PLAME: Piecewise-Linear Approximate Measure for Additive Kernel SVM

Tsz Nam Chan, *Member, IEEE*, Zhe Li, Leong Hou U, *Member, IEEE*, Reynold Cheng, *Member, IEEE*

Abstract—Additive Kernel SVM has been extensively used in many applications, including human activity detection and pedestrian detection. Since training an additive kernel SVM model is very time-consuming, which is not scalable to large-scale datasets, many efficient solutions have been developed in the past few years. However, most of the existing methods normally fail to achieve one of these three important conditions which are (1) low classification error, (2) low memory space, and (3) low training time. In order to simultaneously fulfill these three conditions, we develop the new piecewise-linear approximate measure (PLAME) for additive kernels. By incorporating PLAME with the well-known dual coordinate descent method, we theoretically show that this approach can achieve the above three conditions. Experimental results on twelve real datasets show that our approach can achieve the best trade-off between the accuracy, memory space, and training time compared with different types of state-of-the-art methods.

Index Terms—PLAME, additive kernels, SVM

1 Introduction

Kernel methods [55] are the commonly used techniques in classification tasks [12], [29], [30], [32], [34], [55], [64], which have also received significant attention in both database [10]–[12], [24], [65], [68], [77] and data mining [30], [50], [58], [72], [73] communities in the last few years. Recently, additive kernels have been extensively studied in different communities, e.g., machine learning [33], [46], [47], [70], [75], computer vision [35], [43]–[45], [59], [64], and database [11] due to a wide range of applications. Human activity detection systems [27], [37], [40], [41], [51], [52], [63], [79] utilize SVM models with additive kernels to predict human activities, e.g., walk and sit down, from the sensor data. Pedestrian detection systems [4], [5], [45], [64] utilize SVM models with additive kernels to detect pedestrians in an image. Geoscientists [8], [21] utilize SVM models with additive kernels to characterize spatial properties of objects in a scene. Medical scientists [36], [78] utilize SVM models with additive kernels to identify different types of cancer (e.g., colorectal cancer [36] and melanoma skin cancer [78]) in an image. In above studies, the scholars believe that using SVM models with additive kernels (e.g., χ^2 kernel) can generally provide superior performance in their applications. Some representative examples of recent studies are quoted as follows:

- *"... empirical studies have demonstrated the superiority of the chi2 kernel for image classification ..."* [78]
- *"... we train a SVM classifier with chi-square kernel for multi-class recognition task, which is beneficial for classifying the histogram features."* [27]
- *"... by adding non-linear additive kernels whose effectiveness have been validated in computer vision ..."* [80]

However, due to the nonlinearity of additive kernel functions, it is time-consuming to train SVM models with this type of kernel functions, especially for large-scale datasets. Therefore, many research studies also complain about the inefficiency issue for using these nonlinear kernel functions.

- *"... it was necessary to use support vector machine (SVM) classifier with nonlinear χ^2 kernel which resulted in high computational cost."* [54]
- *"... a dataset with half a million training examples and each of which consists of thousands of features might take days to train with any nonlinear kernels, not to mention using billion or trillion features."* [64]
- *"... non-linear SVMs $f(\mathbf{x};\alpha) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x_i}, \mathbf{x})$ are expanded in term of evaluations of a non-linear kernel function $K(\mathbf{x}, \mathbf{x}')$ and are much slower to compute as well as train."* [60]

Compared with training SVM models using additive kernels, it is computationally efficient for training linear SVM models by using some existing libraries, e.g., LIBLINEAR [23] and Pegasos [56]. However, linear SVM models normally provide inferior accuracy results [47], [64].

To achieve the efficient and accurate classification with additive kernels, most of the existing studies [33], [35], [43], [47], [59], [64] utilize the feature approximation techniques, which construct the new and higher dimensional feature vectors (e.g., $\psi(\mathbf{q})$ and $\psi(\mathbf{p})$) for the original feature vectors (e.g., $\mathbf{q}$ and $\mathbf{p}$), in order to approximate the kernel function $K(\mathbf{q}, \mathbf{p})$, i.e.,

$$\psi(\mathbf{q})^T \psi(\mathbf{p}) \approx K(\mathbf{q}, \mathbf{p}) \quad (1)$$

Once they obtain these high-dimensional feature vectors (e.g., $\psi(\mathbf{q})$ and $\psi(\mathbf{p})$), they train linear SVM models, which

- *T. N. Chan is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong.*
  E-mail: edisonchan@comp.hkbu.edu.hk
- *Z. Li is with the Alibaba Cloud, Hangzhou.*
  E-mail: huoju.lz@alibaba-inc.com
- *L. H. U is with the Department of Computing and Information Science, University of Macau, Macau.*
  E-mail: ryanlhu@um.edu.mo
- *R. Cheng is with the Department of Computer Science, The University of Hong Kong, Hong Kong.*

can be more efficient compared with training nonlinear SVM models.

However, there are two main issues for using the feature approximation methods. First, most of these feature approximation methods produce high-dimensional feature vectors (e.g., 9x larger compared with the original feature vectors) in order to achieve better accuracy, which significantly consume more memory resources. Given the human activity recognition (casas) dataset [1], [19] with 12.6M training data points and 37 dimensions (i.e., 3.46GB), the size of the dataset with new feature vectors can be 31.1GB. Second, based on our experimental observations, these methods may not achieve competitive accuracy compared with original additive kernel SVM models.

To avoid using huge memory resources, another type of research studies [69], [71], [75] focuses on using a simple quadratic function to approximate the kernel aggregation function $F(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$ [12], where α_i and $(\mathbf{x}_i, y_i)$ denote the i^{th} constant value and the i^{th} point-label pair, respectively (which will be discussed in detail in Section 2). In these studies, they show that, by replacing $F(\mathbf{x})$ with a quadratic function, they can train a SVM model efficiently by adopting some simple modifications of the dual coordinate descent method [28] (adopted in the software LIBLINEAR [23]). Unlike the feature approximation approach, these studies do not need a huge amount of additional space. However, the kernel aggregation function $F(\mathbf{x})$ may not necessarily follow the form of quadratic function, which significantly degrades the training accuracy.

Observe that all types of methods cannot simultaneously fulfill these three conditions: (1) low classification error (high accuracy), (2) low memory consumption, and (3) low training time (cf. Table 1). As such, we develop the new similarity measure, called Piecewise-Linear Approximate MEasure (PLAME), which directly utilizes the piecewise-linear function to approximate the additive kernel function $K(\mathbf{x}_i, \mathbf{x})$. By adopting some simple modifications of the dual coordinate descent method [28], we show that this algorithm can train SVM models with PLAME with slight time overhead. In addition, we show that this new method only incurs a few additional space. Furthermore, we also ensure that the approximation error between PLAME and $K(\mathbf{x}_i, \mathbf{x})$ can be theoretically close which results in low classification error.

TABLE 1: Different methods for training SVM with additive kernels.

Method	Classification error	Memory space	Training time
Kernel SVM solver [13]	low	low	high
Linear SVM solver [23], [28]	high	low	low
Feature approximation [35], [47], [59], [60] [7], [17], [33], [64]	high	high	low
Function approximation [69], [71], [75]	high	low	low
PLAME (ours)	low	low	low

The rest of the paper is organized as follows. We first review the background of linear SVM, kernel SVM, and additive kernels in Section 2. Then, we define our new similarity measure PLAME and illustrate how to utilize PLAME to train SVM models with theoretical analysis in Section 3. Next, we present experimental results on twelve real datasets in Section 4. After that, we review existing work in Section 5. Lastly, we conclude in Section 6. The appendix can be found in Section 7.

