a slightly decreasing tendency after reaching a maximum performance. This will be also observed in the next experiment. Finally, as is natural, the performance of both methods improves with the number of training instances N.

5.2. The cubes dataset

This experiment shows that the proposed methods scale up to even larger datasets, reaching $N = 100000$ training instances. Moreover, its synthetic nature allows us to i) have access to the true labels for test instances, and ii) have true *sensitivity* and *specificity* values for the annotators (and, therefore, evaluate the accuracy of their estimation). In order to analyze the differences with the previous experiment, we simulated a classification dataset with similar

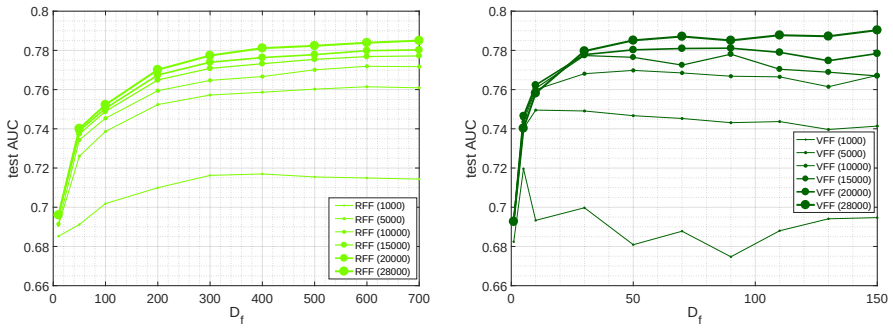

Figure 5: Predictive performance as a function of the number of Fourier frequencies used in *RFF* (left) and *VFF* (right) for the *sphere* dataset. In both cases, different training set sizes N are used. As theoretically hypothesized, *RFF* performance increases with D_f (regardless of N). However, *VFF* may suffer from over-fitting when D_f exceeds some complexity limit (which usually increases with the training set size N).

dimensionality, $D = 15$. Its structure is simple[7], and consists of a cube fitted inside a bigger one. Figure 7 shows the intuitive idea in $\mathbb{R}$ and $\mathbb{R}^2$.

More specifically, the *cubes* dataset is defined in $[-1, 1]^{15} \subset \mathbb{R}^{15}$, i.e. the 15 features are in the interval $[-1, 1]$. Training and test datasets are sampled from $[-1, 1]^{15}$ uniformly and independently. In order to define the probability that $\mathbf{x} \in [-1, 1]^{15}$ belongs to class 1, we resort to the so-called *infinity* norm, $||\mathbf{x}||_\infty = \max(|x_1|, \ldots, |x_{15}|)$. The level hyper-surfaces of this norm (i.e. the points that satisfy $||\mathbf{x}||_\infty = \text{ct.}$) are (the border of) the hyper-cubes inside $[-1, 1]^{15}$. Therefore, defining $\text{p}(y = 1|\mathbf{x}) = \varphi(||\mathbf{x}||_\infty)$ with $\varphi : [0, 1] \to \mathbb{R}$ an increasing function, we obtain a dataset in which class 1 is mainly located in the border of the $[-1, 1]^{15}$ hyper-cube whereas class 0 is mainly located in its center. More specifically, we used the function $\varphi(w) = \max\left(0, 128(w - 0.5)^7\right)$, which is represented in Figure 6. The reasons for this choice is that $\varphi(0) = 0$, $\varphi(1) = 1$, and that it generates a balanced dataset (because the measures of the subsets $\{\mathbf{x} \in [-1, 1]^{15} : 0 \leq \varphi(||\mathbf{x}||_\infty) \leq 0.5\}$ and $\{\mathbf{x} \in [-1, 1]^{15} : 0.5 \leq \varphi(||\mathbf{x}||_\infty) \leq 1\}$ are very similar).

Then, five annotators, with sensitivities $\boldsymbol{\alpha} = \{0.9, 0.7, 0.8, 0.1, 0.9\}$ and specificities $\boldsymbol{\beta} = \{0.6, 0.8, 0.5, 0.2, 0.8\}$, are simulated. This produces both very reliable annotators (e.g. the fifth) and adversarial ones (e.g. the fourth). Training and test sets with 100000 and 200000 instances, respectively, were generated. As in the previous experiment, training sets of increasing size were considered in order to examine the scalability of the compared methods, namely

[7]The more complex the dataset structure is, the more relevant it is to have a large training dataset which can reveal more detailed patterns (in other words, if the structure of the dataset is really simple, say linear, the amount of training data needed to puzzle it out reduces). Therefore, by avoiding complex dataset, we prevent the introduction of artificial complexities that could favor the proposed methods.

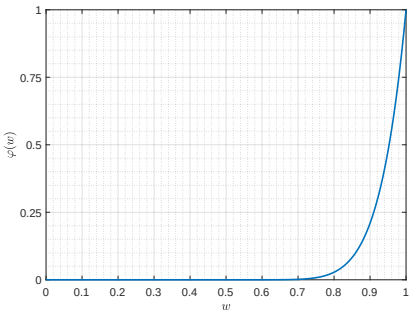

Figure 6: Graphical representation of φ, the function used to define the separation between classes in the synthetic dataset *cubes*.

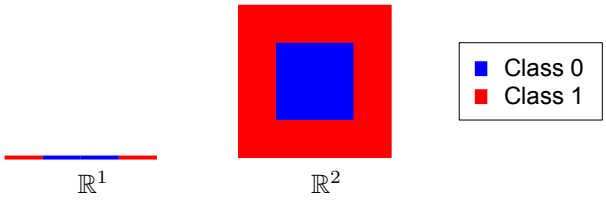

Figure 7: Structure of the two classes in the synthetic dataset *cubes*. It consists of a cube (more generally an *hyper-cube*, since we work in $\mathbb{R}^{15}$) fitted inside a bigger one. The probability of class 1 grows as we approach the border.

$N \in \{1000, 5000, 10000, 15000, 50000, 100000\}$. As before, classical GP-based methods *GP-MV* and *VGPCR* could not be trained beyond $N = 15000$, and *Rodrigues* used its own grid $N \in \{100, 500, 1000, 2500\}$ (although this time it did not exhibit convergence problems, its inference procedure is again slow in practice). Finally, the same grids $D_f \in \{10, 50, 100, 200, 300, 400, 500, 600, 700\}$ and $D_f \in \{1, 5, 10, 30, 50, 70, 90, 110, 130, 150\}$ were used for the number of Fourier frequencies in *RFF* and *VFF*, respectively. For completeness, all the raw results are shown in Appendix A, Tables A.7 and A.8 (test AUC), A.9 and A.10 (train CPU time), and A.11 and A.12 (test CPU time).

First, let us analyze the trade-off between generalization capability and training computational cost for the compared methods, see Figure 8. Again, in terms of predictive performance (see x-axis), we observe a clear distinction between LR-based methods (*Raykar*, *Yan*), which can only provide linear boundaries, and GP-related ones (*GP-MV*, *VGPCR*, *RFF*, *VFF*). *Rodrigues* is located in the middle since, although it also provides non-linear boundaries, its inference procedure limited its application to $N = 2500$ training instances (Figure 10 will analyze its lack of scalability).

