Robust Bayesian Kernel Machine via Stein Variational Gradient Descent for Big Data

Khanh Nguyen
nkhanh@deakin.edu.au
Deakin University
Australia

Trung Le, Tu Dinh Nguyen, Dinh Phung, Geoffrey I. Webb
{trung.le1, tu.dinh.nguyen, dinh.phung, geoff.webb}@monash.edu
Monash University
Australia

ABSTRACT

for their strong generalization ability, especially on limited data to effectively generalize on unseen data. However, most kernel methods, including the state-of-the-art LIBSVM, are vulnerable to the curse of kernelization, making them infeasible to apply to large-scale datasets. This issue is exacerbated when kernel methods are used in conjunction with a grid search to tune their kernel parameters and hyperparameters which brings in the question of model robustness when applied to real datasets. In this paper, we propose a robust *Bayesian Kernel Machine* (BKM) – a Bayesian kernel machine that exploits the strengths of both the Bayesian modelling and kernel methods. A key challenge for such a formulation is the need for an efficient learning algorithm. To this end, we successfully extended the recent Stein variational theory for Bayesian inference for our proposed model, resulting in fast and efficient learning and prediction algorithms. Importantly our proposed BKM is resilient to the curse of kernelization, hence making it applicable to large-scale datasets and robust to parameter tuning, avoiding the associated expense and potential pitfalls with current practice of parameter tuning. Our extensive experimental results on 12 benchmark datasets show that our BKM without tuning any parameter can achieve comparable predictive performance with the state-of-the-art LIBSVM and significantly outperforms other baselines, while obtaining significantly speedup in terms of the total training time compared with its rivals.

KEYWORDS

Kernel methods, Stein divergence, random feature, multiclass supervised learning, Bayesian inference, variational method, big data

ACM Reference Format:
Khanh Nguyen, Trung Le, Tu Dinh Nguyen, Dinh Phung and Geoffrey I. Webb. 2018. Robust Bayesian Kernel Machine via Stein Variational Gradient Descent for Big Data. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3219819.3220015

Acknowledgment: This work is partially supported by the Australian Research Council under the Discovery Projects DP160109394 and DP150100031

1 INTRODUCTION

Kernel methods have become essential tools in machine learning and widely applied with enormous success in various domains [11]. The most popular and successful model which stands the test-of-time is the Support Vector Machine (SVM) which was first proposed in [6] for binary classification, and later extended for multiclass classification in [7]. LIBSVM [3], the most well-known implementation of SVM, has become very popular with over 35,500 citations so far and widely applied in many application domains due to its competitive and stable predictive performances over a wide range of real-world datasets. The underlying idea of kernel methods is to map data from input space to feature space via a transformation and then learn a simple geometric shape (i.e., an optimal hyperplane or hypersphere) in this high and infinite dimensional feature space. This offers kernel methods a strong generalization capacity that allows their models trained on limited data to be able to predict well unseen data and a powerful ability to capture nonlinear correlation between data and labels. However, since the feature space accompanied with the most effective kernel (i.e., RBF and Gaussian kernels) is infinitely dimensional in the Hilbert space, models in this feature space cannot be stored directly. Alternatively, these models are stored as a linear combination of feature vectors of training examples, hence parameters grow linearly with training data size, leading to a highly expensive computation when computing the dot product between models and feature vector, which is a crucial operation in kernel methods. This issue, known as the curse of kernelization, makes kernel methods scale poorly with the training size [2, 22] and more seriously in applying to large-scale datasets. Another important issue of kernel methods is how to tune both kernel and hyperparameters (i.e., regularization parameters) which may vary greatly for real-world datasets.

A popular and widely-used approach for tuning kernel and hyperparameters is to perform grid search in which a large range of combinations of parameters are tried [12]. This requires a vast number of trials which grows exponentially with the number of parameters explored, consequently placing a high burden on computational systems. This is especially serious for kernel methods which are vulnerable to the curse of kernelization and scale poorly with training size. Moreover, the values of parameters can be continuous and unbounded whilst the grid contains discrete values only, hence there is no guarantee that the tuned parameters are optimal. To learn kernel parameters only, a gradient-based approach was used in [4, 13] and the modification of a kernel alignment [8] metric in a two-stage procedure was proposed in [5]. However, these methods still suffer the curse of kernelization, hence not effective at handling large-scale datasets. An alternative approach that can indirectly learn a kernel is via multiple kernel learning (MKL) which attempts to learn combination of a predefined list of kernels [10]. However, these works typically introduced more workload to kernel methods, hence cannot scale with large-scale datasets well.

One can alleviate the time-consuming drawback of the grid search approach by avoiding the curse of kernelization. The first solution is to use a budget strategy whose underlying idea is to keep the model size fixed by using a budget maintenance procedure such as: *removal* [9], *merging* [19] or *projection* [25]. Removal removes completely the least affected data points whenever model size exceeds budget, hence can compromise predictive performance especially when training dataset grows. Merging and projection tackle removed data points more effectively by keeping their information. However, they also further increase the computational burden. Therefore, these aforementioned methods neither obtained low predictive performance nor scale with large-scale datasets. Fourier random feature [21] is an another recent efficient way to address the curse of kernelization. Using this technique, the curse of kernelization can be avoided by explicit representation and storing of model parameters in the Hibert space, hence could potentially scale up kernel methods [14, 16]. Nonetheless, the Fourier random elements in these methods are drawn from a distribution involving a kernel parameter, hence introduce a different kind of challenge. Recently, the works of [1, 18] proposed using the reparameterization trick to shift the source of randomness to a fixed distribution, transforming the kernel parameter learning problem into an optimization problem. This optimization is, however, highly non-covex, resulting in learned kernel parameter sensitive to its initialization and easy to get stuck in suboptimal local minima.

Bayesian inference, on the other hand, is a powerful building block for many machine learning algorithms. This has been combined with kernel machines to leverage its ability in tackling variance in data and infering necessary parameters with strong generalization capacity of kernel machines [17, 23, 26, 27]. The underlying idea of these methods is to sample models and parameters from posterior distributions using Gibbs samplers. However, due to the slow convergence of Gibbs sampling, these methods are not applicable to large-scale datasets. Recently, the work of [15] proposed the Stein Variational Gradient Descent (SVGD) framework to enable efficiently sampling from a distribution for which an unnormalized version is known. In particular, a set of particles which are at first sampled from a prior distribution are moved in small, finite steps to gradually approach those sampled from the target distribution. Two advantageous features of SVGD are that the formula for each move is in a closed form and it can be theoretically guaranteed that the particles gradually converge toward the target distribution.

Combining these strengths together, this paper leverages the SVGD framework for Bayesian inference with kernel methods to propose the robust *Bayesian Kernel Machine* (BKM)– a Bayesian kernel machine which is free from the curse of kernelization and robust against parameter tuning, and hence can efficiently deal with large-scale datasets. Our model development proceeds is as follows: (1) we first view the optimization problem of the proposed method as a maximum a posteriori (MAP) estimate that bridges us to a posterior inference problem and (2) we then apply the SVGD framework to enable us to sample from the posterior distribution. In sum, our contribution in this paper includes:

- We successfully extend SVGD framework for Bayesian inference with kernel methods to propose a novel *Bayesian Kernel Machine* (BKM) that is scalable to large-scale datasets without the requirement of parameter tuning. This is because our BKM learns kernel parameter directly via the SVGD, while other parameters are not sensitive to predictive performance, hence can be set to default parameters without significantly compromising predictive performance (cf. Section 4.2).
- We empirically found that although the works of [1, 18] can theoretically learn kernel parameters via solving an optimization problem using stochastic gradient descent based methods, these methods are very sensitive to the initialization of kernel parameter and learning rate. Consequently, inappropriate values for these parameters may lead to unsatisfactory experimental results, hence a grid search on the initialization of kernel parameter and learning rate are unavoidable (cf. Section 4.3). The reason is that the objective functions in these methods are highly non-convex, hence making them easy to get stuck in suboptimal local minimas with inappropriate initializations. In contrast, our BKM is developed in the spirit of Bayesian inference with recent well-grounded Stein inference method, hence resulting in very stable predictive performances on a wide range of real-world datasets (cf. Section 4.3).
- We conduct extensive experiments to analyze behaviors and validate our proposed method. We also establish the experiments extensively on 12 benchmark datasets to compare our proposed BKM with the state-of-the-art baselines in terms of predictive performance and total running time. The experimental results demonstrate that our proposed BKM without parameter tuning can obtain comparable predictive performances with the state-of-the-art LIBSVM, while significantly outperforming other baselines. Regarding the total time taken for training, our proposed method gains a magnitude of speedups compared with the baselines.

