Data Mining and Knowledge Discovery (2021) 35:2282–2312
https://doi.org/10.1007/s10618-021-00785-1

Isolation kernel: the X factor in efficient and effective large scale online kernel learning

Kai Ming Ting[1] · **Jonathan R. Wells**[2] · **Takashi Washio**[3]

Received: 3 November 2020 / Accepted: 21 July 2021 / Published online: 19 August 2021

Abstract

Large scale online kernel learning aims to build an efficient and scalable kernel-based predictive model incrementally from a sequence of potentially infinite data points. Current state-of-the-art large scale online kernel learning focuses on improving efficiency. Two key approaches to gain efficiency through approximation are (1) limiting the number of support vectors, and (2) using an approximate feature map. They often employ a kernel with a feature map with intractable dimensionality. While these approaches can deal with large scale datasets efficiently, this outcome is achieved by compromising predictive accuracy because of the approximation. We offer an alternative approach that puts the kernel used at the heart of the approach. It focuses on creating a *sparse and finite-dimensional feature map* of a kernel called Isolation Kernel. Using this new approach, to achieve the above aim of large scale online kernel learning becomes extremely simple—simply use Isolation Kernel instead of a kernel having a feature map with intractable dimensionality. We show that, using Isolation Kernel, large scale online kernel learning can be achieved efficiently without sacrificing accuracy.

Keywords Data dependent kernel · Online kernel learning · Kernel functional approximation · Large scale data mining

Responsible editor: Jingrui He

✉ Kai Ming Ting
tingkm@nju.edu.cn

Jonathan R. Wells
jonathan.wells.research@gmail.com

Takashi Washio
washio@ar.sanken.osaka-u.ac.jp

1 National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

2 School of Information Technology, Deakin University, Geelong, Australia

3 The Institute of Scientific and Industrial Research, Osaka University, Osaka, Japan

1 Introduction

In the age of big data, the ability to deal with large datasets or online data with potentially infinite data points is a key requirement of machine learning methods. Kernel methods are an elegant machine learning method to learn a nonlinear boundary from data. However, its applications in the age of big data is limited because of its perennial problem of high computational cost on high dimensional and large datasets.

Current state-of-the-art large scale online kernel learning focuses on improving efficiency. There are two key approaches (e.g., Wang and Vucetic (2010); Wang et al. (2012); Cavallanti et al. (2007); Lu et al. (2016)) to gain efficiency through approximation, i.e., (1) limiting the number of support vectors, and (2) using an approximate feature map. These approaches often employ a kernel which has a feature map with intractable dimensionality. While successfully achieving the efficiency gain, both approaches must also manage the inevitably negative impact of the approximation on accuracy as much as possible.

Here we offer a third approach that puts the kernel used at the heart of the approach. The idea is to create and use a *sparse and finite-dimensional* feature map of a kernel called Isolation Kernel (Ting et al. 2018; Qin et al. 2019). We show that a feature map with these characteristics yields an online kernel learning method which achieves efficiency gain without the need to manage accuracy degradation.

For example, the predictive accuracy of a current method can be degraded to an unacceptable low level when it is applied to datasets having more than 1000 dimensions; while the new method can maintain high accuracy with equivalent or better efficiency gain. The details are provided in Section 8.

The contributions of this paper are:

1. Offering a new approach to online kernel learning which does not employ a kernel with closed form expression that has a feature map with intractable dimensionality.
2. Revealing that a recent Isolation Kernel has no close form expression but a sparse and finite-dimensional feature map.
3. Showing that Isolation Kernel with its sparse and finite-dimensional feature map is a crucial factor in enabling efficient large scale online kernel learning without compromising accuracy. Specifically, the proposed feature map enables three key elements: learning with the feature map, efficient dot product and GPU acceleration that lead to the success of the proposed method.
4. Demonstrating the impact of Isolation Kernel on an existing algorithm of online kernel learning called Online Gradient Descent (OGD) and also support vector machines (SVM). Using Isolation Kernel, instead of a kernel having a feature map with intractable dimensionality, the same algorithms (OGD and SVM) often achieve better predictive accuracy and always have significantly faster runtime by up to three orders of magnitude.
5. Unveiling for the first time that (a) the Voronoi-diagram based implementation of Isolation Kernel produces better predictive accuracy than the tree based implementation in kernel methods using OGD; (b) the GPU version of the implementation is up to four orders of magnitude faster than the CPU version.

Note that the work reported in Lu et al. (2016) and Wang et al. (2012), including OGD and NOGD, and our proposed method do not address the concept change issue in online setting. Nevertheless, all these works address the efficiency issue in online setting which serves as the foundation to tackling the efficacy issue of large scale online kernel learning under concept change.

The rest of the paper is organised as follows. Section 2 describes the current challenges and key approach in large scale online kernel learning. Section 3 presents the two previously unknown advantages of Isolation Kernel and its current known advantage. Section 4 describes the current understanding of Isolation Kernel: its definition, implementations and characteristics. Section 5 presents our four conceptual contributions in relation to learning with the feature map of Isolation Kernel. Its applications to online gradient descent and support vector machines are presented in Section. 6. The experimental settings and results are provided in the next two sections. Section 9 describes the relationship with existing approaches to efficient kernel methods, followed by discussion and concluding remarks in the last three sections.

2 Current challenges and key approach in large scale online kernel learning

We will describe the current challenges in online kernel learning and an influential approach to meet one of the challenges in the next two subsections

2.1 Challenges in online kernel learning

Kernel methods are an elegant way to learn a nonlinear boundary. But they are hampered by high computational cost. There are two approaches in improving its efficiency, depending on whether one is solving the dual or primal optimisation problem.

In solving the dual optimisation problem that employs the kernel trick to avoid feature mapping, one of its main computational costs is due to the prediction function used, i.e., ${}_{dual}f(\mathbf{x}) = \sum_{i=1}^{s} \alpha_i c_i K(\mathbf{x}_i, \mathbf{x})$, where K is the chosen kernel function; α_i is the learned weight and $c_i \in \{+, -\}$ is the class label of support vector $\mathbf{x}_i$; and s is the number of support vectors. The sign of ${}_{dual}f(\mathbf{x})$, i.e., $+$ or $-$, yields the final class prediction.

The evaluation of the prediction function ${}_{dual}f$ has high cost if the number of support vectors is high.

The first approach to improve efficiency is to limit the number of support vectors, and it is often called budget online kernel learning (e.g., Cavallanti et al. (2007); Wang and Vucetic (2010); Wang et al. (2012); and see Lu et al. (2016) for a review.) The key limitation of this approach is that it is unable to deal with an unlimited number of support vectors.

Abandoning the kernel trick by using an approximate feature map of a chosen nonlinear kernel, one usually solves the primal optimisation problem because its prediction function has less cost. The evaluation of the prediction function ${}_{primal}f$ has cost proportional to the number of features in the feature map Φ, i.e., ${}_{primal}f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$,

Table 1 Feature map size comparison for three kernels. $C(\cdot, \cdot)$ is a binomial coefficient. See Chang et al. (2010) for details about polynomial kernel. $\mathbf{x}$ and $\mathbf{y}$ are data points of d dimensions; D is the given dataset. All other variables are scalar parameters

Kernel		Feature map (#dimensions)
Gaussian	$exp(-\gamma \parallel \mathbf{x} - \mathbf{y} \parallel^2)$	infinite
Polynomial	$(\alpha \mathbf{x} \cdot \mathbf{y} + r)^h$	$C(d+h, h)$
Isolation	space-partitioning$(t, \psi \mid D)$	$t\psi \rightarrow t$

where $\mathbf{w} = \sum_{i=1}^{s} \alpha_i c_i \Phi(\mathbf{x}_i)$ can be pre-computed once the support vectors are determined.

The success of the second approach relies on a method to produce a good approximate feature map. This approximation can be costly; and some method can only afford to use a data subsample for the approximation in order to reduce the time complexity. This requirement has the same impact of degrading the accuracy as limiting the number of support vectors in ${}_{dual}f$ used in the first approach.

Kernel methods, that are aimed for large scale datasets, solve the primal optimisation problem because ${}_{primal}f$ has constant time cost, independent of the number of support vectors, as used in a recent system (Lu et al. 2016).

In a nutshell, the key challenge in large scale online kernel learning that employs ${}_{primal}f$ is to **obtain a good approximate feature map of a chosen nonlinear kernel function** such that the inevitable negative impact they have on accuracy is reduced as much as possible.

2.2 An existing influential approach

The need to approximate a feature map of a chosen nonlinear kernel arises because existing nonlinear kernels such as Gaussian and polynomial kernels have either infinite or a large number of features. Table 1 provides the sizes of their feature maps.

One influential approach to meet this challenge is kernel functional approximation; and its two popular methods are: (a) The Nyström embedding method (Williams and Seeger 2001) which uses sample points from the given dataset to construct a matrix of low rank r and derive a vector representation of data of r proxy features. (b) Derive random features based on Fourier transform (Rahimi and Recht 2007; Felix et al. 2016) or Laplacian transform (Yang et al. 2014), independent of the given dataset. Both produce an approximate feature map of a chosen nonlinear kernel using proxy features which are aimed to be used as input to a linear learning algorithm.

A recent proposal of online kernel learning (Lu et al. 2016) has employed the Nyström embedding method. The algorithm called NOGD (Nyström Online Gradient Descend) has shown encouraging results, dealing successfully with large scale datasets and has good predictive accuracy in online setting for datasets less than 800 dimensions (Lu et al. 2016).

However, because the feature map is an approximation, the approach reduces the time and space complexities with the expense of accuracy. In addition, we demonstrate

that NOGD has performed poorly on datasets more than 1000 dimensions (see the results in Sect. 8).

We show here that the challenge on online kernel learning only exists because of the kind of kernels employed. For existing commonly used nonlinear kernels, the dimensionality of their feature maps is not controllable by a user, and has infinite or a large number of dimensions. The kernel functional approximation approach is a workaround without addressing the root cause of the challenge.

In summary, the two current approaches to large scale online learning often employ a kernel that has a feature map with intractable dimensionality. We show in the next section that using a kernel with a different characteristic gives rise to a new approach.

3 Advantages of Isolation kernel

We show here that a recent kernel called Isolation Kernel (Ting et al. 2018; Qin et al. 2019) has two advantages, unbeknown previously, compared with existing data independent kernels:

i) The unique characteristic is that Isolation Kernel has a feature map which is *sparse* and has a finite number of features that can be controlled by a user.
ii) The sparse and finite-dimensional representation i.e., each feature vector has exactly t out of the $t\psi$ elements being non-zero, enables an efficient dot product implementation.

Isolation Kernel has no closed form expression, but a feature map which is obtained from isolation partitions created from the given dataset.

The first advantage eliminates the need to get an approximate feature map (through kernel functional approximation or other means)—when an exact feature map is available, there is no reason to use an approximate feature map. The existence of such a feature map destroys the premise of the challenge in online kernel learning.