Most importantly, the novel *RFF* and *VFF* exhibit the expected complementary behavior that was already observed in the previous experiment: whereas

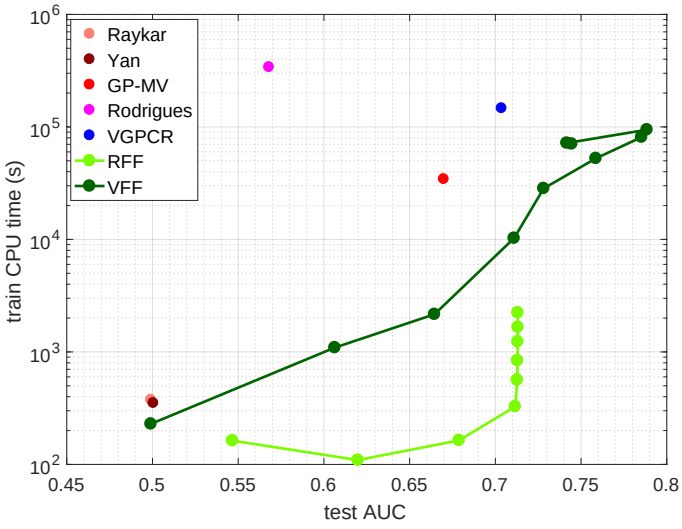

Figure 8: Trade-off between predictive performance (test AUC) and training cost (training CPU time) in the dataset *cubes*. Each method is trained with the maximum possible number of training points. For *RFF* and *VFF*, the D_f-grids specified in the text are used. We observe that *RFF* is more than 50 times faster than *VGPCR* (the other competitive method in terms of predictive performance). Moreover, it slightly outperforms *VGPCR* in that aspect. It is precisely in predictive performance where *VFF* achieves an overwhelming superiority (more than 8 points of test AUC better than *VGPCR*). Moreover, it is also faster than *VGPCR*. Notice the logarithmic scale in the y-axis.

RFF is significantly more efficient and faster (it does not optimize over the Fourier frequencies), the flexibility of *VFF* allows it to capture additional relevant patterns and, therefore, achieve a superior predictive performance. Here, notice that *RFF* with $D_f = 200$ is around 500 times faster than *VGPCR* (the only competitive method in terms of predictive performance), while it is already (slightly) better in that aspect. Moreover, *VFF* reaches a 0.788 in test AUC, whereas classical approaches get a maximum of 0.704 (*VGPCR*).

It is also interesting to observe that *VFF* with $D_f = 1$ obtains a very similar result (in test AUC) to LR-based methods (*Raykar*, *Yan*). This is reasonable according to its formulation, since it is optimizing one Fourier frequency that plays the role of the linear regression coefficients. Moreover, in Figure 11 we will analyze how the number of Fourier frequencies D_f influences the behavior of *RFF* and *VFF*.

The second main idea is the overwhelming superiority of *RFF* and *VFF* in test CPU time, see Figure 9. As in the previous experiment, their theoretical independence on N is confirmed here in practice, as opposed to the $\mathcal{O}(N^2)$ growth of the classical GP-based crowdsourcing methods. This makes the latter prohibitive for any real-world problem where the test time plays an important role. Again, we observe that the test CPU time for *RFF* and *VFF* grows with

D_f, as theoretically expected.

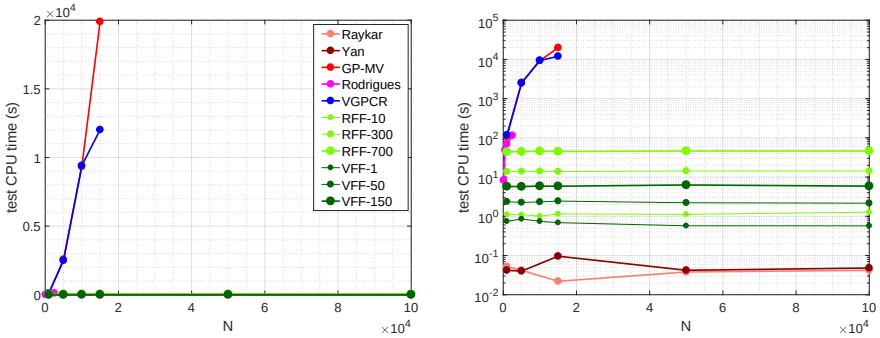

Figure 9: CPU time needed at test step (production time) as a function of the training set size in the *cubes* dataset. The linear (standard) scale in the left plot allows for a more intuitive perception of the methods scalability. The logarithmic scale in the right plot shows the differences between the fastest ones. Different representative values of D_f are shown for *RFF* and *VFF*. These are more than 250 times faster than *VGPCR* (the only competitive method in terms of generalization capability). Moreover, as theoretically expected, their test cost is independent on N.

Third, Figure 10 analyzes the train CPU time scalability of the compared methods in this large dataset. As theoretically justified, recall also the previous experiment, we confirm here that *RFF*/*VFF* growth depends linearly on N, whereas classical GP-based approaches increase with N^3. In fact, notice that our slowest method (*VFF* with $D_f = 150$ and $N = 100000$) is twice faster than *VGPCR* with $N = 15000$ (the best among the competitors, and still 9 points below in predictive performance), and very similar to *VGPCR* with $N = 10000$. This suggests that our methods can be applied to even larger datasets, whereas classical GP-based ones have already achieved their maximum capabilities in a standard machine (recall their $\mathcal{O}(N^2)$ cost in RAM memory).

Figure 10 also confirms that the Fourier frequencies optimization of *VFF* makes it significantly slower than *RFF*. Finally, notice the prohibitive growth of *Rodrigues*, which was conceived to deal with small datasets. Although the RAM memory requirements did not prevent us from training *Rodrigues* until $N = 15000$ (just like the rest of classical GP-based methods), we did not try beyond $N = 2500$ because of this very large training CPU time.

Let us now analyze how the number of Fourier frequencies D_f influences the predictive performance of the proposed methods, see Figure 11. Again, this is in accordance with their theoretical formulation (recall the second-to-last paragraph of Section 2.2) and the results obtained in the previous experiment. For *RFF*, it is simple: increasing D_f improves its approximation to a GP with SE kernel, and therefore enhances its predictive performance. For *VFF*, large values of D_f may lead to excessively complex models which overfit the training data and lose generalization capability. This produces the characteristic evolution observed in Figure 11, in which test AUC increases until an optimal value of

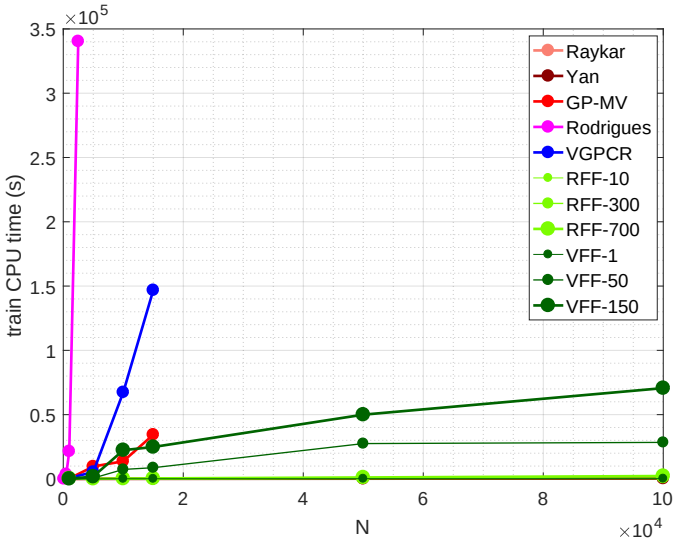

Figure 10: Training computational cost as a function of the training set size in dataset *cubes*. Different values of D_f are shown for *RFF* and *VFF*. As theoretically expected, we observe a linear growth with N for the proposed methods, which makes them suitable for large-scale applications. On the contrary, classical GP-based methods cubic growth is prohibitive for that setting. In fact, notice that our methods training with 100000 data points is faster than *VGPCR* (the best method among the competitors in terms of predictive performance) with 15000 instances (and already analogous to *VGPCR* with 10000 instances).

D_f and then decreases. Naturally, a greater training set size N usually requires a greater complexity D_f to overfit.

Finally, since we have available the real *sensitivity* and *specificity* values of the annotators, it is interesting to assess the quality of the estimations provided by *RFF* and *VFF*. We describe the case $N = 100000$, since the main goal of this work is to deal with large-scale scenarios. The results obtained for $N < 100000$ were almost identical. Table 1 show the estimations of the proposed methods for *sensitivity* and *specificity*. We observe that both *RFF* and *VFF* provide very accurate estimations for all the annotators in both *sensitivity* and *specificity*. Namely, the maximum absolute difference in *sensitivity* is 0.0118 for *RFF*, and 0.0039 for *VFF*. In *specificity*, it is 0.0123 for *RFF*, and 0.0022 for *VFF*. Moreover, the accuracy in the estimation does not depend on D_f. Notice that this is natural from a theoretical viewpoint, since D_f (the number of Fourier frequencies) is not related to the model of the annotations $\mathbf{Y}$ given the latent true labels $\mathbf{z}$ (but to the model of $\mathbf{z}$ given the features $\mathbf{X}$, recall Figure 1).