2 RELATED BACKGROUND

In this section, we briefly present important related background, starting with the random feature space for kernel methods. We then describe the reparameterization trick to make kernel parameters become learnable. Finally, we summarize the Stein Variational Gradient Descent (SVGD) framework to learn an arbitrary distribution.

2.1 Multiclass Classification with Random Feature Space

Consider a multiclass supervised learning problem with training set $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ where $\mathbf{x}_n \in \mathcal{X} \subset \mathbb{R}^d$ and $y_n \in \mathcal{Y} = \{1, 2, \ldots, M\}$. We aim to learn M hyperplanes $\{\mathbf{w}_m\}_{m=1}^{M}$, each corresponds to a class label. For notation convenience, we stack these vectors into a matrix $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_M]^\mathsf{T}$ where $\mathbf{W}$ is now the parameter to be learned via the following optimization problem:

$$\min_{\mathbf{W}} \left\{ \frac{\lambda}{2} \|\mathbf{W}\|_{2,2}^2 + \sum_{n=1}^{N} l(\mathbf{W}; \mathbf{x}_n, y_n) \right\} \tag{1}$$

where $l(\mathbf{W}; \mathbf{x}, y)$ is the loss function.

In general, a suitable loss function $l(\mathbf{W}; \mathbf{x}, y)$ to penalize incorrect classification can be formulated using the true information from label y and the discriminative score for $\mathbf{x}$ computed as $\mathbf{w}_m^\mathsf{T}\mathbf{x}$ for

$m = 1, \ldots, M$ [7]. For the multiclass classification problem of our current interest, two most popular loss functions are: the multi-class Hinge loss and the cross-entropy loss with softmax function.

- *Multiclass Hinge loss* [7]. Given training sample $(\mathbf{x}, y)$, this loss function is defined over the gap between the true and maximal discriminative scores on the remainder class labels:
$$l(\mathbf{W}; \mathbf{x}, y) = \max\left\{0, 1 + \max_{m \neq y}(\mathbf{w}_m^{\mathsf{T}}\mathbf{x}) - \mathbf{w}_y^{\mathsf{T}}\mathbf{x}\right\}$$
This is a convex and non-smooth loss function, hence we have a set of sub-gradients at each point and sub-gradients at different points are typically not smoothly varied.
- *Cross-entropy loss.* We first use the softmax function to compute the probability of classifying $\mathbf{x}$ to class label j as
$$p_j(\mathbf{x}) = \exp\left(\mathbf{w}_j^{\mathsf{T}}\mathbf{x}\right) / \sum_{m=1}^{M} \exp(\mathbf{w}_m^{\mathsf{T}}\mathbf{x}),\ m = 1, \ldots, M$$
Cross-entropy loss is then defined as cross-entropy between the one-hot vector of the true label and the softmax probability, which becomes:
$$l(\mathbf{W}; \mathbf{x}, y) = -\log\left(p_y(\mathbf{x})\right)$$
Similar to multi-class Hinge loss, the cross-entropy loss with softmax is *convex*, however unlike Hinge loss, it is *smooth.*

The above formulation is for the linear case. To capture non-linear relationships between data and labels, the kernel trick can be applied wherein a data point $\mathbf{x}$ in the input space is replaced by a feature vector $\mathbf{\Phi}(\mathbf{x})$ in the feature space via a transformation $\mathbf{\Phi}(\cdot)$ defined by a kernel function $K(\cdot,\cdot)$ that maps from the input space to the feature space. In most kernel methods, model $\mathbf{w}$ is stored as a weighted linear sum of feature vectors in training set as $\mathbf{w} = \sum_{n=1}^{N} \alpha_n \mathbf{\Phi}(\mathbf{x}_n)$. As a consequence, the model size which is define as $\|\alpha\|_0$ linearly depends on the training size. This fact is known as the curse of the kernelization [24]. This problem significantly influences the computational burden on both training and testing processes since any dot product computation between the model $\mathbf{w}$ and a feature vector $\mathbf{\Phi}(\mathbf{x})$ is a linear sum of N kernel operations where N is the training size which can be extremely large as follows:

$$\mathbf{w}^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n \mathbf{\Phi}(\mathbf{x}_n)^{\mathsf{T}} \mathbf{\Phi}(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_n, \mathbf{x}) \tag{2}$$

To overcome the curse of kernelization issue, a random feature mapping $\boldsymbol{\phi}(\mathbf{x})$ [21] was proposed to construct an explicit representation of $\mathbf{\Phi}(\mathbf{x})$. We briefly summarize this technique and refer to [18] for full details. Given a Gaussian kernel $K(\mathbf{x}, \mathbf{x}') = \exp\left(-0.5(\mathbf{x}-\mathbf{x}')^{\mathsf{T}} \operatorname{diag}(\boldsymbol{\gamma})(\mathbf{x}-\mathbf{x}')\right)$ parameterized by kernel width $\boldsymbol{\gamma}$, the Fourier random feature map can be constructed as:

$$\boldsymbol{\phi}(\mathbf{x}) = \frac{1}{\sqrt{D}}\left[\cos\left(\boldsymbol{\omega}_i^{\mathsf{T}}\mathbf{x}\right), \sin\left(\boldsymbol{\omega}_i^{\mathsf{T}}\mathbf{x}\right)\right]_{i=1}^{D}$$

where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \operatorname{diag}(\boldsymbol{\gamma}))$ and its induced kernel $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathsf{T}} \boldsymbol{\phi}(\mathbf{x})$ can accurately and effectively approximate the original kernel $K(\mathbf{x}, \mathbf{x}')$. This formulation, however, does not allow to learn the kernel parameter $\boldsymbol{\gamma}$. A recent solution [1, 18] is to shift the source of randomness, i.e., sampling $\boldsymbol{\omega}_i$, to $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, then the frequency $\boldsymbol{\omega}_i$ is reparameterized as $\boldsymbol{\omega}_i = \boldsymbol{\gamma} \odot \mathbf{e}_i$ where $\mathbf{e}_i \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, $i = 1, \ldots, D$.

2.2 Stein Variational Gradient Descent (SVGD)

Stein Variational Gradient Descent (SVGD) aims to learn a target distribution $p(\mathbf{W})$ by moving an initial distribution $q_0(\mathbf{W})$ step by step towards $p(\mathbf{W})$ using a bijective transformation in each step. This is a powerful tool for us to sample from the target distribution $p(\mathbf{W})$ if we know its any unnormalized version, hence allowing us to sample from the posterior of both model and kernel parameters conditional on data in our problem. In the next section, we will present how to cast our optimization formulation to a posterior inference problem; and how to apply the SVGD framework to sample from this posterior. In what follows, we revise the SVGD framework proposed in [15].

The underlying idea of SVGD originates from variational inference. To approximate a target distribution $p(\mathbf{W})$, a variational inference approach uses a simpler distribution $q(\mathbf{W})$ found in a distribution family $\mathcal{Q}$. The optimal $q^*(\mathbf{W})$ in $\mathcal{Q}$ that best approximates $p(\mathbf{W})$ can be obtained by minimizing the Kullback–Leibler (KL) divergence between q and p. In [15], the distribution family $\mathcal{Q}$ is a set of distributions $q_{[\mathbf{T}]}(\mathbf{W})$ obtained by applying smooth transformations $\mathbf{T}(\mathbf{W})$ to a tractable reference distribution $q_0(\mathbf{W})$. The optimization problem (OP) for finding the optimal $\mathbf{T}^*$ can be written as follows:

$$\min_{\mathbf{T}} KL\left(q_{[\mathbf{T}]} \parallel p\right) \tag{3}$$

Solving the optimization problem in Eq. (3) where the transformation $\mathbf{T}$ is highly free is really challenging, thus SVGD restricts findings to transformation $\mathbf{T}$ that can be formulated as a composition of many simpler bijective transformations. In particular, we limit ourselve to searching in the transformation family: $\mathbf{T}(\cdot) = \mathbf{G}_L(\mathbf{G}_{L-1}(\cdots(\mathbf{G}_2(\mathbf{G}_1(\cdot)))))$ where each $\mathbf{G}_l(\cdot)$ is a simple bijective transformation. In particular, each $\mathbf{G}$ is formed by a small perturbation of the identity map, i.e., $\mathbf{G}(\mathbf{W}) = \mathbf{W} + \epsilon\boldsymbol{\varphi}(\mathbf{W})$ where $\boldsymbol{\varphi}(\mathbf{W})$ is a smooth function controlling the perturbation direction and ϵ is the perturbation magnitude.