The unique characteristic of Isolation Kernel enables kernel learning to solve the primal optimisation problem efficiently. This is because evaluating the prediction function can be conducted more efficiently using ${}_{primal}f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$, where $\mathbf{w} = \sum_{i=1}^{s} \alpha_i c_i \Phi(\mathbf{x}_i)$ can be pre-computed once the support vectors are determined. This is applicable in the testing stage as well as in the training stage.

The second advantage of sparse and finite-dimensional representation enables the dot product in ${}_{primal}f$ to be computed efficiently, i.e., orders of magnitude faster than that without the efficient implementation under some condition.

We show that the above advantages of Isolation Kernel allow an efficient kernel-based prediction model to deal with an unlimited number of support vectors in a sequence of infinite data points.

In a nutshell, the type of kernel used, which has infinite or large number of features, has necessitated an intervention step to approximate its feature map. A considerable amount of research effort (Chang et al. 2010; Williams and Seeger 2001; Rahimi and Recht 2007; Yang et al. 2014) has been invested in order to produce a feature map that has a more manageable dimensionality. Using the type of kernel such as Isolation

Kernel—which has an user-controllable finite-dimensional feature map—eliminates the need of such an intervention step for feature map approximation.

3.1 One known advantage of Isolation Kernel

In addition to the above two (previously unknown) advantages, Isolation Kernel has one known advantage, i.e., it is data dependent (Ting et al. 2018; Qin et al. 2019), as opposed to *data independent* kernels such as Gaussian and Laplacian kernels. It is solely dependent on data distribution, requiring neither class information nor explicit learning. Isolation Kernel has been shown to be a better kernel than existing kernels in SVM classification (Ting et al. 2018), and has better accuracy than existing methods such as multiple kernel learning (Rakotomamonjy et al. 2008) and distance metric learning (Zadeh et al. 2016). Isolation Kernel is also a successful way to kernelise density-based clustering (Qin et al. 2019).

These previous works have focused on the improvements on task-specific performances; but the use of Isolation Kernel has slowed the algorithms' runtimes (Ting et al. 2018; Qin et al. 2019). The feature map of Isolation Kernel was either implicitly stated (Ting et al. 2018) or not mentioned at all (Qin et al. 2019). The SVM classifier employed was based on ${}_{dual}f$ only (Ting et al. 2018).

Here we present the feature map of Isolation Kernel and its characteristic, and the benefits it bring to online kernel learning that would otherwise be impossible—*a kernel learning which can deal with infinite number of support vectors*; and run efficiently to handle large scale datasets, without compromising accuracy.

In summary, the known advantage of data dependency contributes to a trained model's high accuracy; whereas the two previously unknown advantages contribute to efficiency gain. These will be demonstrated in the empirical evaluations reported in Section 8.

4 Isolation Kernel

We provide the pertinent details of Isolation Kernel in this section. Other details can be found in Ting et al. (2018); Qin et al. (2019).

Let $D \subset \mathcal{X} \subseteq \mathbb{R}^d$ be a dataset sampled from an unknown distribution F; and $\mathbb{H}_\psi(D)$ denote the set of all partitionings H that are admissible from $\mathcal{D} \subset D$, where each point in $\mathcal{D}$ has the equal probability of being selected from D. Each H covers the entire space of $\mathbb{R}^d$, and each of the ψ isolating partitions $\theta[\mathbf{z}] \in H$ isolates one data point $\mathbf{z}$ from the rest of the points in $\mathcal{D}$, and $|\mathcal{D}| = \psi$.

Definition 1 For any two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, Isolation Kernel of $\mathbf{x}$ and $\mathbf{y}$ wrt D is defined to be the expectation taken over the probability distribution on all partitionings $H \in \mathbb{H}_\psi(D)$ that both $\mathbf{x}$ and $\mathbf{y}$ fall into the same isolating partition $\theta[\mathbf{z}] \in H, \mathbf{z} \in \mathcal{D}$:

$$K_\psi(\mathbf{x}, \mathbf{y} \mid D) = \mathbb{E}_{\mathbb{H}_\psi(D)}[\mathbf{I}(\mathbf{x}, \mathbf{y} \in \theta[\mathbf{z}] \mid \theta[\mathbf{z}] \in H)] \tag{1}$$

where $\mathbf{I}(\cdot)$ is an indicator function.

In practice, Isolation Kernel K_ψ is constructed using a finite number of partitionings $H_i, i = 1, \ldots, t$, where each H_i is created using $\mathcal{D}_i \subset D$:

$$K_\psi(\mathbf{x}, \mathbf{y}|D) \approx \frac{1}{t}\sum_{i=1}^{t} \mathbf{I}(\mathbf{x}, \mathbf{y} \in \theta \mid \theta \in H_i)$$
$$= \frac{1}{t}\sum_{i=1}^{t}\sum_{\theta \in H_i} \mathbf{I}(\mathbf{x} \in \theta)\mathbf{I}(\mathbf{y} \in \theta) \quad (2)$$

θ is a shorthand for $\theta[\mathbf{z}]$.

K is a shorthand for K_ψ; and '$|D$' is omitted for brevity hereafter.

Isolation Kernel is positive semi-definite as Eq. 2 is a quadratic form. Thus, Isolation Kernel defines a Reproducing Kernel Hilbert Space (RKHS).

4.1 iForest implementation

Here the aim is to isolate every point in $\mathcal{D}$. This is done recursively by randomly selecting an axis-parallel split to subdivide the data into two non-empty subsets until every point is isolated. Each partitioning H produces ψ isolating partitions θ; and each partition contains a single point in $\mathcal{D}$.

The algorithm $iForest$ (Liu et al. 2008) produces t $iTrees$, each built independently using a subset $\mathcal{D} \subset D$, sampled without replacement from D, where $|\mathcal{D}| = \psi$.

4.2 aNNE Implementation

As an alternative to using trees in its first implementation of Isolation Kernel (Ting et al. 2018), a nearest neighbour ensemble (aNNE) has been used instead (Qin et al. 2019).

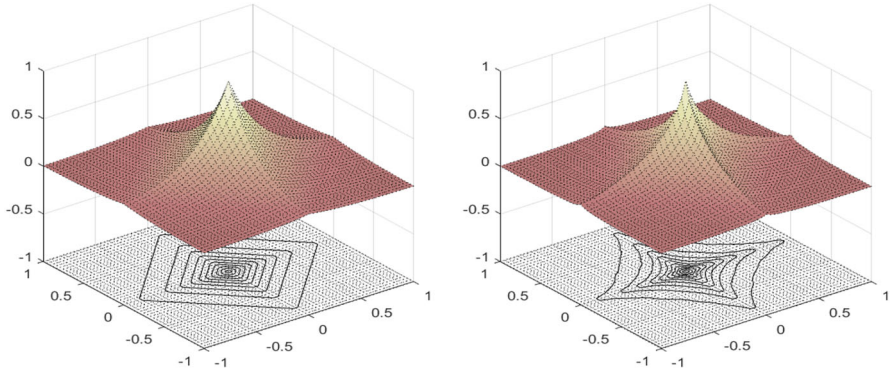
Like the tree method, the nearest neighbour method also produces each H model which consists of ψ isolating partitions θ, given a subsample of ψ points. Rather than representing each isolating partition as a hyper-rectangle, it is represented as a cell in a Voronoi diagram, where the boundary between two points is the equal distance from these two points.

H, being a Voronoi diagram, is built by employing ψ points in $\mathcal{D}$, where each isolating partition or Voronoi cell $\theta \in H$ isolates one data point from the rest of the points in $\mathcal{D}$. The point which determines a cell is regarded as the cell centre.

Given a Voronoi diagram H constructed from a sample $\mathcal{D}$ of ψ points, the Voronoi cell centred at $\mathbf{z} \in \mathcal{D}$ is:

$$\theta[\mathbf{z}] = \{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{z} = \text{argmin}_{\mathbf{z}' \in \mathcal{D}}\, \ell_p(\mathbf{x}, \mathbf{z}')\}.$$

where $\ell_p(\mathbf{x}, \mathbf{y})$ is a distance function and we use $p = 2$ as Euclidean distance in this paper.



(a) Laplacian kernel (b) Isolation Kernel

Fig. 1 Laplacian and Isolation Kernels with reference to point (0, 0) on a 2-dimensional dataset with uniform density distribution

Note that the boundaries of a Voronoi diagram is derived implicitly to be equal distance between any two points in $\mathcal{D}$; and it needs not be derived explicitly for our purpose in realising Isolation Kernel.

4.3 Kernel distributions and kernel characteristic

Figure 1 is extracted from Ting et al. (2018) which shows that the kernel distribution of Isolation Kernel approximates that of Laplacian kernel under uniform density distribution.

Figure 2 shows that the contour plots of the aNNE and iForest implementations of Isolation Kernel. Notice that the contour lines, each denotes the same similarity to the centre (the red point), are further apart from each other in the sparse region but closer in the dense region. This exhibits the unique kernel characteristic of Isolation Kernel: **two points in sparse region are more similar than two points of equal inter-point distance in dense region** (Ting et al. 2018; Qin et al. 2019). This data dependent similarity is a direct outcome of using an isolation mechanism that produces large partitions in sparse region and small partitions in dense region (Ting et al. 2018).

This data dependent similarity has led to the improved accuracy in SVM classifiers and density-based clustering (Ting et al. 2018; Qin et al. 2019), compared with the use of data independent Gaussian kernel and Euclidean distance. In contrast, any data independent kernel/distance measure has the symmetrical contour lines around the centre point, independent of data distribution (as shown in Fig. 1(a)).

The reasons why the Voronoi-diagram based implementation is better than the tree based implementation have been provided earlier (Qin et al. 2019); and this has led to better density-based clustering result than that using the Euclidean distance measure.

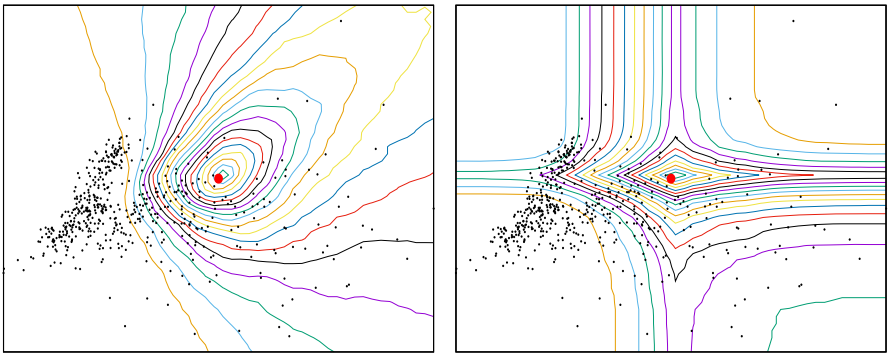


Fig. 2 Contour plots of two different implementations of Isolation Kernel on a real-world dataset. Both aNNE and iForest use $\psi = 10$ and $t = 1000$. Black dots are data points

5 Learning with the feature map of isolation kernel

This section presents our four conceptual contributions. Section 5.1 presents the feature map of Isolation Kernel. Section 5.2 describes the theoretical underpinning of efficient learning with Isolation Kernel. How Isolation Kernel enables the use of ${}_{primal}f$ in solving the primal optimisation problem, and its efficient dot product implementations are provided in the following two subsections.