5.3. Music genre dataset

In this experiment we use the Music Genre dataset presented in [37], which consists of 1000 fragments (30 secs. length) of songs. The goal is to distinguish

Sensitivity (α), RFF										
Annot.	Real	D_f								
		10	50	100	200	300	400	500	600	700
1	0.9	0.903	0.897	0.894	0.893	0.893	0.893	0.893	0.893	0.893
2	0.7	0.704	0.696	0.692	0.691	0.691	0.691	0.691	0.691	0.691
3	0.8	0.799	0.795	0.794	0.793	0.793	0.793	0.793	0.793	0.793
4	0.1	0.091	0.101	0.107	0.108	0.108	0.108	0.108	0.108	0.108
5	0.9	0.903	0.894	0.889	0.888	0.888	0.888	0.888	0.888	0.888

Specificity (β), RFF										
Annot.	Real	D_f								
		10	50	100	200	300	400	500	600	700
1	0.6	0.594	0.603	0.608	0.609	0.609	0.609	0.609	0.609	0.609
2	0.8	0.795	0.801	0.805	0.806	0.806	0.806	0.806	0.806	0.806
3	0.5	0.500	0.505	0.508	0.509	0.509	0.509	0.509	0.509	0.509
4	0.2	0.205	0.194	0.189	0.188	0.188	0.188	0.188	0.188	0.188
5	0.8	0.792	0.804	0.810	0.811	0.811	0.811	0.811	0.811	0.811

Sensitivity (α), VFF											
Annot.	Real	D_f									
		1	5	10	30	50	70	90	110	130	150
1	0.9	0.900	0.899	0.899	0.900	0.900	0.900	0.899	0.899	0.900	0.900
2	0.7	0.700	0.700	0.699	0.700	0.700	0.700	0.700	0.700	0.700	0.700
3	0.8	0.797	0.797	0.797	0.798	0.798	0.797	0.797	0.797	0.797	0.797
4	0.1	0.096	0.097	0.097	0.096	0.096	0.096	0.097	0.097	0.096	0.097
5	0.9	0.898	0.898	0.898	0.899	0.899	0.899	0.899	0.899	0.899	0.899

Specificity (β), VFF											
Annot.	Real	D_f									
		1	5	10	30	50	70	90	110	130	150
1	0.6	0.599	0.599	0.599	0.598	0.598	0.599	0.599	0.599	0.599	0.599
2	0.8	0.798	0.798	0.798	0.798	0.798	0.798	0.799	0.799	0.798	0.798
3	0.5	0.503	0.503	0.503	0.503	0.503	0.503	0.503	0.503	0.503	0.503
4	0.2	0.199	0.199	0.198	0.200	0.200	0.199	0.199	0.199	0.199	0.199
5	0.8	0.799	0.799	0.800	0.799	0.799	0.799	0.800	0.800	0.799	0.799

Table 1: Sensitivity and specificity estimations of *RFF* and *VFF* for the five annotators in the *cubes* dataset. Different values of D_f are used, and N is set to 100000. The results are the mean over five independent runs. We observe very accurate estimations, independently on D_f.

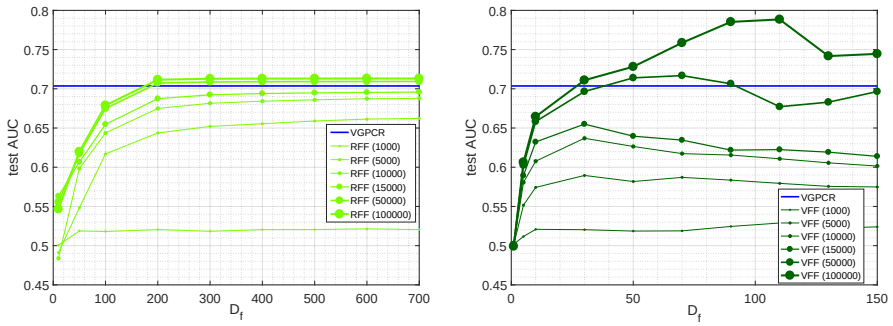

Figure 11: Predictive performance as a function of the number D_f of Fourier frequencies used in *RFF* (left) and *VFF* (right) for the *cubes* dataset. In both cases, different training set sizes N are used. As theoretically hypothesized, *RFF* performance increases with D_f (regardless of N). However, *VFF* may suffer from over-fitting when D_f exceeds some complexity limit (which usually increases with the training set size N).

between 10 music genres: classical, country, disco, hiphop, jazz, rock, blues, reggae, pop, and metal. We use an *one-vs-all* strategy to address this multi-class crowdsourcing classification problem, and the results are averaged over the 10 experiments. For preprocessing and feature extraction, the Marsyas music information tool (`http://marsyas.info/`) was used to extract 124 features from the original dataset [38]. These features include relevant technical metrics such us means and variances of timbral features, time-domain zero-crossings, spectral centroid, rolloff, flux, and Mel-Frequency Cepstral Coefficients (MFCC). The dataset contains 100 samples from each genre, which were randomly divided in 70 samples for training and 30 for testing. Crowdsourcing labels were obtained with Amazon Mechanical Turk [39]. Each annotator listened to a subset of fragments and labeled them as one of the ten genres listed above. A total amount of 2945 labels were provided by 44 different annotators.

Although *RFF* and *VFF* are initially conceived for large-scale problems out of the reach of classical GP-based crowdsourcing methods, it is interesting to analyze their behavior when applied in a small (700 training instances) real crowdsourcing problem. Figure 12 shows the predictive performance (left) and training computational cost (right) for the compared methods, using different values of D_f for *RFF*/*VFF*. Since the training is much faster now, the same fine grid $D_f = \{1, 5, 10, 20, 40, 60, 80, 100, 120, ..., 460, 480, 500\}$ was used for both methods. In all cases, the whole training set was used (i.e. $N = 700$).

Figure 12 is in accordance with the theoretical formulation of the proposed methods. *RFF* is an (efficient and scalable) approximation to *VGPCR* and, therefore, its predictive performance is limited by that of *VGPCR* (as long as they are trained with the same set, like here; the advantage of *RFF* is precisely that i) it can scale up to larger datasets, and ii) it is faster than *VGPCR* even in this small set). Consequently, in practice, and provided that *VGPCR* can handle the dataset at hand, it should be preferred to *RFF* if we are only interested in

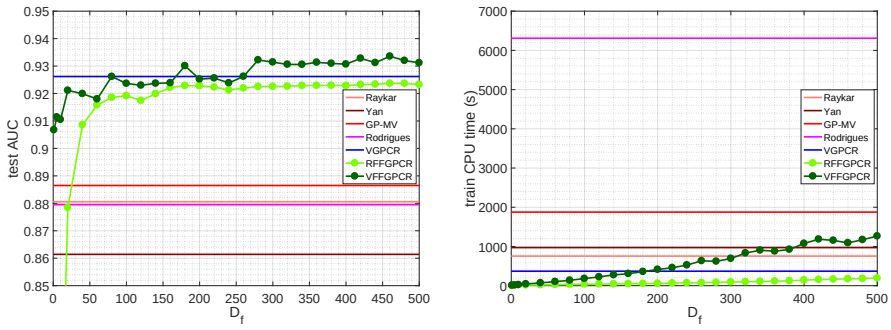

Figure 12: Left: predictive performance of the compared methods in the *Music* dataset. As theoretically expected, *RFF* constitutes an (efficient and scalable) approximation to *VGPCR*. However, *VFF* is a whole new crowdsourcing method which is also competitive with (even outperforms for some values of D_f) the state-of-the-art in small datasets. Right: training computational cost for the compared methods in the *Music* dataset. The approximation *RFF* stands out for its efficiency, whereas the new *VFF* is competitive with the rest of state-of-the-art approaches.

generalization capability. If training CPU time is an issue, the right plot shows that *RFF* becomes an interesting more efficient alternative.