The original optimization problem in Eq. (3) is now decomposed to many sub-optimization problems where in each sub-problem, we have a current distribution $q(\mathbf{W})$ and attempt to learn a simple bijective transformation $\mathbf{G}(\mathbf{W}) = \mathbf{W} + \epsilon\boldsymbol{\varphi}(\mathbf{W})$ that minimizes:

$$\min_{q_{[\mathbf{G}]}}\left(KL\left(q_{[\mathbf{G}]} \parallel p\right)\right) = \min_{\epsilon, \boldsymbol{\varphi}}\left(KL\left(q_{[\mathbf{G}]} \parallel p\right)\right)$$

In other words, we want to find ϵ, $\boldsymbol{\varphi}$ of the optimal transformation $\mathbf{G}^*$, and the current distribution q to minimize the corresponding KL divergence. Note that $q_{[0]} = q$ and the derivative of the KL divergence between $q_{[\mathbf{G}]}$ and p w.r.t ϵ at the current standpoint (i.e., $\epsilon = 0$) is of the following form:

$$\nabla_\epsilon\ KL(q_{[\mathbf{G}]} \parallel p)\Big|_{\epsilon=0} = -\mathbb{E}_{\mathbf{W}\sim q}\left[\operatorname{tr}\left(\mathcal{A}_{p,\boldsymbol{\varphi}}(\mathbf{W})\right)\right]$$

where $\mathcal{A}_{p,\boldsymbol{\varphi}}(\mathbf{W}) = \nabla_{\mathbf{W}} \log p(\mathbf{W})\,\boldsymbol{\varphi}(\mathbf{W})^{\mathsf{T}} + \nabla_{\mathbf{W}}\boldsymbol{\varphi}(\mathbf{W})$ is known as the Stein operator. To achieve the steepest descent of the KL divergence, the optimal $\boldsymbol{\varphi}^*$ can be identified as follows:

$$\boldsymbol{\varphi}^*(\cdot) = \mathbb{E}_{\mathbf{W}\sim q}\left[\kappa(\mathbf{W}, \cdot)\nabla_{\mathbf{W}} \log p(\mathbf{W}) + \nabla_{\mathbf{W}}\kappa(\mathbf{W}, \cdot)\right] \tag{4}$$

where $\kappa(\mathbf{W}, \mathbf{W}')$ is a p.s.d kernel function (that is different from the kernel K in Eq. (2)). With the optimal $\boldsymbol{\varphi}^*$, we have $\nabla_\epsilon\ KL(q_{[\mathbf{G}]} \parallel p)\Big|_{\epsilon=0} = -\mathbb{D}^2(q, p)$ where $\mathbb{D}(q, p) = \|\boldsymbol{\varphi}^*\|$ is the kernelized Stein discrepancy between q and p.

The above sub optimization problem is applied sequentially to find out the optimal bijection transformations $\mathbf{G}_1^*, \cdots, \mathbf{G}_L^*$ whose composition forms the resulting optimal transformation $\mathbf{T}^*$, i.e., $\mathbf{T}^*(\cdot) = \mathbf{G}_L^*(\mathbf{G}_{L-1}^*(\cdots(\mathbf{G}_2^*(\mathbf{G}_1^*(\cdot)))))$. In [15], the distributions $q_{[\mathbf{T}_l^*]}$ where $\mathbf{T}_l^*(\cdot) = \mathbf{G}_l^*(\mathbf{G}_{l-1}^*(\cdots(\mathbf{G}_2^*(\mathbf{G}_1^*(\cdot)))))$ were proven to reach closer the target distribution p when l gradually increases and can approach to p up to any level of precision given sufficiently large number of steps l.

In practice, to sample from the target distribution p, a set of S particles $\{\mathbf{W}^j\}_{j=1}^S$ are drawn from the initial distribution q_0, they are updated iteratively using the transformation $\mathbf{G}_l^*$ at iteration l as follows

$$\mathbf{W}_{l+1}^j = \mathbf{W}_l^j + \tau_l \boldsymbol{\varphi}^*\left(\mathbf{W}_l^j\right) \tag{5}$$

where τ_l is a small step size decaying over time and $\boldsymbol{\varphi}_l^*(\mathbf{W})$ is the optimal perturbation direction which can be approximated as:

$$\boldsymbol{\varphi}^*(\mathbf{W}) = \frac{1}{S}\sum_{s=1}^{S}\left[\kappa\left(\mathbf{W}, \mathbf{W}^s\right)\nabla_{\mathbf{W}}\log p(\mathbf{W}) + \nabla_{\mathbf{W}}\kappa\left(\mathbf{W}, \mathbf{W}^s\right)\right] \tag{6}$$

To decay the step size τ_l, an Adagrad update style can be used. In particular, the step size τ_l at the l-th iteration is computed as

$$\tau_l = \varepsilon/(\delta+\sqrt{r_l}) \text{ and } r_l = r_{l-1} + \mathbf{g} \odot \mathbf{g}$$

where $\mathbf{g} = \nabla_{\mathbf{W}}\log p(\mathbf{W})$ and $r_0 = 0$.

One advantage of SVGD stems from Eq. (6). There are two terms with mutual support. The first term is a weighted sum of gradients of $\log p(\mathbf{W})$ which yields all particles to high density areas of $p(\mathbf{W})$. However, with only this term, all particles are collapsed to the mode of $p(\mathbf{W})$ which cannot characterize the whole distribution $p(\mathbf{W})$. Fortunately, it will not happen because the second term acts as repulsive force. In particular, for the case κ is a Gaussian kernel, the derivative of κ w.r.t the first parameter is $\nabla_{\mathbf{W}}\kappa(\mathbf{W}, \mathbf{W}^s) = -1/\sigma^2(\mathbf{W} - \mathbf{W}^s)\kappa(\mathbf{W}, \mathbf{W}^s)$ shows that $\mathbf{W}$ are pushed far away from $\mathbf{W}^s$ when $\mathbf{W}$ and $\mathbf{W}^s$ are very near, i.e., $\kappa(\mathbf{W}, \mathbf{W}^s)$ is large.

In addition, the most appealing feature of SVGD is that it only requires an unnormalized version $\bar{p}$ of the distribution p to compute $\boldsymbol{\varphi}^*$ in Eq. (6) since we have:

$$\nabla_{\mathbf{W}}\log\bar{p}(\mathbf{W}) = \nabla_{\mathbf{W}}(\log\bar{p}(\mathbf{W}) + \log Z) = \nabla_{\mathbf{W}}\log p(\mathbf{W})$$

where $Z = \int \bar{p}(\mathbf{W})\,d\mathbf{W}$ is the normalization constant and hence $p(\mathbf{W}) = \frac{1}{Z}\bar{p}(\mathbf{W})$. More discussion details of these advantages can be found in [15]

3 FRAMEWORK OF BAYESIAN KERNEL MACHINE WITH STEIN DIVERGENCE

3.1 Problem Formulation

Under the random feature framework with feature map $\boldsymbol{\phi}(\cdot)$ (cf. Section 2.1), we aim to learn M hyperplanes $\{\mathbf{w}_m\}_{m=1}^M$ in the random feature space, each corresponding to a class label. The OP (1) can be generalized as follows:

$$\min_{\mathbf{W}}\left\{J(\mathbf{W}) = \Omega(\mathbf{W}) + \mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}\left[l(\mathbf{W}; \mathbf{x}, y)\right]\right\} \tag{7}$$

where $\Omega(\mathbf{W})$ is a suitable regularization term of our choice (e.g., $\frac{\lambda}{2}\|\mathbf{W}\|_{2,2}^2$), $l(\mathbf{W}, \mathbf{x}, y)$ is the loss function, and $\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$ is the joint data distribution over $\mathcal{X}\times\mathcal{Y}$.

3.2 Optimization via Posterior Inference

Many recent works have successfully shown that solving an optimization problem in the form similar to Eq. (7) can be efficiently transformed into a posterior inference problem under a Bayesian setting (e.g., see [17, 20]).