2 Preliminaries

In this section, we first review the fundamental concept of linear SVM classification and revisit the famous algorithm, i.e., dual coordinate descent algorithm [28], in Section 2.1. Then, we illustrate the concept of kernel SVM in Section 2.2. Lastly, we also review the additive kernels in Section 2.3. Table 2 summarizes the commonly used symbols in this paper.

TABLE 2: Symbols.

Symbol	Description
n	Number of data points in the training dataset
d	Dimensionality of data points
$F(\mathbf{x})$	Kernel aggregation function
$K(\mathbf{x}_i, \mathbf{x})$	Kernel function
$\mathbf{x}_i$	The i^{th} vector in the training dataset
$\mathbf{x}$	Any vector in the testing dataset
$x_i^{(\ell)}$	The ℓ^{th} dimensional value of $\mathbf{x}_i$
$x^{(\ell)}$	The ℓ^{th} dimensional value of $\mathbf{x}$
$k(x_i^{(\ell)}, x^{(\ell)})$	One-dimensional kernel function
$k_{PL}(x_i^{(\ell)}, x^{(\ell)})$	Piecewise-linear approximate measure (PLAME) for $k(x_i^{(\ell)}, x^{(\ell)})$
$f(x^{(\ell)})$	One-dimensional kernel aggregation function
$f_{PL}(x^{(\ell)})$	Approximation of $f(x^{(\ell)})$
P	Number of intervals in $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$
$\mathcal{I}$	Set of intervals in $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$
I_p	The p^{th} interval in $\mathcal{I}$ (with the range $[l_p, u_p]$)
$m_{x^{(\ell)}}(I_p)$	The slope of $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$ in the p^{th} interval I_p
$c_{x^{(\ell)}}(I_p)$	The intercept of $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$ in the p^{th} interval I_p
ϵ	The error parameter
$\mathbb{E}(l, u)$	The maximum error between $k(x_i^{(\ell)}, x^{(\ell)})$ and $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$ with any interval $I = [l, u]$

2.1 Linear SVM classification

Given the set, with size n, of point-label pairs $(\mathbf{x}_i, y_i)$, where each $\mathbf{x}_i$ is the d-dimensional vector and each y_i is either 1 or -1 to denote the positive or negative classes, respectively, the goal of linear SVM classification [55] is to find the linear function $y = \mathbf{w}^T\mathbf{x} + b$ which can separate these data points into two classes (cf. Figure 1).

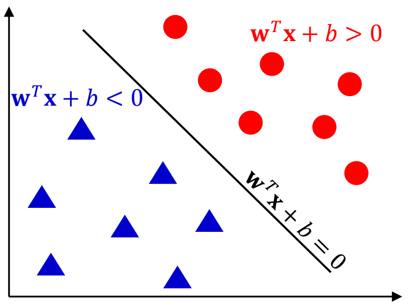

$\mathbf{w}^T\mathbf{x} + b > 0$
$\mathbf{w}^T\mathbf{x} + b < 0$
$\mathbf{w}^T\mathbf{x} + b = 0$

Fig. 1: Linear SVM classification: using the linear line $\mathbf{w}^T\mathbf{x} + b = 0$ to classify the data points into two classes (red circle, $\mathbf{w}^T\mathbf{x} + b > 0$, and blue triangle, $\mathbf{w}^T\mathbf{x} + b < 0$).

To find the best linear function, we need to solve the following primal (optimization) problem, where C is the penalty parameter [55].

$$\begin{aligned} \underset{\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\zeta_i \\ \text{such that} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \zeta_i \text{ where } i = 1..n \\ & \zeta_i \geq 0 \text{ where } i = 1..n \end{aligned} \tag{P}$$

However, instead of solving the primal problem, most of the SVM classification algorithms, including dual coordinate descent [28] and sequential minimal optimization (SMO) methods [55], solve the equivalent dual form of this problem which aims to find the n-dimensional vector $\boldsymbol{\alpha} = [\alpha_1\ \alpha_2\ \cdots\ \alpha_n]^T$.

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \\ \text{such that} \quad & 0 \leq \alpha_i \leq C \text{ where } i = 1,...,n \end{aligned} \tag{D}$$

where $\mathbf{1}$ is the n-dimensional vector with all entries one and Q is the $n \times n$ matrix with $Q_{ij} = y_i\mathbf{x}_i^T\mathbf{x}_j y_j$.

Based on this vector $\boldsymbol{\alpha}$ (cf. Equations 2 and 3), they can obtain the parameters $\mathbf{w}$ and b of linear function [55].

$$\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i \tag{2}$$

$$b = \bar{y} - \mathbf{w}^T\bar{\mathbf{x}} \tag{3}$$

where the corresponding $\bar{\alpha}$ for $(\bar{\mathbf{x}}, \bar{y})$ fulfills $0 < \bar{\alpha} < C$.

To solve the linear SVM classification problem, existing software LIBLINEAR [28] adopts the dual coordinate descent algorithm (cf. Algorithm 1) to solve the optimization problem (D).

Algorithm 1 Dual Coordinate Descent Algorithm [28]

```
1:  procedure DCD($\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}, C$)
2:      $\boldsymbol{\alpha} \leftarrow \mathbf{0}, \mathbf{w} \leftarrow \mathbf{0}$
3:      while $\boldsymbol{\alpha}$ is not optimal do
4:          for $v \leftarrow 1$ to $n$ do
5:              $G \leftarrow y_v\mathbf{w}^T\mathbf{x}_v - 1$
6:              $T \leftarrow \alpha_v$
7:              $\alpha_v \leftarrow \min(\max(\alpha_v - \frac{G}{Q_{vv}}, 0), C)$
8:              $\mathbf{w} \leftarrow \mathbf{w} + (\alpha_v - T)y_v\mathbf{x}_v$
9:      Find $\bar{\alpha}$ in $\boldsymbol{\alpha}$ such that $0 < \bar{\alpha} < C$
10:     $b \leftarrow \bar{y} - \mathbf{w}^T\bar{\mathbf{x}}$                ▷ Equation 3
11:     return $\mathbf{w}$ and $b$
```

In Algorithm 1, the bottlenecks of this algorithm are to compute the aggregation operation $\mathbf{w}^T\mathbf{x}_v$ (line 5) and to update the vector $\mathbf{w}$ (line 8) which take $O(d)$ time in each inner iteration (cf. line 4). Therefore, each outer iteration (line 3) takes $O(nd)$ time.

2.2 Kernel SVM classification

Even though linear SVM classification can be efficient in the training phase, the accuracy result is normally inferior as the intrinsic structure of these datasets can be complex which cannot be simply separated by any linear functions (cf. Figure 2).

To handle nonlinear SVM classification, existing studies [13], [55] replace each vector $\mathbf{x}$ with the high dimensional (possibly infinite-dimensional) vector $\phi(\mathbf{x})$ (cf. Figure 2), where:

$$K(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}) \tag{4}$$

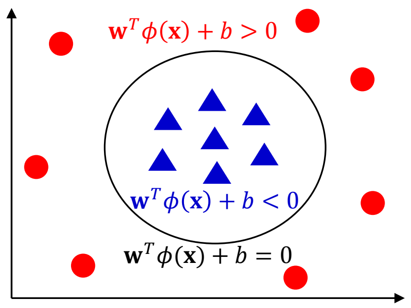

$\mathbf{w}^T\phi(\mathbf{x}) + b > 0$
$\mathbf{w}^T\phi(\mathbf{x}) + b < 0$
$\mathbf{w}^T\phi(\mathbf{x}) + b = 0$

Fig. 2: Kernel SVM classification: using the nonlinear function $\mathbf{w}^T\phi(\mathbf{x}) + b = 0$ to classify the data points into two classes (red circle, $\mathbf{w}^T\phi(\mathbf{x}) + b > 0$, and blue triangle, $\mathbf{w}^T\phi(\mathbf{x}) + b < 0$).