For its part, since it does not approximate a GP with SE kernel but learns its own one, *VFF* is not limited by the performance of *VGPCR* (i.e., it is a whole novel approach). In fact, the left plot shows that *VFF* can outperform *VGPCR* for many choices of D_f. In any case, we observe that *VFF* is a new probabilistic crowdsourcing method that is competitive with the current ones in previously-reachable datasets. Moreover, its scalability to larger datasets makes it push further the state-of-the-art in this field.

5.4. Sentence polarity dataset

Finally, in order to further assess the robustness of the proposed methods, let us evaluate their performance in an additional application domain: sentiment analysis. More specifically, the *sentence polarity* dataset is a real crowdsourcing problem that consists of 10427 sentences extracted from movie reviews in "Rotten Tomatoes" website `http://www.rottentomatoes.com/`. The goal is to decide whether a sentence corresponds to a *positive* or *negative* review. In Table 2 we show six sentences in the dataset. Preprocessing and feature extraction were carried out by Rodrigues *et al.* [38], which resulted in feature vectors with 1200 components. The dataset is divided into train and test sets, with 4999 and 5428 samples, respectively. To obtain crowdsourcing labels, the train set was made available in Amazon Mechanical Turk. A total amount of 27746 labels were obtained from 203 different annotators.

This dataset was used to evaluate *Rodrigues* method in [13]. With 4999 training instances, it is within the reach of classical crowdsourcing approaches. Yet, let us check that our large-scale-oriented methods obtain consistent test

Table 2: Examples of positive and negative samples in *sentence polarity* dataset.

Sentence	**True Label**
"An original gem about an obsession with time." "A taut, intelligent psychological drama." "Clever, brutal and strangely soulful movie."	"positive"
"This is amusing for about three minutes." "The film can depress you about life itself." "The pool drowned me in boredom."	"negative"

results in this setting, comparing them to those reported in [13, Table 3] (namely, 0.783 and 0.781 in test AUC for *GP-MV* and *Rodrigues* respectively).

First, since *RFF* approximates the SE kernel, its performance is expected to be below that of classical methods when the same amount of training instances is used (its power is, precisely, the ability to scale up to large datasets, as shown in previous experiments). Moreover, its performance should increase with the number D_f of Fourier frequencies used, since the SE kernel is recovered when $D_f \to \infty$. These hypotheses are confirmed in the results shown in Figure 13, left plot. Notice how the test performance grows with D_f and approaches that of the previously-reported methods. Observe also that high values of D_f have been used for *RFF* (up to 3500), since the high original dimension of the data (1200 features) requires a large number of Fourier frequencies to approximate the kernel.

Second, as *VFF* learns a new kernel (which might be better suited for the data at hand), its behavior is more difficult to predict from a theoretical viewpoint. In any case, it is expected to be competitive with previous approaches in non-large-scale settings. Indeed, Figure 13, right plot, shows that it outperforms the methods reported in [13], reaching a test AUC of 0.7862 for $D_f = 500$ and 0.789 for $D_f = 1000$. Unlike *RFF*, observe that *VFF* achieves good results with significantly less Fourier frequencies, since they are optimized and therefore have a weaker dependence on the original dimension of the data.

6. Conclusions and future work

We have introduced two new scalable and efficient probabilistic crowdsourcing methods that can deal with previously-prohibitive datasets. Both are closely related to Gaussian Processes (GP), rely on the Fourier features approximation to achieve scalability, and utilize variational inference to estimate all the model unknowns. Unlike classical GP-based crowdsourcing approaches, whose training computational cost and RAM memory requirements grow as $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2)$ respectively, the proposed methods scale up linearly with the training set size N in both aspects. This allows them to go beyond the GP practical limit of $N = 10000$, reaching datasets with up to $N = 100000$ samples. In turn, this allows them to outperform the previous approaches in terms of predictive performance, while still remaining more efficient and faster. Moreover, an overwhelming superiority is achieved in test computational cost (i.e. production

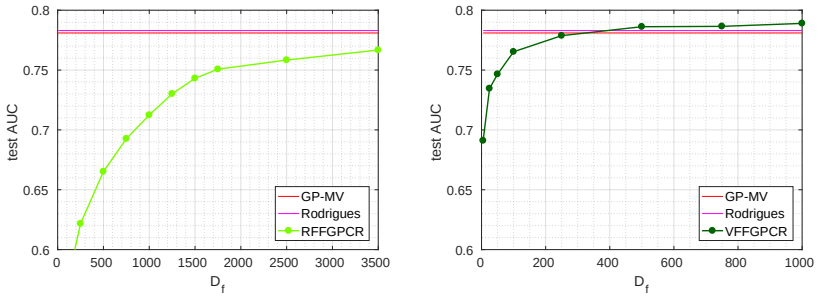

Figure 13: Predictive performance of *RFF* (left) and *VFF* (right) in the *sentence polarity* dataset, compared to the results reported in [13]. As theoretically expected, the approximated *RFF* stays below classical methods in previously-reachable datasets, becoming closer as D_f grows. However, *VFF* is a whole new crowdsourcing algorithm which is competitive with (even outperforms for some values of D_f) the previous approaches in small datasets.

time), where the novel methods are independent on N whereas classical ones grow as $\mathcal{O}(N^2)$. The novel *RFF* is a large-scale approximation to the recent GP-based crowdsourcing method *VGPCR*, while *VFF* is capable of estimating a new kernel (different to the squared exponential one that *VGPCR* is equipped with) tailored to the training data. In exchange, *VFF* is slower in practice, and more prone to overfitting. The proposed methods have proven to be the leading approach for medium-to-large scale problems. They are complementary approaches, and the final choice strongly depends on the application: whereas *RFF* guarantees a very fast and efficient training, *VFF* may achieve a higher predictive performance. Finally, the number of Fourier frequencies used, D_f, is an essential quantity in the novel approaches. As theoretically expected, more frequencies are always better for *RFF*, whereas it might lead to overfitting in *VFF*.

This is precisely the main future research line. A Bayesian treatment of the Fourier features in *VFF* could contribute to *weight* them across a wide posterior probability distribution, instead of relying on a single maximum likelihood estimation. An analogous idea has been successfully applied for regression in [31]. A multi-class crowdsourcing formulation and the use of inducing points (instead of Fourier features) to sparsify the underlying GP will also be explored in the future.

References

References

[1] J. Howe, The rise of crowdsourcing, Wired Magazine 14 (6).

[2] J. Zhang, X. Wu, V. S. Sheng, Learning from crowdsourced labeled data: a survey, Artificial Intelligence Review 46 (4) (2016) 543–576.

[3] P. Smyth, U. M. Fayyad, M. C. Burl, P. Perona, P. Baldi, Inferring ground truth from subjective labelling of Venus images, in: Advances in Neural Information Processing Systems (NIPS), 1995, pp. 1085–1092.

[4] S. Albarqouni, C. Baur, F. Achilles, V. Belagiannis, S. Demirci, N. Navab, AggNet: Deep Learning From Crowds for Mitosis Detection in Breast Cancer Histology Images, IEEE Transactions on Medical Imaging 35 (5) (2016) 1313–1321.

[5] F. Rodrigues, M. Lourenco, B. Ribeiro, F. Pereira, Learning supervised topic models for classification and regression from crowds, IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (12) (2017) 2409–2422.

[6] M. Zevin, S. Coughlin, S. Bahaadini, E. Besler, N. Rohani, S. Allen, et al., Gravity spy: integrating advanced ligo detector characterization, machine learning, and citizen science, Classical and Quantum Gravity 34 (6) (2017) 064003.

[7] F. Rodrigues, F. Pereira, Deep learning from crowds, in: Conference on Artificial Intelligence (AAAI), 2018, pp. 81–92.

[8] R. Goebel, A. Chander, K. Holzinger, F. Lecue, Z. Akata, S. Stumpf, P. Kieseberg, A. Holzinger, Explainable ai: The new 42?, in: A. Holzinger, P. Kieseberg, A. M. Tjoa, E. Weippl (Eds.), Machine Learning and Knowledge Extraction, Springer International Publishing, Cham, 2018.