Our approach adopts a similar strategy to transform the OP (7) into a posterior inference problem, which enables us to develop a novel Stein Variational Gradient Descent (SVGD) to efficiently solve it. Since $\exp(\cdot)$ is a monotonic function, solving the OP (7) is equivalent to $\max_{\mathbf{W}}\{\exp(-J(\mathbf{W}))\} =$

$$\max_{\mathbf{W}}\left\{\exp(-\Omega(\mathbf{W})) \times \prod_{(\mathbf{x},y)\in\mathcal{D}}\exp(-p(\mathbf{x}, y)\,l(\mathbf{W}; \mathbf{x}, y))\right\} \tag{8}$$

where we have used the empirical distribution induced from training data $\mathcal{D}$ in the place for $\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$.

With respect to our parameter of interest $\mathbf{W}$, the first and second terms in Eq. (8) can be respectively viewed as the prior and data likelihood to give rise to the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$, i.e.:

$$\begin{aligned} p(\mathbf{W}) &\propto \exp(-\Omega(\mathbf{W})) \\ p(\mathcal{D} \mid \mathbf{W}) &\propto \prod_{(\mathbf{x},y)\in\mathcal{D}}\exp(-p(\mathbf{x}, y)\,l(\mathbf{W}; \mathbf{x}, y)) \\ p(\mathbf{W} \mid \mathcal{D}) &\propto \bar{p}(\mathbf{W} \mid \mathcal{D}) = \exp(-J(\mathbf{W})) \end{aligned} \tag{9}$$

Solving the OP (8) is now a maximum a posteriori (MAP) problem w.r.t the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$.

To sum up, to learn M hyperplanes as originally formulated in (7), we can alternatively solve a MAP problem w.r.t the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$ defined in (8). However, it is clear that we know $p(\mathbf{W} \mid \mathcal{D})$ through (8) *only up to a normalization constant*. Hence, it presents a general challenge to compute its MAP as typically encountered in a Bayesian inference problem.

To this end, one key contribution in this paper is to develop a novel SVGD under this Bayesian formulation to effectively draw samples for the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$ from its unnormalized version $\bar{p}(\mathbf{W} \mid \mathcal{D})$, and consequently solve its MAP presented as in Section 3.3.

To avoid kernel parameters tuning, one would wish to further learn the kernel hyper-parameter $\boldsymbol{\gamma}$ by considering a joint objective function $J(\mathbf{W}, \boldsymbol{\gamma})$ and learning $(\mathbf{W}, \boldsymbol{\gamma})$ jointly as a MAP problem w.r.t. the joint posterior distribution $p(\mathbf{W}, \boldsymbol{\gamma} \mid \mathcal{D})$. Fortunately, we show that this again can be solved efficiently using SVGD as presented in Section 3.4. Combining these, we arrive at a robust Bayesian formulation for a multi-class kernel machine together with efficient posterior inference on both the model hyperplanes and kernel parameters developed under SVGD framework.

3.3 SVGD for Posterior Inference

Recall the key ingredient of SVGD (cf. Section 2.2) is to design successive steps to transform a set of particles, originally drawn from any arbitrary simple and tractable distribution, and turn them into samples of a more complicated, intractable distribution. In our case, we transform the task of learning the posterior distribution $p(\mathbf{W} \mid \mathcal{D})$ by finding the mapping from $p(\mathbf{W})$ to $p(\mathbf{W} \mid \mathcal{D})$.

More precisely, we start with S particles $\{\mathbf{W}^1, \ldots, \mathbf{W}^S\}$ drawn from the prior distribution $p(\mathbf{W}) \propto \exp(\Omega(\mathbf{W}))$. In particular, if we choose the regularization term $\Omega(\mathbf{W}) = \frac{\lambda}{2}\|\mathbf{W}\|_{2,2}^2$, the prior distribution $p(\mathbf{W})$ is a multivariate Gaussian distribution with mean

Algorithm 1 Pseudo-code of learning model parameters

```
Input: $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, $\tau$, $\lambda$, $\boldsymbol{\gamma}$
Output: $\bar{\mathbf{W}} = \frac{1}{S}\sum_{s=1}^{S} \mathbf{W}^s$
1: Draw $\mathbf{W}^s \sim p(\mathbf{W})$ for $s = 1, \ldots S$
2: repeat
3:     Draw a mini-batch $\mathcal{B}$ from $\mathcal{D}$
4:     Update $\mathbf{W}^1, \ldots, \mathbf{W}^S$ using Eq. (10)
5: until enough epochs
```

$\mathbf{0}$ and covariance matrix $\lambda^{-1}\mathbf{I}$. We then move these particles to become samples of the posterior $p(\mathbf{W} \mid \mathcal{D})$. This can be realized by moving each $\mathbf{W}^j$, $j = 1, \ldots S$ using (5) to arrive at our update equation as follows:

$$\mathbf{W}_{l+1}^{j} = \mathbf{W}_{l}^{j} + \tau_l \boldsymbol{\Psi}^* \left(\mathbf{W}_l^j\right) \tag{10}$$

where the perturbation direction $\boldsymbol{\Psi}^*$ is derived from Eq. (6):

$$\boldsymbol{\Psi}^*(\mathbf{W}^j) = \frac{1}{S}\sum_{s=1}^{S}\left[\kappa_j\left(\mathbf{W}^s\right)\nabla_{\mathbf{W}}\log p\left(\mathbf{W}^s \mid \mathcal{D}\right) + \nabla_{\mathbf{W}}\kappa_j\left(\mathbf{W}^s\right)\right] \tag{11}$$

and we have also defined

$$\kappa_j\left(\mathbf{W}^s\right) = \kappa(\mathbf{W}^s, \mathbf{W}^j)$$

$$\nabla_{\mathbf{W}}\log p(\mathbf{W} \mid \mathcal{D}) = \nabla_{\mathbf{W}}\log \tilde{p}(\mathbf{W} \mid \mathcal{D}) = \nabla_{\mathbf{W}}(-J(\mathbf{W})) \tag{12}$$

$$= -\nabla_{\mathbf{W}}\Omega(\mathbf{W}) - \mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}\left[\nabla_{\mathbf{W}} l(\mathbf{W}; \mathbf{x}, y)\right] \tag{13}$$

For the kernel function κ, we employ a RBF kernel $\kappa(\mathbf{W}^s, \mathbf{W}^j) = \exp(-1/2\sigma^2 \parallel \mathbf{W}^s - \mathbf{W}^j \parallel_{2,2}^2)$. Thus, the derivative of kernel function w.r.t the first parameter is $\nabla_{\mathbf{W}}\kappa_j(\mathbf{W}) = -1/\sigma^2\left(\mathbf{W} - \mathbf{W}^j\right)\kappa\left(\mathbf{W}, \mathbf{W}^j\right)$.

For the Eq. (13), we can approximate $\mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}\left[\nabla_{\mathbf{W}} l(\mathbf{W}; \mathbf{x}, y)\right]$ by sampling a mini-batch $\mathcal{B}$ from the training set $\mathcal{D}$ as follows:

$$\mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}\left[\nabla_{\mathbf{W}} l(\mathbf{W}; \mathbf{x}, y)\right] \approx \frac{1}{|\mathcal{B}|}\sum_{(\mathbf{x}, y)\in\mathcal{B}} \nabla_{\mathbf{W}} l(\mathbf{W}; \mathbf{x}, y) \tag{14}$$

We note that in the formulation of our BKM there are two different kernels: the random feature kernel $k(\mathbf{x}, \mathbf{x}')$ which approximates the Gaussian kernel and the RBF kernel $\kappa(\mathbf{W}, \mathbf{W}')$ which is used in the SVGD framework for moving the particles $\left\{\mathbf{W}^1, \ldots, \mathbf{W}^S\right\}$.

Asymptotic Behavior of BKM. We conclude this section by investigating the asymptotic behavior of our *BKM* when $S = 1$ or the kernel width σ approaches either 0 or $+\infty$.

• *Analysis for* $s = 1$. Combining Eqs. (11, 13) and note that $\nabla_{\mathbf{W}}\kappa_1\left(\mathbf{W}^1\right) = \mathbf{0}$ and $\kappa_1\left(\mathbf{W}^1\right) = 1$, we then have the formula:

$$\boldsymbol{\Psi}^*(\mathbf{W}^1) = -\nabla_{\mathbf{W}}\Omega(\mathbf{W}^1) - \mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}[\nabla_{\mathbf{W}} l(\mathbf{W}^1; \mathbf{x}, y)] = -\nabla_{\mathbf{W}}\mathcal{J}(\mathbf{W}^1)$$

The update formula in Eq. (10) now becomes:

$$\mathbf{W}_{l+1}^{1} = \mathbf{W}_{l}^{1} - \tau_l \nabla_{\mathbf{W}}\mathcal{J}\left(\mathbf{W}_l^1\right) \tag{15}$$

It is evident that the update formula in Eq. (15) has the form of gradient descent or stochastic gradient descent if we approximate the full gradient using mini-batch as in Eq. (14). Therefore, in this case, the only particle of BKM approaches a local minima of the OP in Eq. (7) or equivalently is the solution of the MAP estimate. For sufficiently large values of S, the particles of BKM are nearly sampled from the true posterior. In the Bayesian inference paradigm, it is more desirable to sample from the true posterior than to find the MAP estimate. Our experiment in Section 4.2.1 agrees this conclusion since the experimental results for $S = 1$ are always less than those for sufficiently large S (e.g., $S = 100$) with a margin of several percents.