We call $K(\mathbf{x}_i, \mathbf{x})$ as the kernel function. Even though the vector $\phi(\mathbf{x}_i)$ can be possibly infinite-dimensional [55], the kernel function can be efficiently computed (e.g., $O(d)$ time for additive kernels).

With the concept of kernel function (cf. Equation 4), we can replace $Q_{ij} = y_i\mathbf{x}_i^T\mathbf{x}_j y_j$ by $y_i K(\mathbf{x}_i, \mathbf{x}_j)y_j$ in the optimization problem (D). However, observe from Equation 5 that $\mathbf{w}$ depends on the possibly infinite-dimensional vector $\phi(\mathbf{x}_i)$. We cannot maintain the explicit format of $\mathbf{w}$.

$$\mathbf{w} = \sum_{i=1}^{n}\alpha_i y_i\phi(\mathbf{x}_i) \tag{5}$$

Nevertheless, we can still reuse Algorithm 1, with slight modifications, to obtain the best $\boldsymbol{\alpha}$ for kernel SVM classification. The reason is the main operation, gradient computation, in Algorithm 1 (cf. line 5) only depends on $\mathbf{w}^T\phi(\mathbf{x}_v)$ (in kernel SVM). Therefore, we can replace it by the following kernel aggregation function (cf. Equation 6).

$$F(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_{i=1}^{n}\alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) \tag{6}$$

In order to support kernel SVM classification, we only need to modify Algorithm 1 by replacing line 5 with $G \leftarrow y_v F(\mathbf{x}_v) - 1$, removing line 8, and replacing b (line 10, i.e., Equation 3) by:

$$b = \bar{y} - F(\bar{\mathbf{x}}) \tag{7}$$

The pseudocode, based on the above modifications, is described in Algorithm 2.

Algorithm 2 Dual Coordinate Descent Algorithm (Kernel Version)

```
1:  procedure DCD_KERNEL($\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}, C$)
2:      $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
3:      while $\boldsymbol{\alpha}$ is not optimal do
4:          for $v \leftarrow 1$ to $n$ do
5:              $G \leftarrow y_v F(\mathbf{x}_v) - 1$
6:              $T \leftarrow \alpha_v$
7:              $\alpha_v \leftarrow \min(\max(\alpha_v - \frac{G}{Q_{vv}}, 0), C)$
8:      Find $\bar{\alpha}$ in $\boldsymbol{\alpha}$ such that $0 < \bar{\alpha} < C$
9:      $b \leftarrow \bar{y} - F(\bar{\mathbf{x}})$                ▷ Equation 7
10:     return $\boldsymbol{\alpha}$ and $b$
```

However, since the evaluation of kernel aggregation function (cf. Equation 6) takes $O(nd)$ time, computing one outer iteration (line 3 in Algorithm 2) takes $O(n^2d)$ time which is infeasible for medium to large-scale datasets (e.g., $n = 100000$). As such, existing solution for kernel SVM classification is still based on SMO method [55] (used in LIBSVM [13]), which is still slow in practice.

2.3 Additive Kernels

Additive kernels have been successfully used in both computer vision [44], [45], [59], [64] and machine learning [22], [46], [47], [70] communities. Given $\mathbf{x}_i$ and $\mathbf{x}$ as two d-dimensional vectors, where $x_i^{(\ell)}$ and $x^{(\ell)}$ denote the ℓ^{th} dimensional values of these two vectors, respectively, Table 3 summarizes four famous additive kernel functions, which are χ^2, JS, intersection, and Hellinger kernels.

TABLE 3: Four representative additive kernel functions.

Kernel name	$K(\mathbf{x}_i, \mathbf{x})$
χ^2	$\sum_{\ell=1}^{d} \frac{2x_i^{(\ell)}x^{(\ell)}}{x_i^{(\ell)}+x^{(\ell)}}$
JS	$\sum_{\ell=1}^{d} \frac{1}{2}x_i^{(\ell)} \log_2\left(\frac{x_i^{(\ell)}+x^{(\ell)}}{x_i^{(\ell)}}\right) + \frac{1}{2}x^{(\ell)} \log_2\left(\frac{x_i^{(\ell)}+x^{(\ell)}}{x^{(\ell)}}\right)$
Intersection	$\sum_{\ell=1}^{d} \min(x_i^{(\ell)}, x^{(\ell)})$
Hellinger	$\sum_{\ell=1}^{d} \sqrt{x_i^{(\ell)}x^{(\ell)}}$

Observe that these kernel functions exhibit the following additive property [11], [45] (cf. Definition 1).

Definition 1. *If the kernel function $K(\mathbf{x}_i, \mathbf{x})$ is the sum of d one-dimensional kernel functions (denoted as $k(x_i^{(\ell)}, x^{(\ell)})$), $K(\mathbf{x}_i, \mathbf{x})$ is the additive kernel function.*

$$K(\mathbf{x}_i, \mathbf{x}) = \sum_{\ell=1}^{d} k(x_i^{(\ell)}, x^{(\ell)}) \tag{8}$$

As an example, $k(x_i^{(\ell)}, x^{(\ell)}) = \frac{2x_i^{(\ell)}x^{(\ell)}}{x_i^{(\ell)}+x^{(\ell)}}$ for χ^2 kernel function. Based on Definition 1, we can decompose the kernel aggregation function (cf. Equation 6) into the addition of d one-dimensional kernel aggregation functions (cf. Lemma 1). The proof of this lemma can be found from [11], [45].

Lemma 1. *Let $f(x^{(\ell)})$ be the one-dimensional kernel aggregation function, where:*

$$f(x^{(\ell)}) = \sum_{i=1}^{n} \alpha_i y_i k(x_i^{(\ell)}, x^{(\ell)}) \tag{9}$$

Then, we can express $F(\mathbf{x})$ as:

$$F(\mathbf{x}) = \sum_{\ell=1}^{d} f(x^{(\ell)}) \tag{10}$$

Recall from Section 2.2 that the bottleneck of the training phase for kernel SVM classification is the computation of the kernel aggregation function, e.g., evaluating the gradient (line 5 in Algorithm 2) in the training phase. As such, if we can efficiently evaluate Equation 10, we can significantly boost the efficiency performance for training additive kernel SVM models. Since Equation 10 is the addition of each $f(x^{(\ell)})$, which is independent for each dimension ℓ, our goal is to develop fast method for computing $f(x^{(\ell)})$ (cf. Equation 9).