[9] A. Dawid, A. Skene, Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm, Journal of the Real Statistical Society. Series C (Applied Statistics) 28 (1) (1979) 20–28.

[10] V. Raykar, S. Yu, L. Zhao, G. Hermosillo Valadez, C. Florin, L. Bogoni, L. Moy, Learning from crowds, Journal of Machine Learning Research 11 (Apr) (2010) 1297–1322.

[11] Y. Yan, R. Rosales, G. Fung, M. Schmidt, G. Hermosillo Valadez, L. Bogoni, L. Moy, J. Dy, Modeling annotator expertise: Learning when everybody knows a bit of something, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS), Vol. 9, 2010, pp. 932–939.

[12] Y. Yan, R. Rosales, G. Fung, R. Subramanian, J. Dy, Learning from multiple annotators with varying expertise, Machine Learning 95 (3) (2014) 291–327.

[13] F. Rodrigues, F. Pereira, B. Ribeiro, Gaussian process classification and active learning with multiple annotators, in: International Conference on Machine Learning (ICML), 2014, pp. 433–441.

[14] C. E. Rasmussen, C. K. I. Williams, Gaussian Processes for Machine Learning, MIT, 2006.

[15] P. Ruiz, R. Molina, A. Katsaggelos, Joint data filtering and labeling using gaussian processes and alternating direction method of multipliers, IEEE Transactions on Image Processing 25 (7) (2016) 3059–3072.

[16] P. Morales-Álvarez, A. Pérez-Suay, R. Molina, G. Camps-Valls, Remote sensing image classification with large-scale gaussian processes, IEEE Transactions on Geoscience and Remote Sensing 56 (2) (2018) 1103–1114.

[17] C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[18] T. P. Minka, A family of algorithms for approximate bayesian inference, Ph.D. thesis, Cambridge, MA, USA (2001).

[19] T. S. Jaakkola, Tutorial on variational approximation methods, in: Advanced Mean Fields Methods: Theory and Practice, MIT Press, 2000, pp. 129–159.

[20] D. M. Blei, A. Kucukelbir, J. D. McAuliffe, Variational inference: A review for statisticians, Journal of the American Statistical Association 112 (518) (2017) 859–877.

[21] E. Besler, P. Ruiz, R. Molina, A. K. Katsaggelos, Classification of multiple annotator data using variational gaussian process inference, in: European Signal Processing Conference (EUSIPCO), 2016, pp. 2025–2029.

[22] P. Ruiz, P. Morales-Álvarez, R. Molina, A. Katsaggelos, Learning from crowds with variational Gaussian Processes, Accepted for publication in Pattern Recognition (2018).

[23] N. D. Lawrence, M. Seeger, R. Herbrich, Fast sparse gaussian process methods: The informative vector machine, in: Advances in Neural Information Processing Systems (NIPS), MIT Press, 2003, pp. 625–632.

[24] E. Snelson, Z. Ghahramani, Sparse gaussian processes using pseudo-inputs, in: Advances in neural information processing systems (NIPS), 2006, pp. 1257–1264.

[25] J. Quiñonero-Candela, C. E. Rasmussen, A unifying view of sparse approximate gaussian process regression, Journal of Machine Learning Research 6 (Dec) (2005) 1939–1959.

[26] M. Titsias, Variational learning of inducing variables in sparse gaussian processes, in: International Conference on Artificial Intelligence and Statistics, Vol. 5, 2009, pp. 567–574.

[27] J. Hensman, N. Fusi, N. D. Lawrence, Gaussian processes for big data, in: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI), 2013, pp. 282–290.

[28] M. Bauer, M. van der Wilk, C. E. Rasmussen, Understanding probabilistic sparse gaussian process approximations, in: Advances in neural information processing systems (NIPS), 2016, pp. 1533–1541.

[29] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: Advances in neural information processing systems (NIPS), 2008, pp. 1177–1184.

[30] M. Lázaro-Gredilla, J. Quiñonero Candela, C. E. Rasmussen, A. R. Figueiras-Vidal, Sparse spectrum gaussian process regression, Journal of Machine Learning Research 11 (2010) 1865–1881.

[31] Y. Gal, R. Turner, Improving the gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs, in: International Conference on Machine Learning (ICML), 2015, pp. 655–664.

[32] T. P. Minka, Expectation propagation for approximate bayesian inference, in: Seventeenth Conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., 2001, pp. 362–369.

[33] P. Ruiz, J. Mateos, G. Camps-Valls, R. Molina, A. Katsaggelos, Bayesian active remote sensing image classification, IEEE Transactions on Geoscience and Remote Sensing 52 (4) (2014) 2186–2196.

[34] Y. Yan, G. Fung, R. Rosales, J. Dy, Active learning from crowds, in: Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 1161–1168.

[35] J. Yang, T. Drake, A. Damianou, Y. Maarek, Leveraging crowdsourcing data for deep active learning an application: Learning intents in alexa, in: Proc. of the 2018 World Wide Web Conference, International World Wide Web Conferences Steering Committee, 2018, pp. 23–32.

[36] N. Twomey, T. Diethe, M. Kull, H. Song, M. Camplani, S. Hannuna, X. Fafoutis, N. Zhu, P. Woznowski, P. Flach, I. Craddock, The sphere challenge: Activity recognition with multimodal sensor data, arXiv preprint arXiv:1603.00797.

[37] G. Tzanetakis, P. Cook, Musical genre classification of audio signals, IEEE Transactions on Speech and Audio Processing 10 (5) (2002) 293–302.

[38] F. Rodrigues, F. Pereira, B. Ribeiro, Learning from multiple annotators: Distinguishing good from random labelers, Pattern Recognition Letters 34 (12) (2013) 1428–1436.

[39] R. Snow, B. O'Connor, D. Jurafsky, A. Ng, Cheap and fast, but is it good?: evaluating non-expert annotations for natural language tasks, in: Conference on Empirical Methods in Language Processing (EMNLP), 2008, pp. 254–263.

Appendix A. Tables of results

This appendix contains all the results obtained in the previously-prohibitive datasets *sphere* and *cubes*. Tables A.1, A.3 and A.5 show the test AUC, train CPU time and test CPU time, respectively, in the *sphere* dataset for all the methods except for *Rodrigues* (respectively, Tables A.2, A.4 and A.6 are dedicated to *Rodrigues*). Analogously, Tables A.7, A.9 and A.11 show the test AUC, train CPU time and test CPU time, respectively, in the *cubes* dataset for all the methods except for *Rodrigues* (respectively, Tables A.8, A.10 and A.12 are dedicated to *Rodrigues*).