• *Analysis for* $\sigma \rightarrow 0$. Since $\kappa_j\left(\mathbf{W}^s\right) = \delta_{js}$ where δ_{js} is the Kronecker delta symbol and $\nabla_{\mathbf{W}}\kappa_j(\mathbf{W}) = \mathbf{0}$ from the equation

$$\lim_{t\rightarrow+\infty}\left(-t\exp(-t)\right) = 0,$$

we then have the formula:

$$\boldsymbol{\Psi}^*(\mathbf{W}^j) = \frac{1}{S}\left[-\nabla_{\mathbf{W}}\Omega(\mathbf{W}^j) - \mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}[\nabla_{\mathbf{W}} l(\mathbf{W}^j; \mathbf{x}, y)]\right]$$
$$= -\frac{1}{S}\nabla_{\mathbf{W}}\mathcal{J}(\mathbf{W}^j)$$

The update formula in Eq. (10) now becomes:

$$\mathbf{W}_{l+1}^{j} = \mathbf{W}_{l}^{j} - \frac{\tau_l}{S}\nabla_{\mathbf{W}}\mathcal{J}\left(\mathbf{W}_l^j\right) \tag{16}$$

It shows that the update formula in Eq. (16) has the form of gradient descent or stochastic gradient descent with a smaller learning rate.

• *Analysis for* $\sigma \rightarrow +\infty$. Since $\kappa_j\left(\mathbf{W}^s\right) = 1$ and $\nabla_{\mathbf{W}}\kappa_j(\mathbf{W}) = \mathbf{0}$, we then have the formula:

$$\boldsymbol{\Psi}^*(\mathbf{W}^j) = \frac{1}{S}\sum_{s=1}^{S}\left[-\nabla_{\mathbf{W}}\Omega(\mathbf{W}^s) - \mathbb{E}_{\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}}[\nabla_{\mathbf{W}} l(\mathbf{W}^s; \mathbf{x}, y)]\right]$$
$$= \frac{1}{S}\sum_{s=1}^{S}\nabla_{\mathbf{W}}\mathcal{J}(\mathbf{W}^s)$$

The update formula in Eq. (10) now becomes:

$$\mathbf{W}_{l+1}^{j} = \mathbf{W}_{l}^{j} - \frac{\tau_l}{S}\sum_{s=1}^{S}\nabla_{\mathbf{W}}\mathcal{J}\left(\mathbf{W}_l^s\right) \tag{17}$$

It is evident that the update formula in Eq. (17) has the style of gradient descent or stochastic gradient descent, but we simultaneously maintain S particles and use the average of gradients at these particles to update. This might help stabilize the solution compared with the standard gradient descent or stochastic gradient descent.

To sum up, this section has presented our approach which uses SVGD to learn the model parameters. This is summarized in Algorithm 1. In this algorithm, we assume the kernel parameters are fixed. In next section, we extend our method to further learn the kernel parameters jointly with model parameters, resulting into a robust inference method for our proposed BKM.

3.4 Learning Kernel Parameter Distribution

To enable learning both $\mathbf{W}$ and $\boldsymbol{\gamma}$, we consider the joint posterior distribution $p\left(\mathbf{W}, \boldsymbol{\gamma} \mid \mathcal{D}\right) = \exp\left(-J\left(\mathbf{W}, \boldsymbol{\gamma}\right)\right)$ as defined in Eq. (9) and then apply SVGD framework to draw the particles from this distribution. In particular, we first draw $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^S$ particles where $\boldsymbol{\theta}^s = \{\mathbf{W}^s, \boldsymbol{\gamma}^s\}$ from the corresponding prior distribution, i.e., $\mathbf{W}^s \sim p(\mathbf{W})$ and $\boldsymbol{\gamma}^s \sim p\left(\boldsymbol{\gamma}\right)$. We employ a Truncated Normal distribution $\mathcal{TN}\left(\mathbf{0}, \mathbf{I}_d, 0, +\infty\right)$ for the prior $p\left(\boldsymbol{\gamma}\right)$. Subsequently, we update these particles using the SVGD framework as follows:

$$\boldsymbol{\theta}_{l+1}^{j} = \boldsymbol{\theta}_{l}^{j} + \tau_l \boldsymbol{\Psi}^*(\boldsymbol{\theta}_l^j) \tag{18}$$

where the perturbation direction $\boldsymbol{\Psi}^*$ is derived as

$$\boldsymbol{\Psi}^*\!\left(\boldsymbol{\theta}^j\right) = \frac{1}{S}\sum_{s=1}^{S}\left[\kappa_j\left(\boldsymbol{\theta}^s\right)\nabla_{\boldsymbol{\theta}}\log p\left(\boldsymbol{\theta}^s \mid \mathcal{D}\right) + \nabla_{\boldsymbol{\theta}}\kappa_j\left(\boldsymbol{\theta}^s\right)\right] \tag{19}$$

and we have further defined

$$\kappa_j\left(\boldsymbol{\theta}^s\right) = \kappa(\boldsymbol{\theta}^s, \boldsymbol{\theta}^j)$$

$$\nabla_{\boldsymbol{\theta}}\log p\left(\boldsymbol{\theta} \mid \mathcal{D}\right) = \left[\nabla_{\mathbf{W}}\log p\left(\mathbf{W}, \boldsymbol{\gamma} \mid \mathcal{D}\right), \nabla_{\boldsymbol{\gamma}}\log p\left(\mathbf{W}, \boldsymbol{\gamma} \mid \mathcal{D}\right)\right]^{\mathsf{T}}$$

In our derivation, we employ a RBF kernel for $\kappa(\boldsymbol{\theta}^s, \boldsymbol{\theta}^j)$ as:

$$\kappa(\boldsymbol{\theta}^s, \boldsymbol{\theta}^j) = \exp\left(-1/2\sigma^2(\|\mathbf{W}^s - \mathbf{W}^j\|_{2,2}^2 + \|\boldsymbol{\gamma}^s - \boldsymbol{\gamma}^i\|_2^2)\right)$$

$$\nabla_{\boldsymbol{\theta}}\kappa_j(\boldsymbol{\theta}) = \left\{\nabla_{\mathbf{W}}\kappa_j(\mathbf{W}), \nabla_{\boldsymbol{\gamma}}\kappa_j(\boldsymbol{\gamma})\right\}$$

$$\nabla_{\mathbf{W}}\kappa_j(\mathbf{W}) = -1/\sigma^2(\mathbf{W} - \mathbf{W}^j)\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}^j)$$

$$\nabla_{\boldsymbol{\gamma}}\kappa_j(\boldsymbol{\gamma}) = -1/\sigma^2(\boldsymbol{\gamma} - \boldsymbol{\gamma}^j)\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}^j)$$

To compute the derivatives of $\log p(\mathbf{W}, \boldsymbol{\gamma} \mid \mathcal{D})$ w.r.t $\mathbf{W}$ and $\boldsymbol{\gamma}$ respectively, we also sample a mini-batch $\mathcal{B}$ like in Section 3.3. In summary, we present the pseudo-code of our proposed method in Algorithm 2.

Algorithm 2 reveals that we first draw the particles $\boldsymbol{\theta}^s = \{\mathbf{W}^s, \boldsymbol{\gamma}^s\}$, $s = 1, \ldots, S$ from the prior $p(\mathbf{W}, \boldsymbol{\gamma}) = p(\mathbf{W})\, p(\boldsymbol{\gamma})$. We then update these particles using Eq. (18). When the number of iterations gradually increases, these particles approach the true samples of the joint posterior distribution $p(\mathbf{W}, \boldsymbol{\gamma} \mid \mathcal{D})$. Finally, we output the average the particles for the final solution.