5.1 The feature map of Isolation Kernel

As Isolation Kernel has no closed form expression and is derived directly from the given dataset D, Equation (2) implies that each isolating partition θ in input space can be treated as an embodiment of a feature in RKHS; and the i-component of the feature space due to H_i can be derived using the mapping $\Phi_i : \mathcal{X} \mapsto \mathbb{I}^{\psi}$ (where $\mathbb{I}$ is a binary domain). Thus, the feature map of $K_i(\mathbf{x}, \cdot)$ can be constructed using partitioning $H_i = \{\theta_j \mid j = 1, ..., \psi\}$ as:

$$\Phi_i(\mathbf{x}) = [\mathbf{I}(\mathbf{x} \in \theta_1), \ldots, \mathbf{I}(\mathbf{x} \in \theta_\psi)]^\top$$

The inner summation of Eq. (2) can then be re-expressed in terms of Φ_i as follows:

$$\sum_{\theta \in H_i} \mathbf{I}(\mathbf{x} \in \theta)\mathbf{I}(\mathbf{y} \in \theta) = \langle \Phi_i(\mathbf{x}), \Phi_i(\mathbf{y}) \rangle = K_i(\mathbf{x}, \mathbf{y})$$

Concatenating Φ_i over $i = 1, \ldots, t$ gives rise to the feature map $\Phi(\mathbf{x})$ of $K(\mathbf{x}, \cdot)$:

$$\Phi(\mathbf{x}) = [\Phi_1(\mathbf{x}), \ldots, \Phi_t(\mathbf{x})]^\top$$

Then, Isolation Kernel of Eq. (2), represented using these features, can be expressed as:

$$K(\mathbf{x}, \mathbf{y}) = \frac{1}{t} \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle$$

As every $\mathbf{x} \in \mathbb{R}^d$ falls into only one of the ψ partitions in each partitioning H_i, Φ is a sparse feature map.

Let Π be a shorthand of $\Phi_i(\mathbf{x})$ such that $\Phi_{ij}(\mathbf{x}) = 1$ and $\Phi_{ik}(\mathbf{x}) = 0, \forall k \neq j$ for any $j \in [1, \psi]$.

Φ is sparse and has the following geometrical interpretation: Every mapped point is denoted as $[\Pi, \ldots, \Pi]^\top$ but it is not a single point in RKHS. These points have $\| \Phi(\mathbf{x}) \| = \sqrt{t}$ and $\Phi_i(\mathbf{x}) = \Pi$ for all $i \in [1, t]$; and $K(\mathbf{x}, \mathbf{x}) = 1$ for all $\mathbf{x} \in \mathbb{R}^d$.

Parameters t and ψ are controlled by a user. Each setting of t and ψ yields a feature map.

Note that the roles of ψ and t are different. The larger ψ is, the sharper the kernel distribution (Ting et al. 2018). This parameter has a role similar to the bandwidth parameter in Gaussian/Laplacian kernel, where the smaller the bandwidth, the sharper its kernel distribution. Thus, this parameter must be tuned to get a good task-specific performance. The t parameter has two roles. First, a higher t leads to a better estimation of the expectation in Eq. (1); and it also leads to lower variance of the estimation. Second, it determines the dimensionality of the feature map, as shown in this section. The effects of these two parameters are demonstrated in Sect. 8.2.2.

5.2 Efficient learning with isolation kernel

This subsection describes the theoretical underpinning of efficient learning with Isolation Kernel.

In a binary class learning problem of a given training set $D = \{(\mathbf{x}_1, c_i), \ldots, (\mathbf{x}_n, c_n)\}$, where points $\mathbf{x}_i \in \mathbb{R}^d$ and class labels $c_i \in \mathcal{C} = \{+1, -1\}$, the goal of SVM is to learn a kernel prediction function f by solving the following optimisation problem (Scholkopf and Smola 2001):

$$\min_{f \in \mathcal{H}_D} \frac{\lambda}{2} \| f \|^2_{\mathcal{H}_K} + \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x}_i); c_i)$$

where $\mathcal{H}_D = span(K(\cdot, \mathbf{x}_1), \ldots, K(\cdot, \mathbf{x}_n))$ is span over all points in the training set D; $L(f(\mathbf{x}); c)$ is a convex loss function wrt the prediction of $\mathbf{x}$; and $\mathcal{H}_K$ is the Reproducing Kernel Hilbert Space endowed with a kernel K.

The computational cost of this kernel learning is high because the search space over $\mathcal{H}_D$ is large for large n.

In contrast, with Isolation Kernel, $\mathcal{H}_D$ is replaced with a smaller set $\mathcal{H}_T = span(\Phi_1(\cdot), \Phi_2(\cdot), \ldots, \Phi_t(\cdot)) \subset \{0, 1\}^{t\psi}$.

As $|\cup_{i=1}^{t}\mathcal{D}_i| \ll |D|$ and $\mathcal{H}_T \subset \mathcal{H}_D$, learning with Isolation Kernel is expected to be faster than learning with commonly used data independent kernels such as Gaussian and Laplacian kernels.

The following subsections provide the implementations, due to the use of Isolation Kernel, which enable the significant efficiency gain without compromising predictive accuracy for online kernel learning.

5.3 Using ${}_{primal}f$ instead of ${}_{dual}f$

As Isolation Kernel has a finite-dimensional feature map, this facilitates the use of prediction function ${}_{primal}f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x})\rangle$; thus solving the primal optimisation problem is a natural choice.

The evaluation of ${}_{primal}f$ is faster than that of ${}_{dual}f$, when the number of support vectors (s) times the number of attributes of $\mathbf{x}$ (d) is more than the effective number of features of $\Phi(\mathbf{x})$, i.e., $sd > t$ (see the reason why t is the effective number of feature of the $\Phi(\mathbf{x})$ in next subsection). Its use yields a significant speedup when the domain is high dimensional and/or in an online setting where the points can potentially be infinite. The online setting necessitates the need to have a kernel learning system which can deal with potentially infinite number of support vectors. The procedure of such a kernel learning system using Isolation Kernel is described in Sect. 6.

5.4 Efficient dot product in ${}_{primal}f$

The use of Isolation Kernel facilitates an efficient dot product $\langle \mathbf{w}, \Phi(\mathbf{x})\rangle$ in ${}_{primal}f$. Recall that, $\forall \mathbf{x} \in \mathbb{R}^d$, $\Phi_i(\mathbf{x})$ has exactly one feature having value=1 in a vector of ψ binary features (stated in Sect. 5.1). Thus, $\langle \mathbf{w}, \Phi(\mathbf{x})\rangle$ can be computed with a summation of t number of w_{ij} (rather than the naive dot product, computing $t\psi$ products $w_{ij} \times \Phi_{ij}(\mathbf{x})$):

$$\langle \mathbf{w}, \Phi(\mathbf{x})\rangle = \sum_{i=1}^{t}\sum_{j=1}^{\psi} w_{ij} \times \Phi_{ij}(\mathbf{x}) = \sum_{i=1, j=\phi_i(\mathbf{x})\in\mathbb{Z}}^{t} w_{ij}$$

where $\Phi_{ij}(\mathbf{x})$ denotes the value of binary feature j of $\Phi_i(\mathbf{x})$; and $\phi_i(\mathbf{x}) = j$ serves as an index to the j-th element of $\Phi_i(\mathbf{x})$ indicating $\mathbf{x} \in \theta_j$.

In summary, $\langle \mathbf{w}, \Phi(\mathbf{x})\rangle$ can be computed more efficiently using ϕ as an indexing scheme.

Note that this efficient dot product is independent of ψ. For large ψ, this dot product could result in orders of magnitude faster than using the naive dot product (see Fig. 4 in Sect. 8.1.2 later).

The indexing scheme ϕ of the feature map of Isolation Kernel is constructed in two steps as shown in Table 2 that convert $\mathbf{x} \in \mathbb{R}^d \rightarrow \phi(\mathbf{x}) \in \mathbb{Z}^t$. The steps taken by the Nyström method (Williams and Seeger 2001; Lu et al. 2016) to construct an approximate feature map is also shown for comparison in the same table.

Table 2 Feature map construction comparison: Nyström versus Isolation Kernel

	Nsytröm (approximate feature map of a chosen kernel $K(\cdot,\cdot)$)		Isolation Kernel ($\phi(\mathbf{x})$)
1.	Sample $\{\mathbf{x}_i \mid i = 1,\ldots,b\}$ from D to construct kernel matrix $\mathbf{K}$	1.	Sample ψ points from D, t times, to construct t partitionings $H_i \in \mathbb{H}_\psi(D)$; and each H_i has ψ partitions.
2.	$(\mathbf{V}_r, \mathbf{A}_r) = eigens(\mathbf{K}, r)$, where $\mathbf{V}_r$ and $\mathbf{A}_r$ are eigenvectors and eigenvalues of $\mathbf{K}$.	2.	$\mathbf{x} \in \mathbb{R}^d \rightarrow \phi(\mathbf{x}) \in \mathbb{Z}^t$, where each integer attribute has values: $1,\ldots,\psi$; and each integer is an index to a partition $\theta \in H_i$. The t attributes represent the partitionings $H_i, i = 1,\ldots,t$.
3.	$\mathbf{x} \in \mathbb{R}^d \rightarrow \acute{\mathbf{x}} \in \mathbb{R}^r$: Convert $\mathbf{x} \in D$ to $\acute{\mathbf{x}} \in \acute{D}$: $\acute{\mathbf{x}} = \mathbf{A}_r^{-0.5}\mathbf{V}_r^\top[K(\mathbf{x},\mathbf{x}_1),\ldots,K(\mathbf{x},\mathbf{x}_b)]^\top$		Convert $\mathbf{x} \in D$ to $\phi(\mathbf{x}) \in \acute{D}$: $\mathbf{x}$ is parsed over the t partitionings.
	Perform learning with feature map on $\acute{D}$		

The computational cost of the mapping from $\mathbf{x}$ to either $\Phi(\mathbf{x})$ or $\phi(\mathbf{x})$ is linear to $t\psi$. But this mapping needs to be done only once for each point. That is, every point needs to examine each partitioning H only once to determine the partition $\theta \in H$ into which the point falls.

5.5 Time complexities

When Isolation Kernel is implemented using iForest (Ting et al. 2018): Each tree building takes $O(\psi)$; and the feature mapping takes $O(\log \psi)$ for each point. Thus, the total build time of t trees is $O(t\psi)$; and the mapping time for n points takes $O(nt \log \psi)$.

Implemented using aNNE (Qin et al. 2019): No build time is required, as it simply stores the $t\psi d$ points, where d is the number of input dimensions. The feature mapping of n points takes $O(nt\psi d)$. As the nearest neighbour search is amenable to parallelisation, the mapping time can be reduced by a factor of m parallelisations to $O(\frac{n}{m}t\psi d)$. With GPU, this parallelisation can lead to orders of magnitude faster. See Sect. 8.3 for the comparison result.