	N					
	1000	5000	10000	15000	20000	28000
Raykar	0.693±0.010	0.699±0.003	0.699±0.001	0.699±0.001	**0.700±0.001**	0.699±0.000
Yan	0.679±0.018	0.683±0.013	0.696±0.005	**0.698±0.003**	0.697±0.001	0.697±0.000
GP-MV	0.717±0.006	0.765±0.006	0.780±0.005	**0.788±0.004**	-	-
VGPCR	0.718±0.005	0.767±0.006	0.780±0.004	**0.788±0.003**	-	-
RFF-10	0.685±0.014	0.691±0.012	0.692±0.008	0.695±0.008	0.696±0.007	**0.696±0.008**
RFF-50	0.691±0.010	0.726±0.003	0.734±0.005	0.737±0.004	0.739±0.004	**0.740±0.004**
RFF-100	0.702±0.002	0.739±0.006	0.745±0.006	0.749±0.004	0.750±0.004	**0.752±0.003**
RFF-200	0.710±0.004	0.752±0.006	0.759±0.005	0.765±0.005	0.767±0.003	**0.770±0.002**
RFF-300	0.716±0.005	0.757±0.006	0.765±0.005	0.771±0.004	0.774±0.003	**0.777±0.003**
RFF-400	0.717±0.005	0.759±0.006	0.767±0.002	0.773±0.003	0.776±0.004	**0.781±0.003**
RFF-500	0.715±0.004	0.760±0.007	0.770±0.005	0.775±0.004	0.778±0.004	**0.782±0.003**
RFF-600	0.715±0.003	0.761±0.007	0.772±0.005	0.777±0.004	0.780±0.002	**0.784±0.001**
RFF-700	0.714±0.003	0.761±0.007	0.772±0.006	0.777±0.004	0.780±0.003	**0.785±0.001**
VFF-1	0.682±0.011	0.691±0.004	0.692±0.004	**0.693±0.003**	0.692±0.002	0.692±0.000
VFF-5	0.720±0.009	0.740±0.011	0.744±0.005	0.746±0.002	**0.747±0.010**	0.740±0.003
VFF-10	0.693±0.012	0.750±0.002	0.760±0.004	0.761±0.007	**0.762±0.007**	0.758±0.009
VFF-30	0.700±0.020	0.749±0.004	0.768±0.009	0.777±0.003	0.778±0.003	**0.780±0.004**
VFF-50	0.681±0.014	0.747±0.004	0.770±0.003	0.776±0.002	0.780±0.002	**0.785±0.001**
VFF-70	0.688±0.008	0.745±0.010	0.769±0.009	0.772±0.006	0.781±0.004	**0.787±0.002**
VFF-90	0.675±0.012	0.743±0.008	0.767±0.010	0.778±0.005	0.781±0.004	**0.785±0.005**
VFF-110	0.688±0.004	0.744±0.004	0.767±0.005	0.770±0.006	0.779±0.005	**0.788±0.002**
VFF-130	0.694±0.010	0.740±0.008	0.761±0.004	0.769±0.009	0.775±0.005	**0.787±0.004**
VFF-150	0.695±0.006	0.741±0.011	0.767±0.005	0.767±0.005	0.778±0.003	**0.790±0.003**

Table A.1: *Sphere* dataset. Mean and standard deviation of test AUC (i.e. generalization capability) over five independent runs, except for *Rodrigues* method. For each method, the highest value is bolded.

	N			
	100	500	1000	2500
Rodrigues	**0.507±0.019**	0.490±0.009	0.498±0.010	0.495±0.003

Table A.2: *Sphere* dataset. Mean and standard deviation of test AUC (i.e. generalization capability) over five independent runs for *Rodrigues*. The highest value is bolded.

	N					
	1000	5000	10000	15000	20000	28000
Raykar	13.0 ± 4.1	76.8 ± 8.2	103.9 ± 5.3	143.1 ± 7.0	199.9 ± 33.3	259.5 ± 48.5
Yan	425.4 ± 89.1	582.0 ± 114.4	695.0 ± 117.9	793.4 ± 121.8	746.8 ± 78.7	2609.0 ± 230.6
GP-MV	160.8 ± 18.2	9262.3 ± 1014.9	41953.7 ± 1725.8	124456.9 ± 3922.0	–	–
VGPCR	201.7 ± 67.0	8853.4 ± 595.5	42144.1 ± 2894.0	115996.4 ± 6836.8	–	–
RFF-10	10.9 ± 3.3	15.8 ± 4.6	19.6 ± 1.2	37.9 ± 17.5	43.5 ± 17.2	62.9 ± 28.6
RFF-50	15.5 ± 2.2	29.1 ± 4.1	53.2 ± 4.5	69.7 ± 7.0	95.8 ± 27.5	135.0 ± 62.4
RFF-100	25.3 ± 3.2	54.4 ± 4.3	112.6 ± 67.6	149.1 ± 46.7	132.5 ± 12.7	311.2 ± 185.2
RFF-200	42.1 ± 7.4	78.7 ± 3.7	154.6 ± 37.1	193.3 ± 9.6	262.2 ± 48.5	499.4 ± 158.9
RFF-300	56.2 ± 7.3	141.0 ± 31.5	264.6 ± 72.1	322.7 ± 18.5	449.0 ± 52.7	731.2 ± 193.5
RFF-400	89.0 ± 10.5	186.6 ± 8.1	344.5 ± 15.9	496.2 ± 39.0	602.3 ± 53.7	874.6 ± 166.6
RFF-500	111.0 ± 17.5	270.7 ± 11.8	548.4 ± 91.1	670.6 ± 33.3	866.9 ± 109.3	1857.0 ± 950.5
RFF-600	144.5 ± 23.6	348.6 ± 16.1	625.8 ± 4.0	1085.2 ± 349.2	1226.1 ± 189.1	1545.5 ± 137.8
RFF-700	184.0 ± 28.0	427.5 ± 31.6	866.6 ± 184.0	1198.5 ± 72.5	2079.3 ± 750.7	2171.6 ± 279.6
VFF-1	0.9 ± 0.3	7.5 ± 1.3	36.8 ± 7.6	50.0 ± 16.7	47.4 ± 15.4	52.5 ± 8.1
VFF-5	6.3 ± 1.7	44.9 ± 16.5	219.0 ± 62.1	322.9 ± 94.1	310.3 ± 64.2	277.1 ± 33.7
VFF-10	20.4 ± 5.6	105.5 ± 21.9	568.1 ± 171.9	642.3 ± 219.9	801.4 ± 170.3	922.8 ± 373.4
VFF-30	35.2 ± 3.7	337.0 ± 52.4	2478.7 ± 843.6	3596.8 ± 501.3	3418.9 ± 781.6	4187.2 ± 1159.9
VFF-50	47.3 ± 3.0	591.3 ± 71.3	4582.3 ± 1115.0	5862.0 ± 545.2	9075.0 ± 2773.2	9383.2 ± 1194.1
VFF-70	52.2 ± 4.8	815.4 ± 146.0	6786.7 ± 2407.5	9589.2 ± 1262.4	11574.1 ± 3645.6	14535.1 ± 2575.1
VFF-90	62.1 ± 2.5	1012.5 ± 165.2	9984.4 ± 2053.7	11908.1 ± 1549.2	15682.6 ± 3789.9	23716.8 ± 3709.6
VFF-110	75.8 ± 15.1	1320.9 ± 275.5	10240.8 ± 1503.8	14653.5 ± 4986.6	16443.2 ± 2009.3	29169.3 ± 6151.4
VFF-130	81.7 ± 7.0	1883.8 ± 442.7	13186.8 ± 2328.1	16614.2 ± 2184.3	22481.6 ± 4992.3	31939.8 ± 3517.6
VFF-150	80.1 ± 9.5	2027.5 ± 420.0	14133.8 ± 4156.4	17888.4 ± 2623.9	22889.4 ± 1343.3	33984.3 ± 1992.8

Table A.3: *Sphere* dataset. Mean and standard deviation of CPU train time over five independent runs, except for *Rodrigues* method.

	N			
	100	500	1000	2500
Rodrigues	163.6±79.7	11285.1±7749.3	40609.9±13363.6	757778.4±455399.7

Table A.4: *Sphere* dataset. Mean and standard deviation of CPU train time over five independent runs for *Rodrigues*.