Considering Eq. (18) and (19), in each loop, we have to compute $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}^s \mid \mathcal{D})$ for all particle as well as calculate $\kappa_j(\boldsymbol{\theta}^s)$ for all $s = 1, \ldots S$ and $j = 1, \ldots S$. It cost $O(|\mathcal{B}|\bar{D})$ to compute $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}^s \mid \mathcal{D})$ and $O(\bar{D}^2)$ to compute $\kappa_j(\boldsymbol{\theta}^s)$ where $\bar{D} = d + D$ and $|\mathcal{B}|$ is the batch size. Thus, the complexity of our method is $O(lS(\bar{D}^2 + |\mathcal{B}|\bar{D}))$ where l is the number of epochs.

4 EXPERIMENTS

In this section, we conduct comprehensive experiments to quantitatively evaluate the capacity and scalability of our proposed BKM on classification problem compared with other baselines. Our code is available at https://github.com/dascimal-org/bkm

4.1 Experimental setup

In order to have the fairest comparison, we use 12 datasets with diverse ranges of size, dimension and domain. These datasets are available on the LIBSVM repository[1] and their statistics are shown in Table 1. All experiments are performed on the computer with Xeon E5 2.6 GHz CPU and 96 GB RAM. We repeat each experiment 5 times and record the corresponding mean value and standard deviation. We use the cross-entropy loss with softmax to take advantage its smoothness and ease in computing its gradients, although our BKM is readily generalizable to other loss functions.

4.2 Model behavior evaluation

4.2.1 Influence of the number of particles S . In the first experiment, we observe the model behavior when varying the number of particles S in the range of $\{1, 2, 3, 10, 20, 30, 100, 200, 300\}$. Predictive performance and training time on *a9a* and *seismic* datasets are shown in Fig. 1. In both datasets, the classification accuracy increases and then remains steady when S increases. The reason is that the number of particles affects the accuracy with which the expectation is approximated in Eq. (4), which requires an adequate number of particles to describe the joint posterior distribution of model and kernel parameters. However, increasing S slows down the training, hence, for all later experiments, we fix $S = 100$ for all datasets which can achieve accurate classifications with reasonable training times.

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Algorithm 2 Pseudo code of *Bayesian Kernel Machine*

```
Input: $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, $\tau$, $\lambda$
Output: $\bar{\boldsymbol{\theta}} = \frac{1}{S}\sum_{s=1}^S \boldsymbol{\theta}^s$ where $\boldsymbol{\theta}^s = \{\mathbf{W}^s, \boldsymbol{\gamma}^s\}$
1: Draw $\mathbf{W}^s \sim p(\mathbf{W})$ and $\boldsymbol{\gamma}^s \sim p(\boldsymbol{\gamma})$ for $s = 1, \ldots S$
2: repeat
3:     Draw a mini-batch $\mathcal{B}$ from $\mathcal{D}$
4:     Update $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^S$ using Eq. (18)
5: until enough epochs
```

Dataset	#train	#test	#dim	#classes
poker	25,010	1,000,000	10	10
a9a	32,561	16,281	123	2
w8a	49,749	14,951	300	2
cod-rna	47,628	11,907	8	2
ijcnn1	49,990	91,701	22	2
seismic	78,823	19,705	50	3
connect-4	54,045	13,512	126	3
skin	196,046	49,011	3	2
epsilon	400,000	100,000	2,000	2
covtype	464,810	116,202	54	2
susy	4,000,000	1,000,000	18	2
higgs	8,800,000	2,200,000	28	2

Table 1: Experimental datasets and their statistics

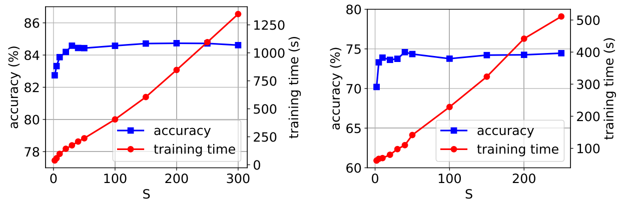

Figure 1: Model behavior on the *a9a* (left) and *seismic* (right) datasets when varying the number particles S

4.2.2 Influence of mini-batch size. Next, we investigate how the size of mini-batch $|\mathcal{B}|$ in Eq. (14) affects the predictive performance of our model. We vary the batch size in the range of $\{2, 3, 10, 20, 30, 100, 200, 300\}$ and measure classification accuracy and training time. Fig 2 visualizes our experimental results on the *a9a* and *seismic* datasets. It shows that when increasing the batch size, the classification accuracy is improved steadily and then does slightly change. The reason lies on the approximation of the expectation in Eq. (14). To estimate that expectation, we take average of gradients from a mini-batch, and hence a too small batch size certainly compromises the predictive performance, but a large batch size slows down the training. However, we note that the training time grows much slower compared to the batch size. In particular, the batch size increases 150 times from 2 to 300, but the training time just grows around 3 times from 180 seconds to 580 seconds on the *a9a* dataset and from 100 seconds to 340 seconds on the *seismic* dataset. The reason is that, with larger batch size, we can take advantage of modern CPUs to parallelize computations. In later experiments, we set the batch size to 200 for all datasets.

4.2.3 Influence of kernel width σ. Next, we further analyze the effect of kernel width σ to the performance. We vary σ and measure the classification accuracy as well as the training time. We

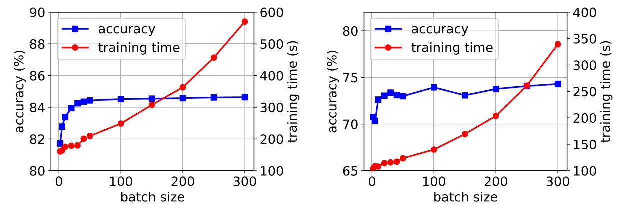

Figure 2: Model behavior on *a9a* (left) and *seismic* (right) datasets when varying batch size $|\mathcal{B}|$

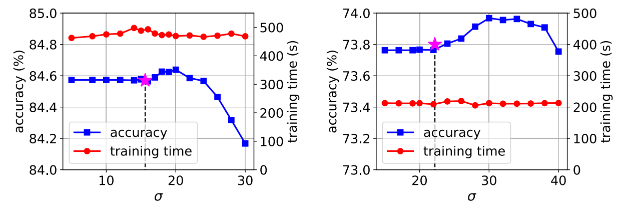

Figure 3: Model behavior on *a9a* (left) and *seismic* (right) datasets when varying kernel width σ. The *star* marker represents the accuracy result using the heuristic approach.

present our analytical results on the *a9a* and *seismic* datasets in Fig 3. Obviously, the kernel width σ does not affect the training times whose plots show the straight lines in the figures. When increasing σ, the accuracy climbs up gradually and achieves the highest value at $\sigma = 20$ for *a9a* and $\sigma = 30$ for *seismic*, then declines steadily. This shows that the parameter σ is *insensitive* to the predictive performance of our *BKM*. To find an appropriate kernel width σ, we employ a heuristic approach used in the SVGD's implementation[2] of the corresponding authors. In short, we compute the median of pairwise Euclidean distances between any two particles. Subsequently, the kernel width σ is computed as $\sigma = \exp(-0.5 \times median/\log S)$. In Fig 3, the *star* marker denotes the result using this approach. The horizontal axis of that marker is the computed value of σ and its vertical axis presents the classification accuracy. Comparing to the highest point of the blue line, the *star* marker is just slightly lower. Thus, we are keen on this approach in the later experiments and observe that this approach for computing σ entails satisfactory classification performances.

4.2.4 Influence of regularization parameter λ. We can integrate the regularization parameter λ to the learnable variable $\boldsymbol{\theta}$ and infer the joint distribution over $(\mathbf{W}, \theta, \lambda)$. However, this requires us to infer from a higher dimensional distribution which makes the convergence slower. In addition, we observe that λ is *insensitive* to the predictive performance of *BKM*. Thus, for other experiments, we set $\lambda = 10^{-3}$ for all datasets. In what follows, we present our empirical observation in the sensitiveness of the regularization parameter λ. We illustrate our results on the *a9a* and *seismic* datasets in Fig 4 when varying λ in the range of $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. In both figures, there is a consistent pattern: the accuracy classification and the training time are stable. It shows that our *BKM* is robustly insensitive to regularization parameter λ.