3 PLAME: Piecewise-Linear Approximate Measure

To provide the fast evaluation of the one-dimensional kernel aggregation function $f(x^{(\ell)})$, we propose a novel approximate measure, which is $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$, to replace $k(x_i^{(\ell)}, x^{(\ell)})$ in Equation 9. As a remark, we denote $x_i^{(\ell)}$ as the ℓ^{th} dimensional value of the i^{th} training data. First, after we know the region, e.g., $[L(\ell), U(\ell)]$[1], of $x_i^{(\ell)}$ for each dimension, we can partition this region into P intervals $\{I_1, I_2, ..., I_P\}$, where $I_p = [l_p, u_p]$ $(1 \leq p \leq P)$. Then, we can define this piecewise-linear approximate measure (PLAME) based on these intervals (cf. Equation 11).

$$k_{PL}(x_i^{(\ell)}, x^{(\ell)}) = \begin{cases} m_{x^{(\ell)}}(I_1)x_i^{(\ell)} + c_{x^{(\ell)}}(I_1) & \text{if } x_i^{(\ell)} \in I_1 \\ m_{x^{(\ell)}}(I_2)x_i^{(\ell)} + c_{x^{(\ell)}}(I_2) & \text{if } x_i^{(\ell)} \in I_2 \\ \vdots & \vdots \\ m_{x^{(\ell)}}(I_P)x_i^{(\ell)} + c_{x^{(\ell)}}(I_P) & \text{if } x_i^{(\ell)} \in I_P \end{cases} \tag{11}$$

where the slope $m_{x^{(\ell)}}(I_p)$ and the intercept $c_{x^{(\ell)}}(I_p)$ are denoted as:

$$m_{x^{(\ell)}}(I_p) = \frac{2{x^{(\ell)}}^2}{(x^{(\ell)} + u_p)(x^{(\ell)} + l_p)} \tag{12}$$

$$c_{x^{(\ell)}}(I_p) = \frac{2x^{(\ell)} l_p u_p}{(x^{(\ell)} + u_p)(x^{(\ell)} + l_p)} \tag{13}$$

We utilize χ^2-kernel as an example, i.e., $k(x_i^{(\ell)}, x^{(\ell)}) = \frac{2x_i^{(\ell)}x^{(\ell)}}{x_i^{(\ell)}+x^{(\ell)}}$. Figure 3 shows one possible choice of PLAME where we partition this curve with three intervals ($P = 3$), i.e., $I_1 = [0, 0.2]$, $I_2 = [0.2, 0.4]$ and $I_3 = [0.4, 1]$.

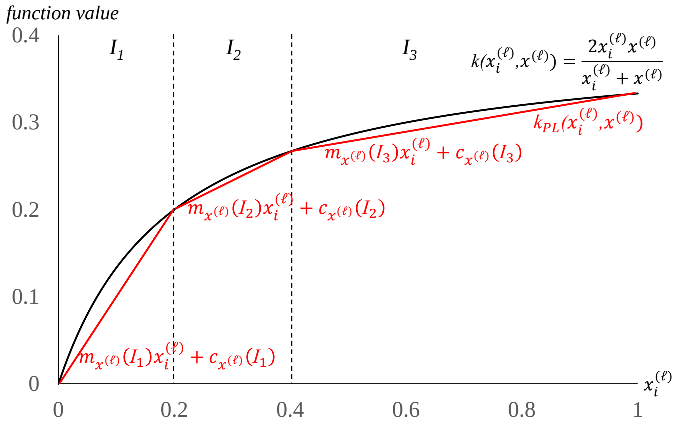


Fig. 3: PLAME for χ^2 kernel, where $L(\ell) = 0$ and $U(\ell) = 1$.

Since χ^2 kernel is the most famous one, which is frequently used in many recent research studies (e.g., [27],

1. Since we need to perform normalization before training SVM classifier [13], the region for each dimension is usually $[0, 1]$.

[78]), among these four additive kernels, we mainly focus on χ^2 kernel in this paper. Like Figure 3, we can also utilize PLAME (cf. Equation 11) to approximate other additive kernel functions (cf. Table 3). Since the concept is similar, we omit the details in this paper.

3.1 Efficient Training with PLAME

With the concept of PLAME (cf. Equation 11), we can replace the additive kernel function from $K(\mathbf{x}_i, \mathbf{x})$ (cf. Equation 8) to $K_{PL}(\mathbf{x}_i, \mathbf{x})$ (cf. Equation 14) and the one-dimensional kernel aggregation function from $f(x^{(\ell)})$ (cf. Equation 9) to $f_{PL}(x^{(\ell)})$ (cf. Equation 15).

$$K_{PL}(\mathbf{x}_i, \mathbf{x}) = \sum_{\ell=1}^{d} k_{PL}(x_i^{(\ell)}, x^{(\ell)}) \tag{14}$$

$$f_{PL}(x^{(\ell)}) = \sum_{i=1}^{n} \alpha_i y_i k_{PL}(x_i^{(\ell)}, x^{(\ell)}) \tag{15}$$

We state that $f_{PL}(x^{(\ell)})$ can be evaluated in $O(P)$ time (cf. Lemma 2), which can be significantly faster than the evaluation of $f(x^{(\ell)})$ (i.e., $O(n)$ time) if $P << n$.

Lemma 2. *$f_{PL}(x^{(\ell)})$ can be computed in $O(P)$ time if $A_\ell(I_p)$ and $B_\ell(I_p)$ are stored in advance, where:*

$$A_\ell(I_p) = \sum_{x_i^{(\ell)} \in I_p} \alpha_i y_i x_i^{(\ell)} \tag{16}$$

$$B_\ell(I_p) = \sum_{x_i^{(\ell)} \in I_p} \alpha_i y_i \tag{17}$$

Proof. Since each $x_i^{(\ell)}$ belongs to only one interval, we can express this equation as:

$$\begin{aligned} f_{PL}(x^{(\ell)}) &= \sum_{p=1}^{P} \sum_{x_i^{(\ell)} \in I_p} \alpha_i y_i (m_{x^{(\ell)}}(I_p) x_i^{(\ell)} + c_{x^{(\ell)}}(I_p)) \\ &= \sum_{p=1}^{P} m_{x^{(\ell)}}(I_p) A_\ell(I_p) + \sum_{p=1}^{P} c_{x^{(\ell)}}(I_p) B_\ell(I_p) \end{aligned}$$

Since $A_\ell(I_p)$ and $B_\ell(I_p)$ are independent of $x^{(\ell)}$, we can compute $f_{PL}(x^{(\ell)})$ in $O(P)$ time once we have stored $A_\ell(I_p)$ and $B_\ell(I_p)$ in advance. □

In Lemma 2, we have stated that $f_{PL}(x^{(\ell)})$ can be computed in $O(P)$ time once $A_\ell(I_p)$ and $B_\ell(I_p)$ can be stored in advance. However, during the training process, the parameters α_i can be updated in each iteration. Therefore, we also need to efficiently maintain the parameters $A_\ell(I_p)$ and $B_\ell(I_p)$ for each dimension ℓ. Fortunately, dual coordinate descent method only updates one α_i in each iteration. We can therefore update all $A_\ell(I_p)$ and $B_\ell(I_p)$ in $O(Pd)$ time, using the similar concept of line 8 in Algorithm 1. Algorithm 3 shows the revised dual coordinate descent method for handling PLAME. We further show that each outer iteration (line 6) of Algorithm 3 takes $O(nPd)$ time in Theorem 1. In practice, we have $P << n$, which will be discussed in detail from Section 3.2 to Section 3.4. Therefore, Algorithm 3 is significantly faster than Algorithm 2 (with $O(n^2 d)$ time) and slightly slower than the linear SVM solver (with $O(nd)$ time), i.e., Algorithm 1.