6 Applications to kernel learning that uses online gradient descent and support vector machines

Online kernel learning aims to build an efficient and scalable kernel-based predictive model incrementally from a sequence of potentially infinite data points. One of the early methods is given by Kivinen et al. (2001). One key challenge of online kernel learning is managing a growing number of support vectors, as every misclassified point is typically added to the set of support vectors.

One recent implementation of online kernel learning is called OGD (Lu et al. 2016) which employs ${}_{dual}f$:

$$f(\mathbf{x}) = \sum_{i=1}^{s} \alpha_i c_i K(\mathbf{x}_i, \mathbf{x})$$

If $L(f(\mathbf{x}); c) > 0$ (incorrect prediction) then add $\mathbf{x}$ to the set of support vectors with $\alpha = -\eta \nabla L(f(\mathbf{x}); c)$, where η is the learning rate.

Without setting a budget, the number of support vectors (s) usually increases linearly with the number of points observed. Therefore, the testing time becomes increasingly slower as the number of points observed increases.

Here we show the benefits of Isolation Kernel will bring to online kernel learning: Its use improves both the time and space complexities of OGD significantly from $O(sd)$ to $O(t\psi)$ for every prediction while *allowing s to be infinite—eliminating the need to have a budget for support vectors*. This is because $t\psi$ is constant while s grows as more points are observed.

This is done on exactly the same OGD implementation. The only change required in the procedure is that the function f is evaluated based on its feature map Φ of Isolation Kernel as follows:

$$f(\mathbf{x}) = \sum_{i=1}^{s} \alpha_i c_i \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \rangle = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle,$$

where $\mathbf{w} = \sum_{i=1}^{s} \alpha_i c_i \Phi(\mathbf{x}_i)$.

During training, s is the number of support vectors at the time an evaluation of the prediction function is required. For every addition of a new support vector $\mathbf{x}$ during the training process, the weight vector $\mathbf{w} = \mathbf{w} + \alpha c \Phi(\mathbf{x})$ is updated incrementally while s increments. At the end of the training process, the final $\mathbf{w}$ is ready to be used with ${}_{primal}f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ to evaluate every test point $\mathbf{x}$.

Although the above expressions are in terms of Φ, the computation is conducted more efficiently using ϕ, effectively as an indexing scheme for Φ, as described in Sect. 5.4, for $\langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ as well as $\sum_{i=1}^{s} \alpha_i c_i \Phi(\mathbf{x}_i)$.

We named the OGD implementation which employs Isolation Kernel and ${}_{primal}f$ as IK-OGD. The algorithms of OGD (as implemented by Lu et al. (2016)) and IK-OGD are shown as Algorithms 1 and 2, respectively.

Algorithm 1 $OGD(\eta, K)$

Require: η - learning rate; K - Kernel.
1: Initialize set of support vectors $S = \emptyset$;
2: **while** there is a new point $\mathbf{x}$ **do**;
3: $f(\mathbf{x}) = \sum_{i=1}^{|S|} \alpha_i c_i K(\mathbf{x}_i, \mathbf{x})$;
4: **if** $L(f(\mathbf{x}); c) > 0$ (incorrect prediction) **then**
5: Add $\mathbf{x}$ to S with $\alpha = -\eta \nabla L(f(\mathbf{x}); c)$;
6: **end if**
7: **end while**

Algorithm 2 $IK\text{-}OGD(\eta, \Phi)$

Require: η - learning rate; Φ - feature mapping of IK.
1: Initialize $\mathbf{w}$ to $\mathbf{0}$;
2: **while** there is a new point $\mathbf{x}$ **do**;
3: $g(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$;
4: **if** $L(g(\mathbf{x}); c) > 0$ (incorrect prediction) **then**
5: $\mathbf{w} = \mathbf{w} - \eta \nabla L(g(\mathbf{x}); c) c \Phi(\mathbf{x})$;
6: **end if**
7: **end while**

To apply Isolation Kernel to support vector machines, we only need to use the algorithm which solves the primal optimisation problem such as LIBLINEAR (Fan et al. 2008) after converting the data using the feature map of Isolation Kernel.

7 Experimental settings

We design experiments to evaluate the impact of Isolation Kernel on Online Kernel Learning. We use the implementations of the kernelised online gradient descent (OGD) and Nyström online gradient descent (NOGD)[1]. The kernelised online gradient descent (Kivinen et al. 2001) or OGD solves the dual optimisation problem; whereas IK-OGD solves the primal optimisation problem, so as NOGD (Lu et al. 2016). We also compare with a recent online method that employs multi-kernel learning and random fourier features, called AdaRaker (Shen et al. 2018).

Laplacian kernel is used as a base-line kernel because Isolation Kernel (implemented using Isolation Forest) approximates Laplacian kernel under uniform density distribution[2]. As a result, Isolation Kernel and Laplacian kernel can be expressed using the same 'sharpness' parameter ψ.

Two existing implementations of Isolation Kernel are used: (1) Isolation Forest (Liu et al. 2008), as described in Ting et al. (2018); and (2) aNNE, a nearest neighbour ensemble that partitions the data space into Voronoi diagram, as described in Qin et al. (2019). We refer IK-OGD to the iForest implementation. When a distinction is required, we denote IK_i-OGD as the iForest implementation; and IK_a-OGD the aNNE implementation. The code for IK-OGD is available at https://github.com/IsolationKernel/Codes.

All OGD related algorithms used the hinge loss function and the same learning rate $\eta = 0.5$, as used in Lu et al. (2016). The only parameter search required for these algorithms is the kernel parameter. The search range in the experiments is listed in Table 3. The parameter is selected via 5-fold cross-validation on the training set. Once the best parameter is obtained, it is used to train one model; and the testing set is used to assess the accuracy of this model. Because the training set and testing set are large,

[1] The code is available at https://github.com/LIBOL/KOL.

[2] As pointed in Ting et al. (2018), Laplacian kernel can be expressed as $\exp(-\gamma \sum_{j=1}^{d} |x_j - y_j|) = \psi^{-\frac{1}{d}\sum_{j=1}^{d}|x_j - y_j|}$, where $\gamma = \frac{\log(\psi)}{d}$. Laplacian kernel has been shown to be competitive to Gaussian kernel in SVM in a recent study (Ting et al. 2018).

Table 3 Search ranges of parameters

Kernel/Algorithm	Search range
Laplacian & Isolation Kernels	$\psi \in \{2^m \mid m = 2, 3, \ldots, 12\}$
AdaRaker	$\lambda \in \{10^m, \frac{10^m}{2} \mid m = -2, -3, -4, -5\}$
	$\sigma \in \{2^m \mid m = -10, \ldots, 4, 5\}$

the train-and-test experiment was conducted once only (unless stated otherwise), as done by previous studies (e.g., Lu et al. (2016) and Wang et al. (2012).)

The default settings for NOGD (Lu et al. 2016) are: the Nyström method uses the Eigenvalue-Decomposition; and sampling size or budget[3] $b = 100$; and the matrix rank is set to $r = 0.2b$. The default parameter used to create Isolation Kernel is set to $t = 100$.

AdaRaker (https://github.com/yanningshen/AdaRaker) employs sixteen Gaussian kernels and the specified bandwidths σ for these kernels are listed in Table 3. (The default three kernels in the code gave worse accuracy than that reported in the next section). In addition, AdaRaker uses 50 orthogonal random features (equivalent to $b = 100$ for the Nyström method) and $\eta = 0.5$ as default. The search range of λ through 5-fold cross-validation is given in Table 3.

Eleven datasets from www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ are used in the experiments. The properties of these datasets are shown in Table 4. The datasets are selected in order to have diverse data properties: data sizes (20,000 to 2,400,000) and dimensions (22 to more than 3.2 million). Because the OGD and NOGD versions of the implementation we used work on two-class problems only, three multi-class datasets have been converted to two-class datasets of approximately equal class distribution[4].

Four experiments are conducted: (a) in online setting, (b) in batch setting, (c) examine the runtime in GPU, and (d) an investigation using SVM. The CPU experiments ran on a Linux CPU machine: AMD 16-core CPU with each core running at 1.8 GHz and 64 GB RAM. The GPU experiments ran on a machine having GPU: 2 x GTX 1080 Ti with 3584 (1.6 GHz) CUDA cores & 12GB graphic memory; and CPU: i9-7900X 3.30GHz processor (20 cores), 64GB RAM.

The results are presented in four subsections of Sect. 8.

In online setting, we simulate an online setting using each of the four largest datasets over half a million points (after combining their given training and testing sets) as follows. Given a dataset, it is first shuffled. Then, the initial training set has data size as the training set size shown in Table 4; and it is used to determine the best parameter based on 5-fold cross-validation before training the first model. The online stream is assumed to arrive sequentially in blocks of 1000 points. Each block is assumed to have no class labels initially: In testing mode, the latest trained model is used to make a prediction for every point in the block. After testing, class labels are made available:

[3] Note that this parameter is called budget in Lu et al. (2016); but this is different from the budget used to limit the number of support vectors, mentioned in Section 2.1.

[4] The two-class conversions from the original class labels were done for three multi-class datasets: mnist: {3, 4, 6, 7, 9} and {0, 1, 2, 5, 8}. smallNORB: {1, 4} and {0, 2, 3}. cifar-10: {0, 2, 3, 4, 5} and {1, 6, 7, 8, 9}.

Table 4 Properties of the datasets used in the experiments. $nnz\% = \#nonzero\_values/((\#train+\#test)\times \#dimensions)\times 100$. Datasets with $nnz\% < 1\%$ are regarded as sparse datasets; otherwise, they are dense datasets

	#train	#test	#dimensions	nnz%
url	30,000	2,366,130	3,231,961	0.0036
news20.binary	15,997	3,999	1,355,191	0.03
rcv1.binary	20,242	677,399	47,236	0.16
real-sim	57,848	14,461	20,958	0.24
smallNORB	24,300	24,300	18,432	100.0
cifar-10	50,000	10,000	3,072	99.8
epsilon	400,000	100,000	2,000	100.0
mnist	60,000	10,000	780	19.3
a9a	32,561	16,281	123	11.3
covertype	464,810	116,202	54	22.1
ijcnn1	49,990	91,701	22	59.1

The block is in training mode and the model is updated[5]. The above testing and training modes are repeated for each current block in the online stream until the data run out. The test accuracy up to the current block is reported along the data stream.

In batch setting, we report the result of a single trial of train-and-test for each dataset which consists of separate training set and testing set. The assessments are in terms of predictive accuracy and the total runtime of training and testing. Since AdaRaker has problem dealing with large datasets, it is used in the batch setting only.

In online setting, Isolation Kernel and ϕ for IK-OGD are established using the initial training set only. Once established, the kernel and ϕ are fixed for the rest of the data stream. This applies to the b points selected for NOGD as well. In the batch setting, the given training set is used for these purposes.