4.2.5 Comparison with learning from multiple models. In this experiment, we want to prove that a naive and simple strategy for ensembling of multiple models cannot improve the predictive performance, hence aggregating multiple models which can boost

[2]https://github.com/DartML/Stein-Variational-Gradient-Descent

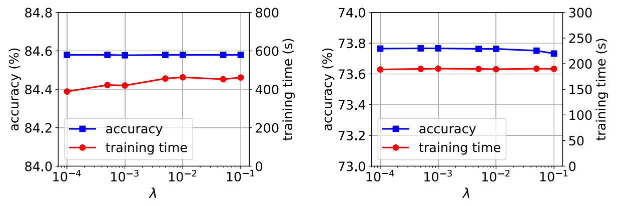

Figure 4: Model behavior on *a9a* (left) and *seismic* (right) datasets when varying regular parameter λ

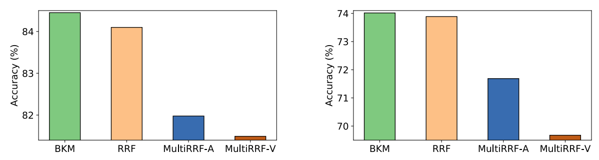

Figure 5: Comparison with learning from multiple kernels on *a9a* (left) and *seismic* (right) datasets

the predictive performance as in *BKM* is not a trivial task. A naive and straightforward approach is to learn multiple models using different hyperparameters and then ensemble these resulting models. We empirically show that this strategy actually degrades performance relative to a single model approach. In particular, we employ the RRF method and learn multiple RRF models departing from different initial values of kernel width and use two strategies to construct multiple RRF models: averaging strategy (MultiRRF-A) and voting strategy (MultiRRF-V). Our experimental results, illustrated in Fig. 5, clearly show that both averaging and voting achieve lower accuracies as compared with the single RRF model and our *BKM*. It appears that constructing multiple models in such the way does not improve predictive performance. In our *BKM*, particles share information together via the kernel function $\kappa(\boldsymbol{\theta}, \boldsymbol{\theta}')$, thus they improve each other, hence gradually yielding a stronger classifier.

4.3 Classification performance comparison

4.3.1 Baselines. We employ 3 popular baselines which require grid search and 2 state-of-the-art methods which have the capability to learn kernel parameters as follows:

- LIBSVM [3]: the most popular kernelized SVM. We use the latest version that supports multitasking on multi-core CPUs.
- BSGD: budget stochastic gradient descent algorithm with merging strategy [24] that can break the curse of kernelization to efficiently handle large-scale datasets. We use the merging strategy because it is the best strategy in terms of predictive performance as the authors reported.
- RKS: random kitchen sink algorithm [21] which uses random features to approximate a shift-invariant and p.s.d kernel function for addressing the curse of kernelization. Consequently, it can alleviate the time-consuming drawback in tuning hyperparameters.
- FKL: Fourier kernel learning [1] which applies reparameterization trick to make kernel parameters learnable. Consequently, FKL can learn model and kernel parameters.
- RRF: a kernel learning method [18] which applies another transformation and source of randomness in reparameterization trick that allows kernels to be approximated more tightly than FKL.

Dataset	**poker**		**a9a**		**w8a**	
Algorithm	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)
BKM	**0.40**	50.52±0.71	**0.42**	84.45±0.18	0.93	99.03±0.01
LIBSVM	NA	NA	(1.42+0.02) 1.44	**84.92**	(0.86+0.01) **0.87**	**99.06**
RKS	(20.02+0.01) 20.03	49.10±1.30	(20.57+0.01) 20.58	84.26±0.16	(23.13+0.01) 23.14	96.96±0.01
BSGD	(28.01+0.12) 28.13	**54.21±0.33***	(33.64+0.12) 33.76	84.17±0.04	(51.98+0.09) 52.07	98.27±0.11
FKL	(2.07+0.02) 2.09	33.87±0.66	(2.34+0.02) 2.36	84.26±0.16	(2.20+0.02) 2.22	96.96±0.01
RRF	(3.50+0.01) 3.51	37.46±0.31	(3.38+0.03) 3.41	83.99±0.03	(3.09+0.03) 3.12	96.96±0.01
Dataset	**cod-rna**		**ijcnn1**		**seismic**	
Algorithm	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)
BKM	**1.05**	96.13±0.05	1.14	**97.42±0.14**	2.06	**73.77±0.47**
LIBSVM	(0.03+1.93) 1.96	**96.39**	(0.65+0.01) **0.66**	97.35	NA	NA
RKS	(20.56+0.01) 20.57	93.46±0.03	(20.08+0.01) 20.09	93.23±0.11	(39.01+0.02) 39.03	66.69±0.49
BSGD	(50.15+0.08) 50.23	95.76±0.06	(21.77+0.04) 21.81	97.22±0.11	(271.52+0.24) 271.76	73.14±0.09
FKL	(1.79+0.02) 1.81	83.54±0.36	(2.24+0.02) 2.26	95.40±0.08	(2.15+0.04) 2.19	71.98±0.06
RRF	(2.04+0.02) 2.06	95.50±0.05	(1.92+0.02) 1.94	86.12±0.46	(1.69+0.07) **1.76**	72.88±0.11
Dataset	**connect-4**		**skin**		**epsilon**	
Algorithm	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)
BKM	**2.61**	**78.87±0.16***	**3.20**	**99.91**	**4.18**	**88.98±0.46***
LIBSVM	NA	NA	(6.69+13.53) 20.22	**99.91**	NA	NA
RKS	(81.83+0.04) 81.87	76.14±0.11	(9.23+0.04) 9.27	97.63±0.15	(15.69+0.04) 15.73	63.29±0.01
BSGD	NA	NA	(8.40+0.16) 8.56	99.87±0.01	NA	NA
FKL	(8.71+0.09) 8.80	73.61±0.11	(10.02+0.05) 10.07	99.90±0.01	(33.74+0.10) 33.84	70.03±0.91
RRF	(6.57+0.07) 6.64	76.85±0.14	(11.18+0.06) 11.24	97.63±0.15	(25.07+0.08) 25.15	83.82±0.45
Dataset	**covtype**		**susy**		**higgs**	
Algorithm	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)	Time (h)	Accuracy (%)
BKM	**4.09**	**70.06±1.31***	**13.63**	**79.97±0.09**	**19.70**	**70.86±0.63***
LIBSVM	NA	NA	NA	NA	NA	NA
RKS	(16.07+0.04) 16.11	67.28±0.06	(14.11+0.14) 14.28	79.56±0.01	(27.72+0.16) 27.88	53.31±0.01
BSGD	(82.37+0.89) 83.26	66.36±0.63	(55.71+12.68) 68.39	78.25±0.02	(188.50+17.71) 206.21	62.89±0.07
FKL	(51.71+0.07) 52.28	55.12±1.57	(30.06+0.21) 30.27	79.68±0.02	(31.07+0.25) 31.32	53.03±0.01
RRF	(46.67+0.06) 47.07	63.32±0.69	(21.27+0.17) 21.44	79.75±0.03	(48.09+0.30) 48.39	53.02±0.01

Table 2: Performance comparison on real datasets ordered from smallest to largest. We report the accuracy (in %) and the total running time (in hours). For methods requiring a tuning hyperparameters procedure, the total running time includes the tuning hyperparameters (the left value in the parenthesis) and the training time (the right value in the parenthesis). For all experiments exceeds the limit of 50 days, we denote NA. The best value per dataset is emphasized in boldface and the runner-up is underlined. The best methods denoted with a asterisks are statistically different from the runner-up using a paired t-test with p-value < 0.05

4.3.2 Experimental setting. We use grid search with 5-folds cross-validation for methods requiring tuning hyperparameters, i.e., LIBSVM, BSGD, and RKS. For unscalable methods, i.e., LIBSVM and BSGD, we do grid search on a subset of the training set. In particular, we randomly extract 10% for datasets with over ten thousand data instances (i.e., *poker*, *a9a*, *w8a*, *cod-rna*, *ijcnn1*, *seismic*, and *acoustic*), 1% for datasets with over hundred thousand data instances (i.e., *skin*, *epsilon*, and *covtype*), and 0.1% for datasets with over million data instances (i.e., *susy* and *higgs*). For RKS, we grid search on the whole datasets, except *skin*, *epsilon*, *covtype*, *susy* and *higgs* datasets which are searched in the subset 10%, 10%, 10%, 1%, 1% respectively. The searching range for the kernel width is $\{2^{-5}, 2^{-3},..., 2^{15}\}$, and the trade-off parameter C in LIBSVM is searched in the range of $\{2^{-5}, 2^{-3},..., 2^{15}\}$ as suggested in [12]. For the regularization parameter λ in BSGD and RKS, we vary in the range of $\{2^{-15}, 2^{-13},..., 2^{5}\}$. The range for learning rate in RKS is $\{10^{-7}, 10^{-5}, 10^{-3}, 10^{-1}\}$. We use the merging strategy for budget maintenance in BSGD and set the budget size to 200 for all experiments.