Algorithm 3 Dual Coordinate Descent Algorithm for PLAME

```
1:  procedure DCD_PLAME({(x_1, y_1), ..., (x_n, y_n)}, C)
2:      $\boldsymbol{\alpha} \leftarrow \mathbf{0}$
3:      for $\ell \leftarrow 1$ to $d$ do
4:          for $p \leftarrow 1$ to $P$ do
5:              $A_\ell(I_p) \leftarrow 0$, $B_\ell(I_p) \leftarrow 0$
6:      while $\boldsymbol{\alpha}$ is not optimal do
7:          for $v \leftarrow 1$ to $n$ do
8:              $G \leftarrow y_v \sum_{\ell=1}^{d} f_{PL}(x_v^{(\ell)}) - 1$
9:              $T \leftarrow \alpha_v$
10:             $\alpha_v \leftarrow \min(\max(\alpha_v - \frac{G}{Q_{vv}}, 0), C)$
11:             for $\ell \leftarrow 1$ to $d$ do
12:                 $p \leftarrow$ find($x_v^{(\ell)}$)        ▷ find $I_p$ ($O(P)$ time)
13:                 $A_\ell(I_p) \leftarrow A_\ell(I_p) + (\alpha_v - T) y_v x_v^{(\ell)}$
14:                 $B_\ell(I_p) \leftarrow B_\ell(I_p) + (\alpha_v - T) y_v$
15:     Find $\bar{\alpha}$ in $\boldsymbol{\alpha}$ such that $0 < \bar{\alpha} < C$
16:     $b \leftarrow \bar{y} - F_{PL}(\bar{\mathbf{x}})$
17:     return $b$, $A_\ell(I_p)$ and $B_\ell(I_p)$, where $1 \leq \ell \leq d$ and $1 \leq p \leq P$
```

Theorem 1. *Each outer iteration (line 6) in Algorithm 3 takes $O(nPd)$ time.*

Proof. The main bottlenecks of this algorithm are in the evaluation of gradient G (line 8) and the update of the parameters $A_\ell(I_p)$ and $B_\ell(I_p)$ (lines 12 to 14) which can be computed in $O(Pd)$ time. As such, one outer iteration takes $O(nPd)$ time. □

Unlike existing feature approximation methods (e.g., [59], [64]) for additive kernel functions, this method only needs additional space for storing $A_\ell(I_p)$ and $B_\ell(I_p)$ which only takes additional $O(Pd)$ space (cf. Lemma 3). Due to the few additional space, we can avoid using the huge amount of precious memory resources for training the large-scale datasets, especially for using the single computer with limited memory (e.g., 8GB/16GB).

Lemma 3. *The space complexity of Algorithm 3 is $O(nd + Pd)$.*

3.2 How to Minimize P with Error Guarantee in PLAME?

Compared with linear SVM, which takes $O(nd)$ time for the outer iteration (cf. line 3 in Algorithm 1), training kernel SVM with PLAME has additional overhead with the number of intervals P (cf. Theorem 1). However, the larger the number P, the more accurate between $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$ and $k(x_i^{(\ell)}, x^{(\ell)})$, i.e., the smaller the error $|k(x_i^{(\ell)}, x^{(\ell)}) - k_{PL}(x_i^{(\ell)}, x^{(\ell)})|$ (cf. Figure 3). Therefore, there is a trade-off between the accuracy and the training time.

Since our final goal is to provide the better approximation for the additive kernel $K(\mathbf{x_i}, \mathbf{x})$ in order to provide better accuracy, we propose to specify the error parameter ϵ, such that:

$$|k(x_i^{(\ell)}, x^{(\ell)}) - k_{PL}(x_i^{(\ell)}, x^{(\ell)})| \leq \epsilon \tag{18}$$

In Lemma 4, we show that once we specify for each dimension ℓ, $|k(x_i^{(\ell)}, x^{(\ell)}) - k_{PL}(x_i^{(\ell)}, x^{(\ell)})| \leq \epsilon$, we can obtain the good approximation $K_{PL}(\mathbf{x_i}, \mathbf{x})$ (cf. Equation 14) for $K(\mathbf{x_i}, \mathbf{x})$ with bounded error.

Lemma 4. *For each dimension ℓ, if:*

$$|k(x_i^{(\ell)}, x^{(\ell)}) - k_{PL}(x_i^{(\ell)}, x^{(\ell)})| \leq \epsilon$$

we obtain: $|K_{PL}(\mathbf{x_i}, \mathbf{x}) - K(\mathbf{x_i}, \mathbf{x})| \leq \epsilon d$.

Proof. We have:

$$k(x_i^{(\ell)}, x^{(\ell)}) - \epsilon \leq k_{PL}(x_i^{(\ell)}, x^{(\ell)}) \leq k(x_i^{(\ell)}, x^{(\ell)}) + \epsilon$$

By taking summation in both sides with respect to ℓ, we have:

$$K(\mathbf{x_i}, \mathbf{x}) - \epsilon d \leq K_{PL}(\mathbf{x_i}, \mathbf{x}) \leq K(\mathbf{x_i}, \mathbf{x}) + \epsilon d$$

Hence, we have proved the claim. □

As stated in Lemma 4, we need to ensure the largest gap between each linear segment in $k_{PL}(x_i^{(\ell)}, x^{(\ell)})$ and $k(x_i^{(\ell)}, x^{(\ell)})$ should be at most ϵ for every possible $x^{(\ell)}$ in order to achieve the theoretical guarantee. Here, we focus on the simple case (only one linear segment). Given the initial position l, we need to find u such that the gap $L(x_i^{(\ell)}, x^{(\ell)})$ (cf. Equation 19) is smaller than ϵ (cf. Figure 4).

$$L(x_i^{(\ell)}, x^{(\ell)}) = k(x_i^{(\ell)}, x^{(\ell)}) - (m_{x^{(\ell)}}(I) x_i^{(\ell)} + c_{x^{(\ell)}}(I)) \quad (19)$$

where the slope $m_{x^{(\ell)}}(I)$ and the intercept $c_{x^{(\ell)}}(I)$ (with respect to the interval $I = [l, u]$) are represented by Equation 12 and Equation 13, respectively.

Therefore, our goal is to ensure the maximum error (gap) value $\mathbb{E}(l, u)$, i.e., the value of the following optimization problem (G), is smaller than ϵ.

$$\begin{aligned} \mathbb{E}(l,u) = \quad & \underset{x_i^{(\ell)}, x^{(\ell)}}{\text{maximize}} \; L(x_i^{(\ell)}, x^{(\ell)}) \\ \text{such that} \quad & l \leq x_i^{(\ell)} \leq u \\ & \mathcal{L} \leq x^{(\ell)} \leq \mathcal{U} \end{aligned} \quad \text{(G)}$$

where $[\mathcal{L}, \mathcal{U}]$ is the possible region of $x^{(\ell)}$, which is normally $[0, 1]$, based on the normalization process of SVM [13].

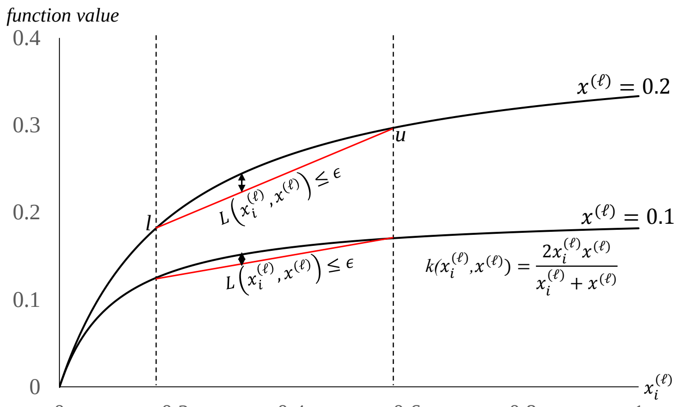


Fig. 4: Find the position u (solid red line) such that it achieves the gap $L(x_i^{(\ell)}, x^{(\ell)})$ with error guarantee ϵ, using the χ^2 kernel.