8 Empirical results

8.1 Results in online setting

Figure 3 shows that, in terms of accuracy, IK-OGD has higher accuracy than OGD and NOGD on four datasets, except that OGD has better accuracy on epsilon only[6]. Notice that, as more points are observed, OGD and IK-OGD have more rooms for accuracy improvement than NOGD because the former two have no budget and the

[5] This simulation is more realistic than the previous online experiments which assume that class label of each point is available immediately after a prediction is made to enable model update (Lu et al. 2016). In practice, the algorithm can be made to be in the training mode whenever class labels are available, either partially or the entire block.

[6] Note that the first points in the accuracy plots can swing wildly because each is the accuracy on the first data block.

latter has a limited budget. We will examine the extent to which increasing the budget and t improve the accuracies of NOGD and IK-OGD, respectively, in Sect. 8.2.

In terms of runtime, IK-OGD runs faster than both OGD and NOGD on high dimensional datasets (url, rcv1.binary and epsilon); and it is only slower than NOGD in the low dimensional covertype dataset. Notice that the gap in runtime between IK-OGD and NOGD stays the same over the period because the time spent on ${}_{primal}f$ is the same. In contrast, the gap between OGD and IK-OGD increases over time because the time spent on ${}_{dual}f$ used by OGD increases as the number of support vectors increases over time. The runtimes of IK-OGD and NOGD are in the same order; but IK-OGD is 2 to 4 orders magnitude faster than OGD; and on url, OGD could only complete the first five points in Fig. 3(a) & 3(b) after more than one week.

NOGD maintains fast execution by limiting the number of support vectors while using ${}_{primal}f$. The use of Laplacian kernel (or any other kernel) which has infinite or large number of features necessitates the use of a feature map approximation method. Despite all these measures for efficiency gain in NOGD, IK-OGD without budget still ran faster than NOGD with budget ($b = 100$) on the three high-dimensional datasets! The efficiency gain in NOGD is a trade-off with accuracy—both the feature map approximation and the limit on the number of support vectors reduce the accuracy.

The use of Isolation Kernel provides a cleaner and simpler utilisation of ${}_{primal}f$ in online setting than the kernel functional approximation approach (in which NOGD is a good representative method). As a result, IK-OGD achieves the efficiency gain without compromising the accuracy because of its data dependency.

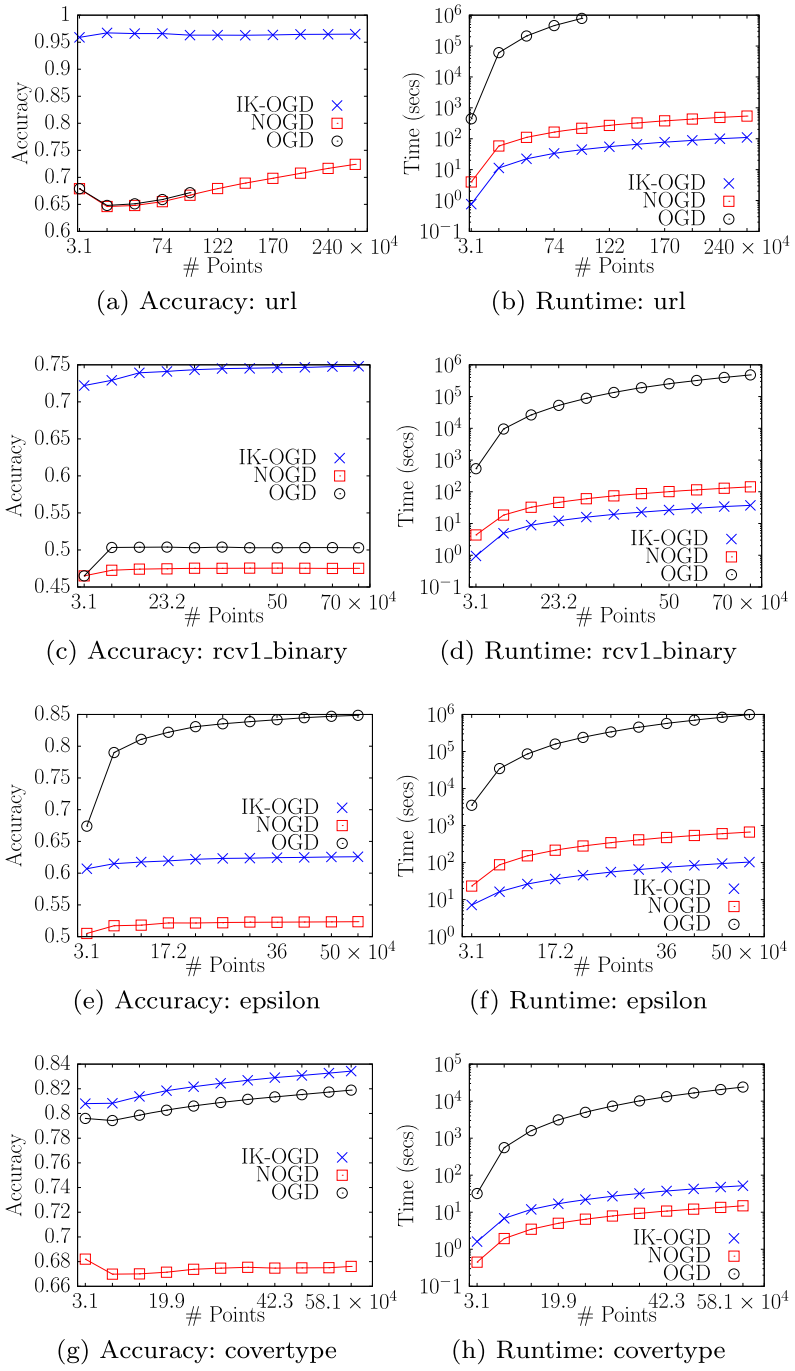
The next two subsections provide empirical evidence of efficiency gains by using IK-OGD, previously described in Sects. 5.3 and 5.4.

8.1.1 The effect of ${}_{primal}f$ or ${}_{dual}f$ on IK-OGD

To demonstrate the impact of the type of prediction function used in IK-OGD (stated in Sect. 5.3), we create a version which employs ${}_{dual}f$ named IK-OGD(dual) to compare with IK-OGD which employs ${}_{primal}f$.

The proportions of time spent on the two prediction functions out of the total runtimes are given as follows: IK-OGD took 2.3% and 0.77% on rcv1.binary and epsilon, respectively. In contrast, IK-OGD(dual) took 99.9% and 99.8%, respectively. This shows that ${}_{primal}f$ has reduced the time spent on the prediction function from almost the total runtime to a tiny fraction of the total runtime!

The total runtimes of IK-OGD versus IK-OGD(dual) are 37 seconds versus 280,656 seconds on rcv1.binary; and 103 seconds versus 235,966 seconds on epsilon. In other words, it also reduced the total runtime significantly by 3 to 4 orders of magnitude. The difference in runtimes enlarges as more points are observed because the number of support vectors increases which affects IK-OGD(dual) only. The number of support vectors used at the end of the data stream is: 349,009 for rcv1.binary; and 349,481 for epsilon.



(a) Accuracy: url (b) Runtime: url

(c) Accuracy: rcv1_binary (d) Runtime: rcv1_binary

(e) Accuracy: epsilon (f) Runtime: epsilon

(g) Accuracy: covertype (h) Runtime: covertype

Fig. 3 Results in online setting in terms of accuracy and runtime (which were what each algorithm could complete within one week)

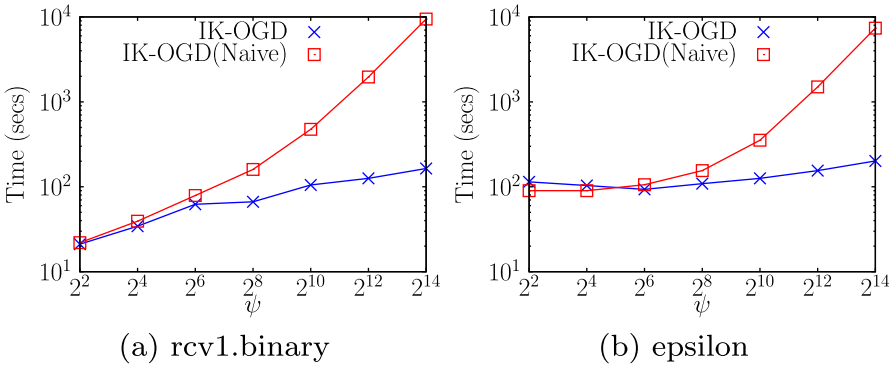


Fig. 4 Runtime comparison: IK-OGD vs IK-OGD(naive) with increasing ψ with $t = 100$ in online setting

8.1.2 The effect of efficient dot product on IK-OGD

Here we show the effect of the efficient dot product, described in Sect. 5.4. The implementation which computes the summation of $t\psi$ products is named IK-OGD(naive). It is compared with IK-OGD with the efficient implementation. As the impact on runtimes varies with ψ, the experiment is conducted with increasing ψ.

Figure 4 shows that the runtime difference between IK-OGD and IK-OGD(naive) enlarges as ψ increases; and IK-OGD(naive) was close to two orders of magnitude slower than IK-OGD at $\psi = 16384$ on both datasets. Note that the efficient dot product in IK-OGD is independent of ψ. IK-OGD's runtime depends on ψ only in the process of mapping $\mathbf{x}$ to $\phi(\mathbf{x})$ (recall the mapping stated in Table 2).

8.2 Results in batch setting

Observations from the results shown in Table 5 in terms predictive accuracy are:

- IK$_i$-OGD performs better than OGD on six datasets; it has equal or approximately equal accuracy on mnist, a9a and ijcnn1. This outcome is purely due to the kernel employed—Isolation Kernel approximates Laplacian kernel under uniform density distribution; and it adapts to density structure of the given dataset (Ting et al. 2018). This relative result between Isolation Kernel and Laplacian Kernel on OGD is consistent with the previous relative result on SVM (Ting et al. 2018). The only two datasets on which IK$_i$-OGD performs significantly worse than OGD are smallNORB and epsilon. We will see in Section 8.2.2 that the gap can be significantly reduced by increasing t, without a significant runtime increase.
- NOGD has lower accuracy than OGD on seven out of eleven datasets because it employs an approximate feature map of the Laplacian kernel. As a consequence, NOGD can be significantly worse than OGD. Examples are smallNORB, cifar-10, epsilon and mnist. While increasing its budget may improve NOGD's accuracy to approach the level of accuracy of OGD; it will still perform worse than IK$_i$-OGD. Indeed, NOGD performed worse than IK$_i$-OGD on ten out of eleven datasets in Table 5.

Table 5 Comparing IK-OGD with OGD, NOGD and AdaRaker in terms of accuracy. OGD and NOGD use Laplacian kernel; IK_i-OGD uses Isolation Kernel implemented with iForest; and IK_a-OGD uses Isolation Kernel implemented with aNNE. AdaRaker employs 16 Gaussian kernels. The best and the worst accuracies on each dataset are boldfaced and underlined, respectively. AdaRaker could not complete on many datasets, indicated as '—'.