For RRF and FKL, the searching range of the learning rate is the same as RKS. Although they can learn kernel parameter $\boldsymbol{\gamma}$, their prediction performance are sensitive to the initial value $\boldsymbol{\gamma}_0$ of the kernel parameter. Thus, we also do grid search to find the optimal $\boldsymbol{\gamma}_0$. The searching range of $\boldsymbol{\gamma}_0$ is $\{2^{-15}, 2^{-11},..., 2^{5}\}$. We search on the whole datasets, except *skin*, *epsilon*, *covtype*, *susy* and *higgs* datasets, like in RKS. In our method, we just randomly initialized the kernel width $\boldsymbol{\gamma}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. In RRF, FKL and our method, we set the regular parameter $\lambda = 10^{-3}$ for all datasets. For decaying step size τ_l, we use the Adagrad update style as mentioned in Section 2.2 where we set $\varepsilon = 10^{-3}$ and $\delta = 10^{-6}$.

4.3.3 Experimental results and discussion. We report the total running time and classification results in Table 2. For methods requiring tuning hyperparameters, i.e. LIBSVM, BSGD, and RKS, the grid search time is reported as the first number in the parenthesis. The second number in the parenthesis is the training time w.r.t the optimal hyperparameters on the whole dataset.

For LIBSVM, we do not report the standard deviation because it is a deterministic algorithm. Due to the curse of kernelization, the running time of LIBSVM grows quadratically when the training size increases. Thus, while LIBSVM has comparable times and accuracy to BKM on most of the smallest datasets, although LIBSVM can finish doing grid search in the subset of a large-scale training set, it cannot finish training the whole dataset for only one pair of hyperparameters in 50 days. Moreover, the stopping criteria of LIBSVM are not flexible enough. In some cases, e.g. *poker* dataset, it does not converge, thus the running time exceeds our time limit. For those cases, we denote as NA.

To break the curse of kernelization, BSGD uses a budget strategy to keep the model size fixed, hence, unlike LIBSVM, it can finish the training phrase on large-scale datasets. However, it still requires a hyperparameters tuning procedure. Moreover, the corresponding authors' implementation does not support multi-core CPUs. Thus, it runs slower than the up-to-date version of LIBSVM on some medium-scale datasets and other baselines on all datasets. In comparison of predictive performance, BSGD gains better accuracy than other baselines but still lower than ours and LIBSVM.

The RKS method uses another approach to avoid the curse of kernelization by using a random feature space. The training time per setting of hyperparameters is much faster than other baselines. However, it still requires tuning hyperparameters. In addition, because of the random nature of the algorithm, we need to repeat 5 times per setting of hyperparameters to stabilize the searching results. Moreover, it seems incapable of finding the optimal hyperparameters via doing grid search in a subset of the training set, especially when the subset is significantly small in comparison of the whole training set. In particular, the accuracy scores on *higgs* are significantly lower than other baselines.

RRF and FKL methods can learn kernel parameters, but it still requires to tune initial value for $\boldsymbol{\gamma}$ and learning rate. In some datasets including *cod-rna*, *ijcnn1*, *seismic*, *connect-4, skin, epsilon,* and *susy*, they can improve the predictive performance in comparison with RKS. However, their accuracy are still lower than ours and LIBSVM. Their training times without including grid search times are longer than RKS. That is because the optimization problems in RRF and FKL are more complicated which involve calculating the gradient of the objective function w.r.t to $\boldsymbol{\gamma}$.

In BKM, the training time is slower than RKS, FKL and RRF for a single parameter set. However, if we count the grid search time, the total running time of BKM is considerably smaller than other baselines. Our model can achieve comparable predictive performances with LIBSVM while significantly outperforming other baselines in all datasets.

5 CONCLUSION

This paper has proposed a novel Bayesian Kernel Machine model with combined strength of Bayesian inference, kernel methods, random feature representation and recent advances in Stein variational inference. Our proposed BKM is resilient against curse of kernelization, robust against parameter tuning and can scale seamlessly for big datasets. Our extensive experiments have demonstrated the merit of our proposed approach against state-of-the-art baselines. Several future works are also promising to extend our work for sparse as well as multiple hyperplanes cases.

REFERENCES

[1] Eduard Gabriel Băzăvan, Fuxin Li, and Cristian Sminchisescu. 2012. Fourier kernel learning. In *Computer Vision–ECCV 2012*. Springer, 459–473.

[2] Yoshua Bengio, Olivier Delalleau, and Nicolas L Roux. 2006. The curse of highly variable functions for local kernel machines. In *Advances in neural information processing systems*. 107–114.

[3] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2 (2011), 27:1–27:27. Issue 3.

[4] Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. 2002. Choosing multiple parameters for support vector machines. *Machine learning* 46, 1 (2002), 131–159.

[5] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. 2010. Two-Stage Learning Kernel Algorithms.. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. 239–246.

[6] C. Cortes and V. Vapnik. 1995. Support-Vector Networks. In *Machine Learning*. 273–297.

[7] K. Crammer and Y. Singer. 2002. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research* 2 (2002), 265–292.

[8] Nello Cristianini, Andre Elisseeff, John Shawe-Taylor, and Jaz Kandola. 2001. On kernel-target alignment. (2001).

[9] Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2006. The Forgetron: A kernel-based perceptron on a fixed budget. In *Advances in neural information processing systems*. 259–266.

[10] Mehmet Gönen and Ethem Alpaydın. 2011. Multiple kernel learning algorithms. *The Journal of Machine Learning Research* 12 (2011), 2211–2268.

[11] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. 2008. Kernel methods in machine learning. *The annals of statistics* (2008), 1171–1220.

[12] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. 2003. A practical guide to support vector classification. (2003).

[13] S Sathiya Keerthi, Vikas Sindhwani, and Olivier Chapelle. 2007. An efficient method for gradient-based adaptation of hyperparameters in SVM models. In *Advances in neural information processing systems*. 673–680.

[14] Trung Le, Tu Nguyen, Vu Nguyen, and Dinh Phung. 2016. Dual Space Gradient Descent for Online Learning. In *Advances in Neural Information Processing Systems 29*. 4583–4591.

[15] Qiang Liu and Dilin Wang. 2016. Stein variational gradient descent: A general purpose Bayesian inference algorithm. In *Advances In Neural Information Processing Systems*. 2378–2386.

[16] Jing Lu, Steven CH Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. 2016. Large scale online kernel learning. *Journal of Machine Learning Research* 17, 47 (2016).

[17] Khanh Nguyen, Trung Le, Vu Nguyen, Tu Nguyen, and Dinh Phung. 2016. Multiple Kernel Learning with Data Augmentation. In *Asian Conference on Machine Learning*. 49–64.

[18] Tu Dinh Nguyen, Trung Le, Hung Bui, and Dinh Phung. 2017. Large-scale online kernel learning with random feature reparameterization. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17)*. 2543–2549.

[19] Francesco Orabona, Joseph Keshet, and Barbara Caputo. 2009. Bounded kernel-based online learning. *Journal of Machine Learning Research* 10, Nov (2009), 2643–2666.

[20] Nicholas G. Polson and Steven L. Scott. 2011. Data augmentation for support vector machines. *Bayesian Anal.* 6, 1 (03 2011), 1–23. https://doi.org/10.1214/11-BA601

[21] Ali Rahimi and Benjamin Recht. 2008. Random features for large-scale kernel machines. In *Advances in neural information processing systems*. 1177–1184.

[22] Ingo Steinwart. 2003. Sparseness of support vector machines. *Journal of Machine Learning Research* 4, Nov (2003), 1071–1105.

[23] Michael E Tipping. 2001. Sparse Bayesian learning and the relevance vector machine. *Journal of machine learning research* 1, Jun (2001), 211–244.

[24] Z. Wang, K. Crammer, and S. Vucetic. 2012. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale SVM training. *J. Mach. Learn. Res.* 13, 1 (2012), 3103–3131.

[25] Zhuang Wang and Slobodan Vucetic. 2009. Twin vector machines for online learning on a budget. In *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM, 906–917.

[26] Zhihua Zhang, Guang Dai, and Michael I Jordan. 2011. Bayesian generalized kernel mixed models. *The Journal of Machine Learning Research* 12 (2011), 111–139.

[27] J. Zhu, N. Chen, and E. P. Xing. 2011. Infinite SVM: a Dirichlet process mixture of large-margin kernel machines. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 617–624.