In order to find the best u, i.e., the linear segment which can cover the largest region but it still fulfills the ϵ guarantee, we need to develop the algorithm to search for this value in the $x_i^{(\ell)}$-axis (cf. Figure 4). However, one major challenge is that there are infinite numbers of possible u in the search space. In Lemma 5, we show that $\mathbb{E}(l, u)$ is monotonic increasing with respect to u. Based on this property, we can use the exponential search [6] to find the best u, which avoids searching the infinite numbers of possible u. We leave the proof of this lemma in Section 7.1.

Lemma 5. *Given the initial position l, and two positions u_1 and u_2 in $x_i^{(\ell)}$-axis, we have $\mathbb{E}(l, u_1) \leq \mathbb{E}(l, u_2)$ if $u_1 \leq u_2$.*

Even though the above method can find the best u given the initial position l, we still need to find the set of intervals ($\mathcal{I}$ in Algorithm 4) which covers the whole region $[L(\ell), U(\ell)]$ (cf. Figure 3). As such, we propose the optimal curve partition method (OCPM). The idea of this method is simple. We start the initial position as $L(\ell)$, i.e., $l = L(\ell)$. After that, we iteratively find the best u (with $\mathbb{E}(l, u) \leq \epsilon$) and then set the new l as this u until the new l reaches $U(\ell)$ (cf. Algorithm 4).

Algorithm 4 Optimal Curve Partition Method (OCPM)

```
1: procedure OCPM($\epsilon$, $d$, $[L(\ell), U(\ell)]$)
2:     $l \leftarrow L(\ell), \mathcal{I} \leftarrow \phi$
3:     while $l < U(\ell)$ do
4:         $u_{\text{next}} \leftarrow \arg\max_u \mathbb{E}(l, u)$, s.t. $\mathbb{E}(l, u) \leq \epsilon$        ▷ [6]
5:         $u_{\text{next}} \leftarrow \min(u, U(\ell))$
6:         $\mathcal{I} \leftarrow \mathcal{I} \cup \{[l, u_{\text{next}}]\}$
7:         $l \leftarrow u_{\text{next}}$
8:     return $\mathcal{I}$
```

Although OCPM is simple, this method can achieve the minimum number of intervals (i.e., optimal), as stated in Theorem 2. We leave the proof in Section 7.2.

Theorem 2. *Optimal Curve Partition Method (OCPM) produces the minimum number of intervals P for PLAME which fulfills the error parameter ϵ (cf. Equation 18).*

3.3 How to Solve the Optimization Problem (G)?

In Section 3.2, we utilize the exponential search algorithm to obtain the best u, i.e., line 4 in Algorithm 4. Observe that this method tries different possible values of u and regards the optimization problem (G) as the black box. However, there is still one remaining question. Given the l and u, how can we solve the optimization problem (G)?

In order to solve the optimization problem (G), we can adopt the commonly used coordinate descent method [67] (cf. Algorithm 5).

In Lemma 6, we show that Algorithm 5 converges to the global optimal solution of the optimization problem (G), given any l and u. We leave the proof of this lemma in Section 7.3.

Lemma 6. *Given any l and u, Algorithm 5 converges to the global optimal solution of the optimization problem (G).*

3.4 Case Study for χ^2 Kernel: The Worst-case Efficiency for Training SVM with PLAME

Recall that each outer iteration of Algorithm 3 (line 6) is at most $O(nPd)$ time (cf. Theorem 1), which is worse than

Algorithm 5 Coordinate Descent for (G)

```
1: procedure COORD_DESCENT($l, u, \mathcal{L}, \mathcal{U}$)
2:     $x_i^{(\ell)} \leftarrow u\ x^{(\ell)} \leftarrow \mathcal{U}$
3:     $\eta \leftarrow 0.001$                                   ▷ step size
4:     while $x_i^{(\ell)}$ and $x^{(\ell)}$ does not attain optimal do
5:         $x_i^{(\ell)} \leftarrow \min\Big(\max\Big(x_i^{(\ell)} + \eta\frac{\partial L(x_i^{(\ell)}, x^{(\ell)})}{\partial x_i^{(\ell)}}, l\Big), u\Big)$
6:         $x^{(\ell)} \leftarrow \min\Big(\max\Big(x^{(\ell)} + \eta\frac{\partial L(x_i^{(\ell)}, x^{(\ell)})}{\partial x_i^{(\ell)}}, \mathcal{L}\Big), \mathcal{U}\Big)$
7:     return $L(x_i^{(\ell)}, x^{(\ell)})$
```

Algorithm 1 for training linear SVM by at most P times. Observe that the number of intervals P only depends on ϵ (cf. Figure 4), we vary ϵ from 0.001 to 0.009 and investigate how the number of intervals P is affected by this error parameter ϵ (cf. Figure 5).

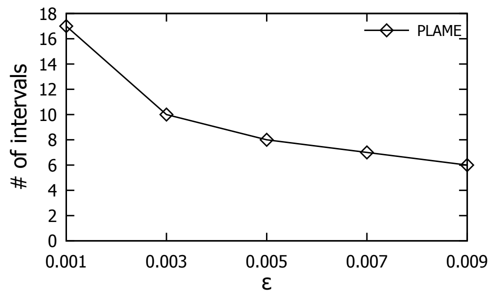


Fig. 5: Case study on how the number of intervals P is affected by ϵ, where $K(\mathbf{x_i}, \mathbf{x})$ is χ^2 kernel.

As a remark, the curve in Figure 5 only depends on the kernel function (cf. Figure 4), but not depends on the datasets. Therefore, we can know how the response time for training SVM with PLAME is worse than training the linear SVM model in advance, given any ϵ. As an example, once we set $\epsilon = 0.005$, the number of intervals P is 8. Therefore, we expect the response time for SVM with PLAME is slower than linear SVM by roughly 8 times.

4 EXPERIMENTS

We first introduce the experimental setting in Section 4.1. Next, we test how the parameter ϵ affects the accuracy and efficiency of SVM with PLAME in Section 4.2. Then, we investigate how the memory space of the feature approximation methods affects their accuracy in Section 4.3. After that, we test the accuracy and efficiency of all methods in Section 4.4 and Section 4.5, respectively. Lastly, we summarize the experimental results for all methods in Section 4.6.

4.1 Experimental Setting

In our experiments, we have used twelve datasets for training SVM classifiers which are summarized in Table 4. All these datasets can be found from either the LIBSVM website [13][2] or the UCI machine learning repository [1]. We divide each dataset into two parts which are the training (n_{tr}) and testing (n_t) datasets. Since the χ^2 kernel function is famous, we choose this kernel for conducting our experiments in this section. Some additional experiments for other kernels can be found in Section 7.4.

TABLE 4: Datasets.