	Accuracy				
	OGD	IK_i-OGD	IK_a-OGD	NOGD	AdaRaker
url	.67	**.96**	**.96**	.67	—
news20.binary	.50	.57	**.89**	.50	—
rcv1.binary	.48	.55	**.96**	.48	—
real-sim	.73	.76	**.96**	.69	—
smallNORB	**.93**	.78	.88	.51	—
cifar-10	.69	.72	**.73**	.50	.54
epsilon	**.88**	.65	.71	.57	—
mnist	.97	.95	**.98**	.85	.80
a9a	**.84**	**.84**	**.84**	**.84**	.79
covertype	.76	.86	**.92**	.70	.70
ijcnn1	.94	.95	**.97**	.93	.90

– IK_a-OGD has equal or better accuracy than IK_i-OGD. This result is consistent with the assessment comparing the two implementations of Isolation Kernel in density-based clustering (Qin et al. 2019). This is because Voronoi diagram produces partitions of non-axis-parallel regions; whereas iForest yields axis-parallel partitions only. Notice that the accuracy difference between IK_a-OGD and OGD is huge on url, news20, rcv1, real-sim and covertype.

In terms of runtime results shown in Table 6:

– While OGD and IK_i-OGD are using exactly the same training procedure (with the exception of the prediction function used), IK_i-OGD has advantage in two aspects:

  i) The differences in runtimes are huge—IK_i-OGD is three orders of magnitude faster than OGD on seven out of the eleven datasets; and at least one order of magnitude faster on other datasets. This is due to the efficient implementations made possible through Isolation Kernel, described in Sect. 5.
  ii) Both OGD and IK_i-OGD can potentially incorporate an infinite number of support vectors. But, the prediction function used has denied OGD the opportunity to live up to its full potential because its testing time complexity is proportional to the number of support vectors. In contrast, IK_i-OGD has constant test time complexity, independent of the number of support vectors.

– Compare with NOGD, IK_i-OGD is up to one order of magnitude faster in runtime in high dimensional datasets. On low dimensional datasets (100 or less), IK_i-OGD ran only slightly slower. This is remarkable given that IK-OGD has no budget and NOGD has a budget of 100 support vectors only. As a result, NOGD has lower accuracy than IK_i-OGD on all datasets, except a9a.

Table 6 Comparing IK-OGD with OGD, NOGD and AdaRaker in terms of total runtime of training and testing in seconds. The best and the worst runtime on each dataset are boldfaced and underlined, respectively. -ME- denotes memory errors

	Runtime (CPU seconds)			
	OGD	IK$_i$-OGD	NOGD	AdaRaker
url	65,319	**62**	303	-ME-
news20.binary	915	**1**	11	-ME-
rcv1.binary	10,499	**22**	114	-ME-
real-sim	1,468	**2**	6	-ME-
smallNORB	64,183	**73**	353	> 1 week
cifar-10	20,260	**15**	69	5,661
epsilon	496,065	**106**	430	> 1 week
mnist	659	**4**	12	1,453
a9a	95	3	**2**	308
covertype	20, 863	25	**10**	3,740
ijcnn1	76	8	**2**	576

The result of IK$_a$-OGD can be found in Sect. 8.3 and Table 7. IN a nutshell, IK-OGD inherits the advantages of OGD (no budget) and NOGD (the use of $_{primal}f$ has low time complexity); yet, it does not have their disadvantages: the use of $_{dual}f$ in OGD has high time complexity; and NOGD needs to have a budget which lowers its predictive accuracy.

8.2.1 Comparison with AdaRaker

Tables 5 and 6 show that multi-kernel learning AdaRaker (Shen et al. 2018) has lower accuracy than OGD (and even NOGD) using a single kernel. This result is consistent with the previous comparison between SimpleMKL (Rakotomamonjy et al. 2008) and SVM using Isolation Kernel (Ting et al. 2018). Out of the five datasets on which it could run within reasonable time and without memory errors, AdaRaker ran slower than OGD in three datasets; but faster in two. Compare with IK$_i$-OGD and NOGD, AdaRaker is at least two orders of magnitude slower on the five datasets. AdaRaker has memory error issues with high dimensional datasets.

8.2.2 The effects of *t* on IK-OGD and *b* on NOGD

Two datasets, epsilon and smallNORB, are used in this experiment because the accuracy differences between OGD and NOGD on these datasets are the largest; and they are the only two datasets in which IK-OGD performed significantly worse than OGD. We examine the effects of parameters t and b on IK-OGD and NOGD.

Figure 5 shows that IK-OGD's accuracy is improved significantly as t increases. Note that, using $t = 10,000$ on epsilon, the accuracy of IK-OGD reached the same level of accuracy of OGD shown in Table 5; yet, IK-OGD still ran two orders of

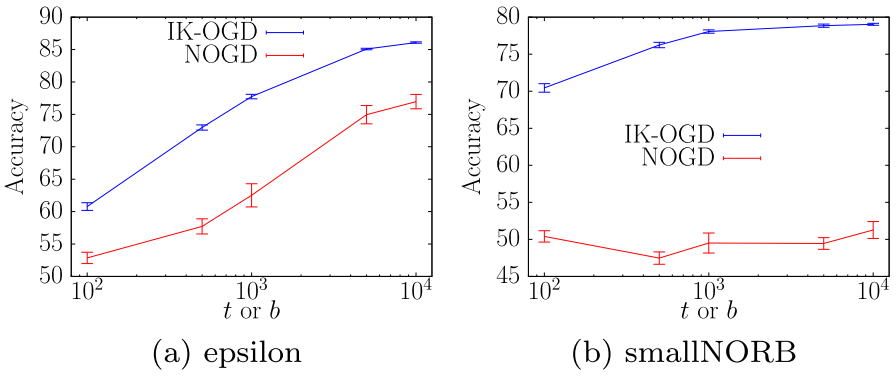


Fig. 5 Accuracy comparison on varying t or b: Experiments with increasing t for IK$_i$-OGD; and increasing b for NOGD (where $\psi = 8$.) Each accuracy is averaged over 10 trials with different random seeds

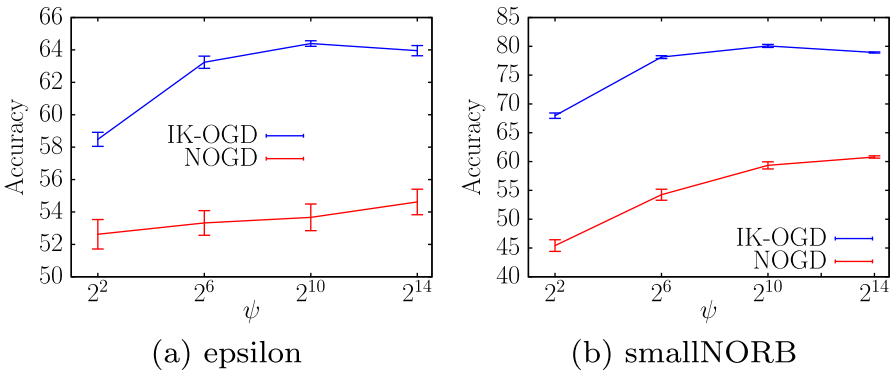


Fig. 6 Accuracy comparison on varying ψ: Experiments with increasing ψ for Isolation Kernel in IK$_i$-OGD and Laplacian kernel in NOGD (where $t = b = 100$). Each accuracy is averaged over 10 trials with different random seeds

magnitude faster than OGD. In contrast, although NOGD's accuracy has improved when b was increased from 100 to 10,000 on epsilon, it still performed worse than OGD and IK-OGD by a large margin of 10%. On smallNORB, IK-OGD also improves its accuracy as t increases; but NOGD has showed little improvement over the entire range between $b = 100$ and $b = 10,000$.

The effect of increasing t in IK can be seen from Eq. (2), where higher t provides a better approximation to Eq. (1); thus, leading to a higher accuracy for the classifier that employs IK.

In addition, a high t also leads to low variance of the accuracy. For example, the standard error has decreased from 0.6 to 0.1 as t increases from 100 to 10,000 in both datasets shown in Fig. 5. This is a direct outcome of a large ensemble for any ensemble based methods. In contrast, NOGD has maintained large standard errors over the entire range between $b = 100$ and $b = 10,000$.

For completeness, we have also provided the accuracy comparison between IK$_i$-OGD and NOGD by varying ψ with $t = b = 100$, shown in Fig. 6.

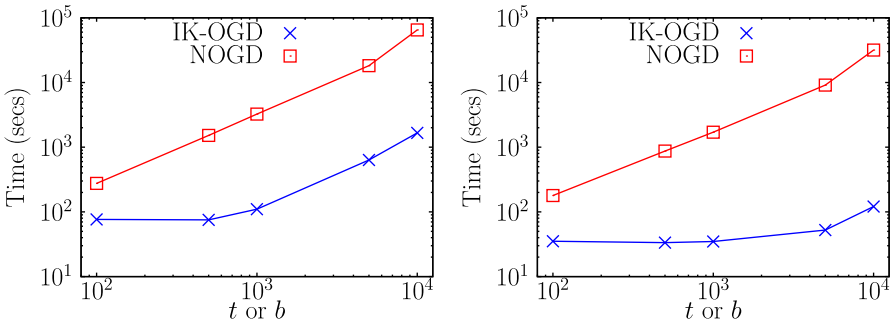


Fig. 7 Runtime comparison: Experiments with increasing t for IK$_i$-OGD; and increasing b for NOGD. Each accuracy is averaged over 10 trials with different random seeds

Figure 7 shows that NOGD at $b = 10,000$ ran two orders of magnitude slower than NOGD at $b = 100$. NOGD's runtime increases linearly wrt b; whereas the runtime of IK-OGD increases sublinearly wrt t.

8.3 CPU and GPU versions of IK$_a$-OGD

The use of Voronoi diagram to partition the data space for Isolation Kernel has slowed down the runtime significantly, compared to that implemented using iForest, mainly due to the need to search for nearest neighbours. However, because the search for nearest neighbours is amenable to GPU accelerations, we investigate a runtime comparison of the CPU and GPU versions of IK$_a$-OGD.

The result is shown in Table 7. The GPU version of IK$_a$-OGD is up to four orders of magnitude faster than the CPU version. Despite this GPU speedup, IK$_a$-OGD is still up to one order of magnitude slower than IK$_i$-OGD ran on CPU on some datasets.

In summary, GPU is a good means to speed up IK$_a$-OGD. When accuracy is paramount, IK$_a$-OGD is always a better choice than IK$_i$-OGD (as shown in Table 5) though the former, even with GPU, runs slower than the latter with CPU.

Note that while it is possible to speed up the original OGD which employs the dual prediction function using GPU, it is not a good solution for two reasons. First, it does not improve OGD's accuracy if the same data independent kernel is used. Second, the GPU-accelerated OGD is expected to still run slower than the CPU version of OGD which employs the primal prediction function using the same kernel.

The runtime reported in Table 7 consists of two components: feature mapping time and OGD runtime. For example, the longest GPU runtime is on epsilon which consists of feature mapping time 457 GPU seconds and OGD runtime of .9 CPU seconds. In other words, the bulk of the runtime is spent on feature mapping; and OGD took only a tiny fraction of a second to complete the job with CPU.