	N					
	1000	5000	10000	15000	20000	28000
Raykar	0.006 ± 0.005	0.008 ± 0.004	0.002 ± 0.004	0.006 ± 0.008	0.004 ± 0.005	0.006 ± 0.008
Yan	0.008 ± 0.012	0.006 ± 0.005	0.010 ± 0.006	0.012 ± 0.010	0.006 ± 0.005	0.020 ± 0.011
GP-MV	6.322 ± 3.403	58.286 ± 11.929	160.682 ± 14.796	370.072 ± 33.214	–	–
VGPCR	3.412 ± 2.055	57.636 ± 13.149	156.912 ± 10.616	359.774 ± 7.491	–	–
RFF-10	0.210 ± 0.250	0.082 ± 0.016	0.068 ± 0.012	0.096 ± 0.026	0.080 ± 0.020	0.076 ± 0.010
RFF-50	0.138 ± 0.054	0.134 ± 0.023	0.120 ± 0.021	0.120 ± 0.015	0.134 ± 0.027	0.128 ± 0.015
RFF-100	0.152 ± 0.040	0.148 ± 0.015	0.146 ± 0.016	0.146 ± 0.019	0.160 ± 0.011	0.174 ± 0.019
RFF-200	0.258 ± 0.064	0.230 ± 0.020	0.234 ± 0.014	0.234 ± 0.027	0.248 ± 0.012	0.236 ± 0.019
RFF-300	0.300 ± 0.023	0.306 ± 0.010	0.328 ± 0.031	0.328 ± 0.044	0.336 ± 0.024	0.344 ± 0.031
RFF-400	0.414 ± 0.034	0.412 ± 0.024	0.420 ± 0.028	0.416 ± 0.024	0.422 ± 0.030	0.416 ± 0.021
RFF-500	0.522 ± 0.034	0.558 ± 0.051	0.556 ± 0.053	0.542 ± 0.053	0.532 ± 0.032	0.544 ± 0.036
RFF-600	0.642 ± 0.052	0.660 ± 0.043	0.648 ± 0.026	0.664 ± 0.051	0.644 ± 0.012	0.668 ± 0.024
RFF-700	0.806 ± 0.031	0.796 ± 0.047	0.802 ± 0.037	0.906 ± 0.114	0.834 ± 0.030	0.838 ± 0.031
VFF-1	0.038 ± 0.004	0.052 ± 0.031	0.030 ± 0.006	0.050 ± 0.045	0.030 ± 0.006	0.042 ± 0.020
VFF-5	0.092 ± 0.029	0.066 ± 0.016	0.060 ± 0.011	0.062 ± 0.015	0.070 ± 0.011	0.076 ± 0.036
VFF-10	0.088 ± 0.015	0.086 ± 0.014	0.074 ± 0.010	0.090 ± 0.017	0.100 ± 0.025	0.068 ± 0.007
VFF-30	0.116 ± 0.008	0.116 ± 0.014	0.116 ± 0.024	0.114 ± 0.008	0.104 ± 0.014	0.090 ± 0.009
VFF-50	0.144 ± 0.005	0.130 ± 0.014	0.118 ± 0.017	0.128 ± 0.019	0.128 ± 0.021	0.132 ± 0.017
VFF-70	0.160 ± 0.013	0.146 ± 0.030	0.134 ± 0.008	0.136 ± 0.012	0.128 ± 0.016	0.150 ± 0.017
VFF-90	0.144 ± 0.008	0.156 ± 0.015	0.138 ± 0.022	0.136 ± 0.012	0.152 ± 0.019	0.194 ± 0.027
VFF-110	0.178 ± 0.017	0.154 ± 0.015	0.166 ± 0.024	0.168 ± 0.019	0.164 ± 0.017	0.166 ± 0.015
VFF-130	0.178 ± 0.019	0.170 ± 0.024	0.178 ± 0.019	0.280 ± 0.177	0.164 ± 0.017	0.194 ± 0.015
VFF-150	0.198 ± 0.013	0.178 ± 0.007	0.176 ± 0.023	0.214 ± 0.020	0.194 ± 0.030	0.214 ± 0.019

Table A.5: *Sphere* dataset. Mean and standard deviation of CPU test time (i.e. production time) over five independent runs, except for *Rodrigues* method.

	N			
	100	500	1000	2500
Rodrigues	0.444±0.153	1.112±0.576	1.806±0.428	5.788±0.396

Table A.6: *Sphere* dataset. Mean and standard deviation of CPU test time (i.e. production time) over five independent runs for *Rodrigues*.

	N					
	1000	5000	10000	15000	50000	100000
Raykar	0.501±0.001	**0.501±0.001**	0.501±0.002	0.501±0.002	0.501±0.001	0.499±0.000
Yan	0.500±0.002	0.501±0.002	0.501±0.002	**0.501±0.002**	0.501±0.001	0.501±0.000
GP-MV	0.489±0.004	0.493±0.001	0.631±0.031	**0.670±0.010**	-	-
VGPCR	0.616±0.031	0.670±0.009	0.694±0.002	**0.704±0.001**	-	-
RFF-10	0.501±0.038	0.491±0.040	0.484±0.050	**0.563±0.005**	0.556±0.009	0.547±0.012
RFF-50	0.519±0.075	0.548±0.061	0.598±0.014	0.607±0.019	0.617±0.021	**0.620±0.020**
RFF-100	0.518±0.041	0.617±0.014	0.644±0.014	0.655±0.014	0.675±0.013	0.679±0.013
RFF-200	0.521±0.045	0.644±0.010	0.675±0.004	0.687±0.003	0.707±0.001	**0.712±0.001**
RFF-300	0.518±0.079	0.652±0.010	0.682±0.003	0.692±0.002	0.709±0.000	**0.713±0.000**
RFF-400	0.520±0.081	0.655±0.008	0.684±0.002	0.694±0.002	0.709±0.000	**0.713±0.000**
RFF-500	0.521±0.079	0.659±0.008	0.686±0.002	0.695±0.002	0.709±0.001	**0.713±0.000**
RFF-600	0.521±0.079	0.661±0.007	0.687±0.002	0.695±0.002	0.709±0.000	**0.713±0.000**
RFF-700	0.520±0.080	0.662±0.007	0.688±0.002	0.696±0.002	0.709±0.000	**0.713±0.000**
VFF-1	**0.502±0.009**	0.501±0.001	0.501±0.002	0.501±0.002	0.501±0.001	0.499±0.000
VFF-5	0.512±0.005	0.552±0.008	0.581±0.007	0.589±0.004	0.602±0.003	**0.606±0.002**
VFF-10	0.521±0.011	0.574±0.009	0.608±0.013	0.632±0.007	0.658±0.003	**0.664±0.001**
VFF-30	0.520±0.009	0.590±0.012	0.637±0.007	0.655±0.007	0.697±0.002	**0.711±0.004**
VFF-50	0.519±0.004	0.582±0.007	0.626±0.008	0.640±0.002	0.714±0.014	**0.728±0.010**
VFF-70	0.519±0.009	0.587±0.012	0.617±0.005	0.635±0.004	0.717±0.013	**0.759±0.011**
VFF-90	0.525±0.008	0.584±0.005	0.616±0.006	0.622±0.008	0.706±0.017	**0.785±0.011**
VFF-110	0.529±0.010	0.579±0.011	0.611±0.005	0.622±0.003	0.677±0.007	**0.788±0.014**
VFF-130	0.522±0.006	0.576±0.004	0.606±0.003	0.619±0.002	0.683±0.014	**0.742±0.021**
VFF-150	0.524±0.008	0.575±0.009	0.602±0.008	0.614±0.004	0.696±0.010	**0.745±0.032**

Table A.7: Mean and standard deviation of test AUC (i.e. generalization capability) over five independent runs, except for *Rodrigues* method. For each method, the highest value is bolded. Dataset: *cubes*.

	N			
	100	500	1000	2500
Rodrigues	0.501±0.003	0.528±0.006	0.544±0.011	**0.568±0.011**

Table A.8: Mean and standard deviation of test AUC (i.e. generalization capability) over five independent runs for *Rodrigues*. The highest value is bolded. Dataset: *cubes*