Name	n_{tr}	n_t	dim	Total data size (MB)	Ref.
skin	220551	24506	3	5.61	[13]
ijcnn1	127522	14169	22	23.78	[13]
cod-rna	331152	157413	8	29.82	[13]
miniboone	117057	13007	50	49.62	[1]
home	836091	92900	10	70.88	[1]
SensIT	78823	19705	100	75.17	[13]
covtype_b	522910	58102	54	239.37	[13]
susy	4000000	1000000	18	686.65	[13]
hepmass	6300000	700000	28	1495.36	[1]
higgs	8800000	2200000	28	2349.85	[13]
casas	12560880	1395653	37	3939.75	[1], [19]
epsilon	400000	100000	2000	7629.39	[13]

Table 5 summarizes different state-of-the-art methods for SVM classification using additive kernels which are divided into three classes. For the first class (i.e., open-source-library-based methods), we modify the LIBSVM library [13] to support the additive kernel functions. In addition, we also train linear SVM models with the LIBLINEAR library [23]. For the second class (i.e., feature approximation methods), both VLFeat [59], Chebyshev [35], PL [47], NN [64], LD [17], [18], and ENL [7], [49] construct the high-dimensional feature vectors in order to capture the similar kernel function value $K(\mathbf{x_i}, \mathbf{x})$ (cf. Equation 8). For the third class (i.e., function approximation methods), PmSVM [69], [71] adopts the single quadratic function to approximate each one-dimensional kernel aggregation function $f(x^{(\ell)})$ (cf. Equation 9). Furthermore, our method, PLAME, utilizes the piecewise-linear function to approximate the one-dimensional kernel function (cf. Figure 3) with bounded error. We implemented all methods in C++[3] and conducted experiments on an Intel i7 3.2GHz PC. In this paper, we measure the training time (sec) and accuracy (cf. Equation 20, based on [13]) of each method.

$$\text{Accuracy} = \frac{\text{Number of correctly classified testing data}}{\text{Number of testing data } (n_t)} \tag{20}$$

As a remark, we only report the results in which the space consumption is within 16GB and the training time is within 259200 (sec) (i.e., three days).

4.2 Effect of ϵ for SVM with PLAME

Recall from Section 3 that both the efficiency and accuracy of training SVM models with PLAME depend on the error parameter ϵ. In this section, we vary the ϵ from 0.001 to 0.009 and report both the accuracy and efficiency results for PLAME in two datasets, skin and casas.

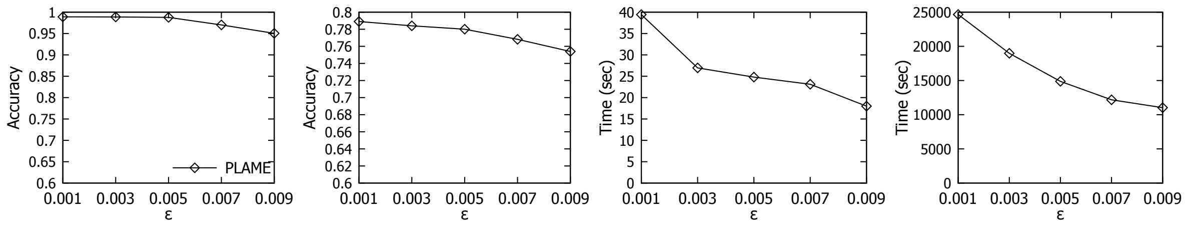
In Figure 6, observe that once we reduce the error parameter ϵ from 0.009 to 0.001, the accuracy increases in both datasets (cf. Figures 6a and 6b) since we can achieve more accurate approximation for the kernel function (cf. Lemma 4). However, as the number of intervals increases with smaller error ϵ (cf. Figure 5), it takes longer time for each iteration of Algorithm 3 (cf. Theorem 1). Therefore, the

2. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

3. The source code of our method PLAME can be found in this Github link https://anonymous.4open.science/r/PLAME-Code-0877/.

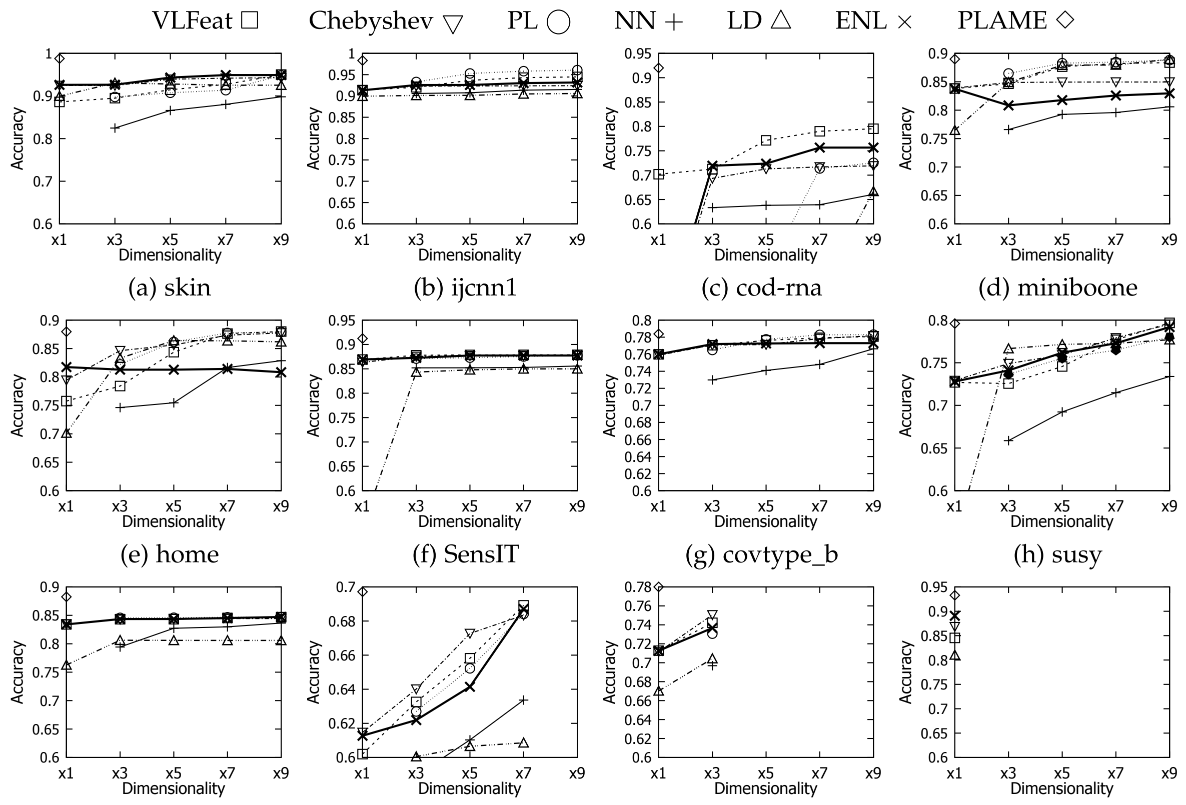
TABLE 5: Methods.

Type	Open-source library		Feature approximation						Function approximation	
Method	LIBSVM	LIBLINEAR	VLFeat	Chebyshev	PL	NN	LD	ENL	PmSVM	PLAME
Ref.	[13]	[23]	[59]	[35]	[47]	[64]	[17], [18]	[7], [49]	[69], [71]	ours



(a) Accuracy (skin) (b) Accuracy (casas) (c) Efficiency (skin) (d) Efficiency (casas)

Fig. 6: Accuracy and efficiency for SVM classification with PLAME, varying ϵ from 0.001 to 0.009, in different datasets.



(a) skin (b) ijcnn1 (c) cod-rna (d) miniboone

(e) home (f) SensIT (g) covtype_b (h) susy

(i) hepmass (j) higgs (k) casas (l) epsilon

Fig. 7: Trade-off between the memory space (i.e., dimensionality of the feature vectors) and the accuracy of feature approximation methods.

TABLE 6: Accuracy of all methods (n.a.: The running time of this method takes more than 3 days or the memory consumption of this method is more than 16GB.), where the top-2 accuracy values are in bold type for each dataset.