Table 7 Runtime comparison of the CPU and GPU versions of IK_a-OGD in batch setting (in CPU and GPU seconds, respectively)

	CPU	GPU
url	1,527	65
news20.binary	1,079	10
rcv1.binary	100,247	67
real-sim	31,946	10
smallNORB	406,256	178
cifar-10	340,047	147
epsilon	1,029,092	458
mnist	56,774	45
a9a	5,589	3
covertype	100,081	42
ijcnn1	11,999	3

Table 8 SVM versus IK_a-SVM. Runtime in CPU seconds. SVM is LIBSVM with Laplacian kernel; and IK_a-SVM is LIBLINEAR with Isolation Kernel

	Accuracy		Runtime	
	SVM	IK_a-SVM	SVM	IK_a-SVM
url	.67	.96	29,528	1.3
news20.binary	.50	.92	684	1.5
rcv1.binary	.54	.96	7,472	.6
real-sim	.75	.96	1,116	1.2
smallNORB	—	.88	$>$ 12 hrs	.9
cifar-10	.51	.71	3,703	1.2
epsilon	—	.70	$>$ 12 hrs	90.0
minst	.98	.99	919	1.0
a9a	.85	.84	69	.5
covtype	—	.93	$>$ 12 hrs	63.1
ijcnn1	.99	.98	59	2.4

8.4 Results with SVM

8.4.1 SVM versus IK-SVM

Without kernel functional approximation, Isolation Kernel is the only nonlinear kernel, as far as we know, that allows the trick of using $_{primal}f$ to be applied to kernel-based methods, including SVM. We apply Isolation Kernel to SVM to produce IK-SVM. It is realized using LIBLINEAR (Fan et al. 2008) since IK-SVM is equivalent to applying the IK feature mapped data to a linear SVM. IK-SVM is compared with LIBSVM (Chang and Lin 2011) with Laplacian kernel (denoted as SVM).

Table 8 shows the comparison result of SVM and IK_a-SVM. The relative result between SVM and IK_a-SVM is reminiscent of that comparing OGD with IK_a-OGD in

Table 5, i.e., IK_a-SVM has better accuracy than SVM in all high dimensional datasets; and they have comparable accuracy in datasets less than 2000 dimensions (mnist, a9a and ijcnn1). In terms of runtime, IK_a-SVM is up to four orders of magnitude faster.

Our result in Table 8 shows that Isolation Kernel enables SVM to deal with large datasets that would otherwise be impossible practically.

Note that the runtime reported in Table 8 does not include the feature mapping time. With GPU, adding the GPU runtime reported in Table 7 (the bulk is the feature mapping time) to that of IK_a-SVM does not change the conclusion: IK_a-SVM runs order(s) of magnitude faster than SVM and has better accuracy in high dimensional and large scale datasets.

8.4.2 Compare with additive kernels using SVM

Additive kernels are a class of nonlinear kernels which has approximate feature maps that can be computed efficiently (Vedaldi and Zisserman 2012a; Maji et al. 2013). These include chi-square and intersection kernels.

They were reported to work well in image datasets when substantial feature engineering such as convolution is performed (e.g., Vedaldi and Zisserman (2012a); Maji et al. (2013)).

Like the Random Fourier Features, the approximate feature maps of additive kernels are generated independent of the given dataset by sampling the continuous spectrum of its Fourier features. The approximate feature map of an additive kernel (Vedaldi and Zisserman 2012a) has $2b + 1$ features per input dimension, where b is a user-control parameter. In other words, the feature map has features $2b + 1$ times more than the total number of input dimensions.

One advantage of additive kernels over other kernels is that they have no kernel parameter which needs tuning.

Table 9 shows the results of comparing kernel and its feature map using SVM for χ^2 additive kernel and Isolation Kernel. Interestingly, the approximate feature map of χ^2 kernel resulted LIBLINEAR to produce accuracies equal or close to those produced by LIBSVM [7]. The same result applies to IK_a [8].

Note that the method that generates χ^2 feature map has less control on the number of features to be used. On high dimensional datasets, a high number of features must be generated. This result is consistent with the previous study that one must use very high number of Random Fourier features to achieve good accuracy (Huang et al. 2014).

Comparing IK_a with χ^2 using LIBLINEAR, IK_a produced better accuracy than χ^2 in many datasets, especially on dense datasets such as smallNORB, cifar-10, mnist, covtype and ijcnn1. This is despite the fact that IK_a employs $t = 100$ features only; and χ^2 employs a lot more features in most datasets. The epsilon dataset is the only

[7] The setting $b = 0$ is used in the experiment in this section. This appears to be the best setting for the datasets we used here. $b > 0$ not only produces poorer accuracy on some datasets, but yields a larger number of features.

[8] LIBSVM appears to perform slightly worse than LIBLINEAR in some datasets with IK_a. This is due to the different heuristics used in the optimisation procedures. Otherwise, both LIBSVM and LIBLINEAR shall yield exactly the same accuracy since the same feature map of Isolation Kernel is used.

Table 9 Kernel versus feature map using SVM for χ^2 additive kernel and Isolation Kernel. LIBLINEAR uses the feature maps; LIBSVM uses the kernels. -GE- denotes feature map generation error due to insufficient memory; and it needed more than 64GB to generate the feature map

	LIBSVM		LIBLINEAR		
	IK_a	χ^2	χ^2	IK_a	χ^2 #Features
url	.96	.98	-GE-	.96	—
news20.binary	.89	.97	.97	.92	1,355,191
rcv1.binary	.96	.96	.95	.96	47,236
real-sim	.95	.97	.97	.96	20,958
smallNORB	.88	.81	.81	.88	18,432
cifar-10	.71	> 1 week	.67	.71	3,072
epsilon	.72	> 1 week	.85	.70	2,000
minst	.98	.92	.91	.99	780
a9a	.84	.85	.85	.84	123
covtype	.93	.77	.75	.93	54
ijcnn1	.97	.90	.91	.98	22

dataset in which χ^2 appears to be significantly better than IK_a (85% vs 70%); but the accuracy of IK_a could be increased by increasing t, as shown in Fig. 7(a).

In terms of runtime speedup from LIBSVM to LIBLINEAR, both χ^2 and IK_a achieve the same orders of magnitude speedup. For example on the rcv1 dataset, both got 2 and 3 orders of magnitude speedup in training and testing, respectively. But, IK_a enables LIBSVM to run one order of magnitude faster than χ^2 on the high dimensional rcv1 dataset, i.e., 57 vs 321 seconds and 661 vs 2972 seconds in LIBSVM training and testing, respectively. Using LIBLINEAR, the comparisons are 0.3 vs 0.6 seconds and 0.2 versus 0.7 seconds in training and testing, respectively: IK_a is slightly faster but they are in the same order. The times quoted are in CPU seconds.

Because of the high number of features generated, the current version of the code[9] used to generate the χ^2 feature map was unable to generate the feature map on the url dataset of over 3 million input dimensions using a machine with 64 GBytes.

We have attempted intersection additive kernel; and it has approximately the same accuracy as the χ^2 kernel.

In summary, while the feature map of χ^2 additive kernel can be approximated well to maintain the accuracy of SVM model achieved by using the kernel, the key weakness is that the number of features cannot be controlled to a manageable number, especially in high dimensional datasets. In addition, it has inferior predictive accuracy in dense datasets in comparison with IK_a in our evaluation.

[9] At https://scikit-learn.org/stable/modules/kernel_approximation.html in Python.

9 Relation to existing approaches for efficient kernel methods

9.1 Kernel functional approximation

Kernel functional approximation is a popular effective approach to produce a user-controllable, finite-dimensional, approximate feature map of a kernel having infinite number of features.

One representative is the Nyström method (Yang et al. 2012; Williams and Seeger 2001; Wu et al. 2017; Musco and Musco 2017). It first samples $b < n$ points from the given dataset, and then constructs a matrix of low rank r, and derives a vector representation of data of r features. This gives $\mathcal{H}_N = span(\varphi_1, \ldots, \varphi_r)$, where φ is a normalised eigen function of $\sum_{i=1}^{b} K(\cdot, \mathbf{x}_i) f(\mathbf{x}_i)$. See Yang et al. (2012) for details. For $r \ll n$, it reduces the search space significantly.

The key overhead is the eigenvalue decomposition computation of the low rank matrix. This overhead is not large only if both b and r are small, relative to the data size n and dimensionality d. The overhead becomes impracticably large for problems which require large b and r.

Also, though the Nyström method depends on data when deriving an approximate feature map of a chosen nonlinear kernel, but the kernel it is approximating is still data independent (e.g., Gaussian and Laplacian kernels).

The second representative is random features method (Rahimi and Recht 2007; Yang et al. 2014). It generates a proxy of features through some transform (such as Fourier or Laplace transform) of the chosen nonlinear shift-invariant kernel function. Only a random subset of these features are used as the feature map. Note that these features are generated independent of the given dataset[10]. Let s be the number of random features generated. This gives $\mathcal{H}_R = span(\vartheta_1, \ldots, \vartheta_s)$. For $s \ll n$, it reduces the search space significantly. This method has high space complexity which requires to store a $s \times d$ matrix for random Fourier features computations; thus it is not suitable when d is high. A study reported that the number of random features needs to be in order of 200,000 to achieve acceptable accuracy in an application (Huang et al. 2014). There are faster versions, e.g., FastFood (Viet Le et al. 2013); but the improved speed often trades off accuracy.

In contrast with the Nyström method, the random features methods do not need to use a dataset or a sample in the approximation process. In other words, they need no budget. But, a comparison using OGD (Lu et al. 2016) has shown that the Fourier Random features version always produced lower accuracy than the Nyström version even when the former used four times more features than the latter in batch mode; and it could run slower in training and testing.

In a nutshell, the efficiency gain from the kernel functional approximation approach comes with the cost of reduced accuracy as it is an approximation of the chosen nonlinear kernel function.

In contrast, Isolation Kernel has an *exact* feature map, given specific parameter settings of t and ψ. This leads to efficiency gain; and its superior accuracy is due to

[10] One study has attributed the data independence of the feature generation as the reason of poorer SVM accuracy in comparison with the Nyström method (Yang et al. 2012).

the data dependency. It is a *direct* method which does not need an intervention step to approximate a feature map from a kernel having infinite or large number of features.

9.2 Sparse kernel approximation

To represent non-linearity, the feature map of a kernel has dimensionality which is usually significantly larger than the dimension of the given dataset. The Nyström method reduces the dimensionality to produce a dense representation.

In contrast, sparse kernel approximation aims to produce high-dimensional sparse features[11]. One proposal (Vedaldi and Zisserman 2012b) approximates each feature vector of $\mathbf{x}$ using a small subset of representative points, e.g., $\mathbf{x}$'s neighbours (rather than all representative points). It then uses product quantization (PQ)[12] to encode the sparse features, and employ bundle methods to learn directly from the PQ codes.