	N					
	1000	5000	10000	15000	50000	100000
Raykar	9.7 ± 1.3	79.7 ± 8.9	113.1 ± 10.3	143.2 ± 11.0	245.5 ± 15.1	376.4 ± 7.8
Yan	29.4 ± 11.9	72.1 ± 10.3	80.9 ± 7.3	89.0 ± 2.7	174.5 ± 1.7	351.6 ± 6.3
GP-MV	293.6 ± 306.5	9662.4 ± 6025.3	13738.3 ± 3357.0	34390.9 ± 503.5	–	–
VGPCR	639.4 ± 601.4	5390.8 ± 256.0	67373.8 ± 11094.2	146773.5 ± 31379.0	–	–
RFF-10	11.5 ± 5.2	34.3 ± 10.1	55.6 ± 17.6	61.8 ± 19.0	114.4 ± 11.8	162.7 ± 32.0
RFF-50	31.2 ± 15.8	20.2 ± 20.5	31.2 ± 21.2	31.8 ± 9.4	58.0 ± 3.1	109.3 ± 6.8
RFF-100	22.2 ± 21.4	19.3 ± 4.3	30.0 ± 4.7	39.0 ± 7.7	90.7 ± 6.0	163.7 ± 13.0
RFF-200	32.4 ± 28.8	22.9 ± 2.3	43.4 ± 2.9	59.0 ± 4.4	169.3 ± 3.9	327.3 ± 8.8
RFF-300	71.9 ± 51.6	37.5 ± 6.2	64.1 ± 3.2	87.5 ± 7.6	298.4 ± 13.3	564.2 ± 9.2
RFF-400	122.8 ± 83.0	56.2 ± 10.0	96.3 ± 9.2	137.1 ± 17.5	426.5 ± 35.3	839.5 ± 12.4
RFF-500	195.7 ± 157.4	96.8 ± 29.2	153.3 ± 27.6	215.9 ± 33.1	613.0 ± 42.5	1233.3 ± 30.2
RFF-600	277.5 ± 202.9	130.5 ± 34.9	210.8 ± 41.7	284.1 ± 53.1	813.6 ± 87.2	1661.3 ± 48.9
RFF-700	399.1 ± 304.0	167.1 ± 59.4	260.4 ± 52.9	402.0 ± 38.3	1127.6 ± 83.1	2234.7 ± 30.9
VFF-1	3.0 ± 1.2	14.4 ± 3.0	116.2 ± 34.1	124.4 ± 33.4	184.9 ± 27.6	229.2 ± 37.5
VFF-5	8.7 ± 1.1	130.6 ± 35.2	791.1 ± 497.6	608.5 ± 150.7	893.3 ± 298.6	1092.7 ± 389.3
VFF-10	21.0 ± 5.1	173.1 ± 25.3	1480.2 ± 273.7	1460.6 ± 488.5	1726.8 ± 288.2	2160.9 ± 583.5
VFF-30	48.7 ± 11.4	399.2 ± 53.2	5039.8 ± 1208.3	6020.8 ± 1427.5	7832.2 ± 2032.4	10261.9 ± 3239.3
VFF-50	61.5 ± 11.2	638.8 ± 117.8	7388.7 ± 2115.0	8720.7 ± 2936.3	27493.7 ± 15726.5	28411.7 ± 11184.2
VFF-70	70.1 ± 12.3	809.5 ± 59.5	11065.6 ± 3504.0	12704.8 ± 4026.7	32159.1 ± 11781.5	52529.9 ± 13354.0
VFF-90	77.3 ± 15.2	1292.3 ± 247.0	12141.8 ± 3080.7	17052.8 ± 4579.0	33085.1 ± 8452.8	81205.7 ± 11555.1
VFF-110	86.6 ± 18.5	1662.5 ± 383.0	19402.3 ± 5435.1	21365.0 ± 5921.8	27909.0 ± 6329.1	94428.9 ± 33174.8
VFF-130	101.7 ± 31.5	2207.8 ± 404.7	20206.9 ± 5174.2	20747.1 ± 4638.8	46683.2 ± 24627.5	71927.6 ± 18965.0
VFF-150	110.1 ± 28.5	1837.7 ± 331.1	22362.4 ± 6844.6	24831.1 ± 6319.1	49974.3 ± 16474.8	70633.5 ± 32070.8

Table A.9: Mean and standard deviation of CPU train time over five independent runs, except for *Rodrigues* method. Dataset: *cubes*

	N			
	100	500	1000	2500
Rodrigues	67.0±23.2	3646.7±919.1	21471.2±1611.1	340339.7±48419.3

Table A.10: Mean and standard deviation of CPU train time over five independent runs for *Rodrigues*. Dataset: *cubes*

	N					
	1000	5000	10000	15000	50000	100000
Raykar	0.052 ± 0.007	0.042 ± 0.010	0.040 ± 0.011	0.022 ± 0.004	0.038 ± 0.007	0.042 ± 0.004
Yan	0.042 ± 0.004	0.040 ± 0.009	0.036 ± 0.005	0.096 ± 0.117	0.042 ± 0.004	0.048 ± 0.012
GP-MV	108.418 ± 2.068	2572.136 ± 72.938	9325.962 ± 27.316	19895.748 ± 661.243	–	–
VGPCR	118.398 ± 2.023	2489.040 ± 103.969	9407.908 ± 230.540	12019.232 ± 3653.075	–	–
RFF-10	1.118 ± 0.121	1.102 ± 0.172	1.010 ± 0.171	1.148 ± 0.146	1.118 ± 0.137	1.276 ± 0.176
RFF-50	2.332 ± 0.359	2.294 ± 0.177	2.436 ± 0.257	2.478 ± 0.103	2.388 ± 0.165	2.360 ± 0.119
RFF-100	3.900 ± 0.336	4.228 ± 0.334	4.460 ± 0.332	4.322 ± 0.223	4.288 ± 0.221	4.396 ± 0.274
RFF-200	7.976 ± 0.290	8.298 ± 0.218	8.360 ± 0.433	8.656 ± 0.450	8.898 ± 0.282	9.068 ± 0.286
RFF-300	13.738 ± 1.553	13.958 ± 0.416	14.048 ± 0.755	13.818 ± 0.405	14.314 ± 0.617	14.252 ± 0.339
RFF-400	19.826 ± 1.940	19.732 ± 0.838	20.694 ± 0.913	20.406 ± 0.746	21.114 ± 0.701	20.890 ± 0.302
RFF-500	27.160 ± 2.920	27.116 ± 2.087	27.178 ± 0.520	27.354 ± 2.912	29.026 ± 1.834	28.456 ± 0.630
RFF-600	34.788 ± 3.241	35.050 ± 3.523	35.674 ± 1.310	37.036 ± 1.893	37.512 ± 0.362	36.772 ± 1.957
RFF-700	43.928 ± 3.407	44.634 ± 1.275	45.790 ± 2.873	44.960 ± 2.203	46.230 ± 0.872	46.236 ± 2.113
VFF-1	0.744 ± 0.066	0.852 ± 0.180	0.744 ± 0.108	0.688 ± 0.106	0.572 ± 0.097	0.570 ± 0.061
VFF-5	0.824 ± 0.125	0.872 ± 0.122	0.942 ± 0.188	0.744 ± 0.082	0.770 ± 0.119	0.758 ± 0.068
VFF-10	1.092 ± 0.106	1.154 ± 0.159	1.102 ± 0.115	0.908 ± 0.101	0.840 ± 0.117	1.016 ± 0.300
VFF-30	1.804 ± 0.100	1.662 ± 0.156	1.694 ± 0.051	1.712 ± 0.118	1.554 ± 0.083	1.634 ± 0.122
VFF-50	2.362 ± 0.076	2.254 ± 0.088	2.326 ± 0.092	2.442 ± 0.184	2.214 ± 0.152	2.160 ± 0.189
VFF-70	2.842 ± 0.107	2.760 ± 0.154	2.974 ± 0.065	2.992 ± 0.151	3.004 ± 0.079	2.956 ± 0.190
VFF-90	3.576 ± 0.132	3.504 ± 0.259	3.552 ± 0.130	3.688 ± 0.279	3.872 ± 0.352	3.622 ± 0.178
VFF-110	4.206 ± 0.175	4.062 ± 0.213	4.264 ± 0.181	4.556 ± 0.443	4.514 ± 0.313	4.284 ± 0.174
VFF-130	5.072 ± 0.430	4.868 ± 0.306	5.030 ± 0.237	5.008 ± 0.158	5.332 ± 0.166	5.240 ± 0.293
VFF-150	5.770 ± 0.321	5.716 ± 0.122	5.892 ± 0.281	5.850 ± 0.267	6.290 ± 0.915	5.878 ± 0.425

Table A.11: Mean and standard deviation of CPU test time (i.e. production time) over five independent runs, except for *Rodrigues* method. Dataset: *cubes*

	N			
	100	500	1000	2500
Rodrigues	8.294±4.201	48.528±34.869	69.630±31.114	114.842±88.935

Table A.12: Mean and standard deviation of CPU test time (i.e. production time) over five independent runs for *Rodrigues*. Dataset: *cubes*