Interestingly, each feature vector of $\mathbf{x}$ of Isolation Kernel is both a sparse representation and a coding which employs exactly t representative points, from t random subsets of ψ points, i.e., exactly one out of the ψ points in one subset is used for the sparse representation, concatenated t times.

There are other sparse representations, e.g., (a) Local Deep Kernel Learning (Jose et al. 2013) learns a tree based feature embedding which is high dimensional and sparse through a generalised version of Localized Multiple Kernel Learning of multiple data independent kernels. (b) Concomitant Rank Order (CRO) kernel (Kafai and Eshghi 2019) approximates the Gaussian kernel on the unit sphere. It uses Discrete Cosine Transform to compute the random projection of CRO feature map which can produce feature vectors efficiently.

The key difference between Isolation Kernel and current sparse kernel approximation is that the former is a *data dependent* kernel (Ting et al. 2018; Qin et al. 2019) which has an *exact* feature map. Sparse kernel approximation may be viewed as another intervention step (alternative to kernel functional approximation) to produce a finite-dimensional sparse *approximate* feature map from one or more *data independent* kernels having infinite number of features. In addition, computationally expensive learning (Jose et al. 2013) or PQ (Vedaldi and Zisserman 2012b) are not required in Isolation Kernel.

10 Discussion

It is important to note that Isolation Kernel is not one kernel function such as Gaussian kernel, but a class of kernels which has different kernel distributions depending on the space partitioning mechanism employed. We use two implementations of Isolation Kernel: (a) iForest (Liu et al. 2008) which has its kernel distribution similar to that of Laplacian Kernel under uniform density distribution (Ting et al. 2018); (b) when a

[11] A sparse representation yields vectors having many zero values, where a feature with zero value means that the feature is irrelevant.

[12] Product Quantization (an improvement over vector quantization) aims to reduce storage and retrieval time for conducting approximate nearest neighbour search.

Voronoi diagram is used to partition the space (Qin et al. 2019), Isolation Kernel has its distribution more akin to an exponential kernel under uniform density distribution. Both realisations of Isolation Kernel adapt to local density of a given dataset, unlike existing data independent kernels. The criterion required of a partitioning mechanism in order to produce an effective Isolation Kernel is described in Ting et al. (2018); Qin et al. (2019). This paper has focused on efficient implementations of Isolation Kernel in online kernel learning, without compromising accuracy.

We have the view that the data samples used to derive Isolation Kernel do not need to be obtained via a method other than random sampling, especially that involving a computationally expensive process. Note that the data dependent characteristic of Isolation Kernel, mentioned in Sect. 4.3, relies on the isolating partitions being large in sparse regions and small in dense regions (Ting et al. 2018). A data sample, which does not allow an isolation mechanism to produce partitions that reflect the underlying data distribution, does not yield an Isolation Kernel. If one wants to produce a different kind of data dependent kernel, then its altered data dependent characteristic needs to be established in order to explain why such a data dependent kernel will work in practice.

When using linear kernel, the trick of using ${}_{primal}f$ instead of ${}_{dual}f$ to speed up the runtime of both the training stage and the testing stage has been applied previously, e.g., in LIBLINEAR (Fan et al. 2008), even when it is solving the dual optimisation problem. This is possible in LIBLINEAR because linear kernel has an exact and finite-dimensional feature map. But, if you are using an existing nonlinear kernel such as Gaussian or Laplacian kernel, such a trick cannot be applied to SVM because its feature map is not finite.

Geurts et al. (2006) describe a kernel view of Extra-Trees (a variant of Random Forest (Breiman 2001)) where its feature map is also sparse and similar to the one we presented here. However, like Random Forest (RF) kernel (Breiman 2000), this kernel was offered as a view point to explain the behaviour of Random Forest; and no evaluation has been conducted to assess its efficacy using a kernel-based method. Ting et al. (2018) have provided the conceptual differences between RF-like kernels and Isolation Kernel; and their empirical evaluation has revealed that RF-like kernels are inferior to Isolation Kernel when used in SVM.

11 Concluding remarks

We began our investigation in questioning the kernel often used in current approaches to large scale online kernel learning, i.e., the kernel has a feature map with intractable dimensionality. While most existing kernels have such a feature map, we reveal that there is one recent kernel called Isolation Kernel that has a *sparse and finite-dimensional* feature map.

The new feature map of Isolation Kernel becomes the heart of the proposed approach to large scale online kernel learning. Using this new approach with Isolation Kernel, we show that large scale online kernel learning can be achieved efficiently without sacrificing accuracy. It has enabled kernel learning to achieve the outcome which has evaded current approaches thus far, i.e., to live up to its full potential in online setting with large scale high dimensional sparse and dense datasets.

Isolation Kernel's sparse and finite-dimensional feature map is the crucial factor that brings about this outcome. Specifically, the proposed feature map enables three key elements: (1) kernel learning with the finite-dimensional feature map; (2) sparse representation enables efficient dot product; and (3) the Voronoi diagram implementation of Isolation Kernel is amenable to GPU acceleration.

The proposed approach is generic in two aspects. First, it is not restricted to Isolation Kernel only. Potentially, any data dependent kernel which has a sparse and finite-dimensional feature map can use this approach. Second, even restricting to Isolation Kernel only, as long as a new space partitioning mechanism that can produce a feature map of similar property, it can use this approach as well.

Like other kernels, Isolation Kernel is a generic similarity measure that can be used to solve different problems (e.g., graph (Xu et al. 2021), anomaly detection (Ting et al. 2020), multi-instance learning (Xu et al. 2019) and density-based clustering (Qin et al. 2019)). Unlike existing kernels, this study suggests that Isolation Kernel has the potential to deal with millions of dimensions effectively. Future work includes examining Isolation Kernel's ability to deal with the curse of dimensionality.

Acknowledgements Kai Ming Ting is supported by Natural Science Foundation of China (62076120). Takashi Washio is supported by JST CREST Grant Number JPMJCR1666 and the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number 20K21815. A discussion with Jaakko Peltonen at Shonan Meeting in October 2018 has provided the initial impetus in creating the feature map of Isolation Kernel described in this paper.

References

Breiman L (2000) Some infinity theory for predictor ensembles. Tech Rep 577, Statistics Dept, University of California, Berkeley

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Cavallanti G, Cesa-Bianchi N, Gentile C (2007) Tracking the best hyperplane with a simple budget perceptron. Mach Learn 69(2):143–167

Chang CC, Lin CJ (2011) LIBSVM: A library for support vector machines. ACM Transactions Intell Syst Technol 2(3):1–27

Chang YW, Hsieh CJ, Chang KW, Ringgaard M, Lin CJ (2010) Training and testing low-degree polynomial data mappings via linear svm. J Mach Learn Res 11:1471–1490

Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) Liblinear: a library for large linear classification. J Mach Learn Res 9:1871–1874

Felix XY, Suresh AT, Choromanski KM, Holtmann-Rice DN, Kumar S (2016) Orthogonal random features. In: Advances in Neural Information Processing Systems, pp 1975–1983

Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42

Huang P, Avron H, Sainath TN, Sindhwani V, Ramabhadran B (2014) Kernel methods match deep neural networks on timit. In: Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing, pp 205–209

Jose C, Goyal P, Aggrwal P, Varma M (2013) Local deep kernel learning for efficient non-linear svm prediction. In: Proceedings of the 30th International Conference on Machine Learning, pp III–486–III–494

Kafai M, Eshghi K (2019) Croification: accurate kernel classification with the efficiency of sparse linear svm. IEEE Transactions Pattern Anal Mach Intell 41(1):34–48

Kivinen J, Smola AJ, Williamson RC (2001) Online learning with kernels. In: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, pp 785–792

Liu FT, Ting KM, Zhou ZH (2008) Isolation forest. In: Proceedings of the IEEE International Conference on Data Mining, pp 413–422
Lu J, Hoi SCH, Wang J, Zhao P, Liu ZY (2016) Large scale online kernel learning. J Mach Learn Res 17(1):1613–1655
Maji S, Berg AC, Malik J (2013) Efficient classification for additive kernel svms. IEEE Transactions Pattern Anal Mach Intell 35(1):66–77
Musco C, Musco C (2017) Recursive sampling for the nystrom method. In: Advances in Neural Information Processing Systems, pp 3833–3845
Qin X, Ting KM, Zhu Y, Lee VCS (2019) Nearest-neighbour-induced isolation similarity and its impact on density-based clustering. In: Proceedings of The Thirty-Third AAAI Conference on Artificial Intelligence, pp 4755–4762
Rahimi A, Recht B (2007) Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems, pp 1177–1184
Rakotomamonjy A, Bach FR, Canu S, Grandvalet Y (2008) SimpleMKL. J Mach Learn Res 9(Nov):2491–2521
Scholkopf B, Smola AJ (2001) Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT Press, Cambridge
Shen Y, Chen T, Giannakis G (2018) Online ensemble multi-kernel learning adaptive to non-stationary and adversarial environments. In: Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics, pp 2037–2046
Ting KM, Zhu Y, Zhou ZH (2018) Isolation kernel and its effect on SVM. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 2329–2337
Ting KM, Xu BC, Washio T, Zhou ZH (2020) Isolation distributional kernel: a new tool for kernel based anomaly detection. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 198–206
Vedaldi A, Zisserman A (2012a) Efficient additive kernels via explicit feature maps. IEEE Transactions Pattern Anal Mach Intell 34(3):480–492
Vedaldi A, Zisserman A (2012b) Sparse kernel approximations for efficient classification and detection. In: Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp 2320–2327
Viet Le Q, Sarlos T, Smola AJ (2013) Fastfood: approximate kernel expansions in loglinear time. In: Proceedings of the 30th International Conference on Machine Learning, pp III–244–III–252
Wang Z, Vucetic S (2010) Online passive-aggressive algorithms on a budget. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp 908–915
Wang Z, Crammer K, Vucetic S (2012) Breaking the curse of kernelization: budgeted stochastic gradient descent for large-scale svm training. J Mach Learn Res 13(1):3103–3131
Williams CKI, Seeger M (2001) Using the nyström method to speed up kernel machines. In: Advances in Neural Information Processing Systems 13. MIT Press, Cambridge, pp 682–688
Wu J, Ding L, Liao S (2017) Predictive nyström method for kernel methods. Neurocomputing 234:116–125
Xu BC, Ting KM, Zhou ZH (2019) Isolation set-kernel and its application to multi-instance learning. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 941–949
Xu BC, Ting KM, Jiang Y (2021) Isolation graph kernel. In: Proceedings of The Thirty-Fifth AAAI Conference on Artificial Intelligence, pp 10487–10495
Yang J, Sindhwani V, Fan Q, Avron H, Mahoney M (2014) Random Laplace feature maps for semigroup kernels on histograms. In: Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp 971–978
Yang T, Li YF, Mahdavi M, Jin R, Zhou ZH (2012) Nyström method vs random fourier features: a theoretical and empirical comparison. In: Advances in Neural Information Processing Systems, pp 476–484
Zadeh P, Hosseini R, Sra S (2016) Geometric mean metric learning. In: Proceedings of the International Conference on Machine Learning, pp 2464–2471