Method	skin	ijcnn1	cod-rna	miniboone	home	SensIT	covtype_b	susy	hepmass	higgs	casas	epsilon
LIBSVM	**0.992**	**0.986**	**0.957**	**0.891**	**0.88**	**0.918**	**0.796**	n.a.	n.a.	n.a.	n.a.	n.a.
LIBLINEAR	0.895	0.919	0.623	0.825	0.808	0.859	0.762	0.726	0.832	0.619	0.71	**0.879**
VLFeat	0.927	0.943	0.79	0.882	0.876	0.879	0.779	0.777	0.845	**0.689**	n.a.	n.a.
Chebyshev	0.942	0.924	0.717	0.85	0.873	0.877	0.778	**0.779**	0.844	0.683	n.a.	n.a.
PL	0.914	0.958	0.714	0.885	0.877	0.875	0.783	0.765	**0.847**	0.684	n.a.	n.a.
NN	0.88	0.914	0.639	0.796	0.816	0.853	0.748	0.715	0.83	0.634	n.a.	n.a.
LD	0.949	0.93	0.757	0.826	0.814	0.877	0.773	0.773	0.845	0.687	n.a.	n.a.
ENL	0.925	0.905	0.462	0.879	0.863	0.85	0.6	0.773	0.806	0.608	n.a.	n.a.
PmSVM	0.919	0.921	0.734	0.838	0.804	0.871	0.778	0.745	0.842	0.643	**0.728**	0.842
PLAME	**0.988**	**0.983**	**0.923**	**0.886**	**0.878**	**0.912**	**0.784**	**0.796**	**0.883**	**0.697**	**0.78**	**0.933**

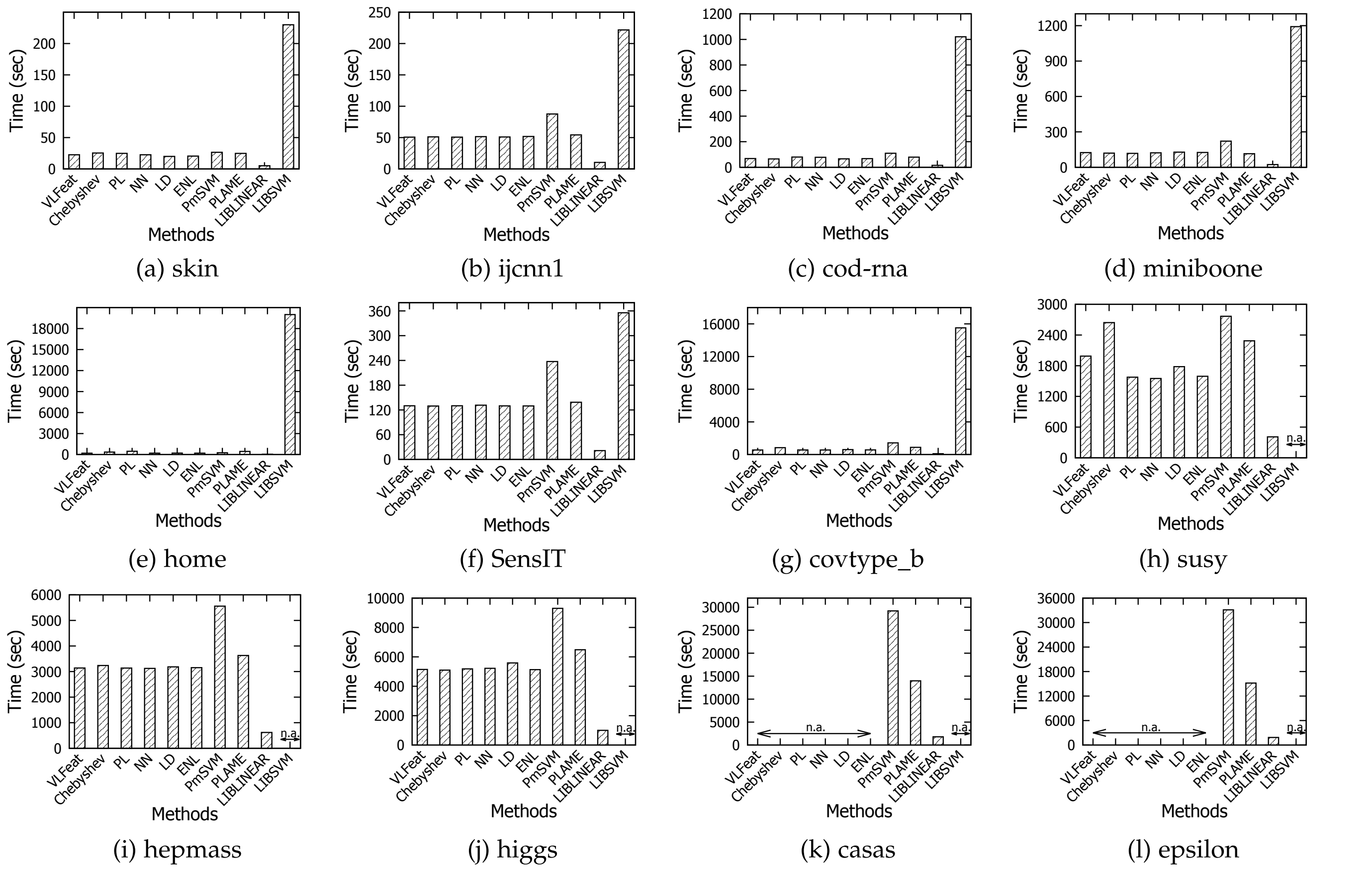


Fig. 8: Response time (sec) for all methods in the training phase, fixing the dimensionality of feature approximation methods as x7 of the original dimensionality.

training time is also longer (cf. Figures 6c and 6d). We notice that the accuracy is stable when we choose $\epsilon = 0.005$. As such, we choose $\epsilon = 0.005$ by default in later experiments.

4.3 Effect of Memory Space for Feature Approximation Methods

Recall that feature approximation methods (cf. Table 1) need to generate the high-dimensional feature vectors in order to achieve the good accuracy. In this experiment, we compare how the memory space affects the accuracy of all feature approximation methods in different datasets, by varying the dimensionality of the feature vectors with x1, x3, x5, x7, and x9 of the dimensionality of the original feature vectors (cf. Figure 7). As a remark, both the feature approximation methods NN and PL do not support generating the feature vector with x1 feature dimensionality. Therefore, we omit those points in these figures. Here, we also report the results of our method PLAME, which follows the default setting, i.e., $\epsilon = 0.005$, as the reference. Unlike the feature approximation methods, PLAME only uses the original feature vector. As such, we only report the accuracy result with x1 feature dimensionality.

In Figure 7, observe that once we increase the dimensionality, most of these feature approximation methods can achieve higher accuracy. However, the accuracy of these methods is normally inferior compared with our method PLAME. In addition, this type of methods can significantly consume more precious memory resources, especially for large scale datasets. Using the casas dataset as an example, once we increase the dimensionality to x5, the space consumption of the feature vectors is roughly 17.3GB, which cannot fit in a single PC with 16GB memory. However, our method PLAME only needs a few additional space (cf. Lemma 3) for training. Here, we observe that these feature approximation methods can provide stable accuracy in most of these datasets with x7 dimensionality. Therefore, we choose x7 dimensionality as default in later experiments.

4.4 Accuracy of All Methods

In this section, we compare the accuracy of our method PLAME with other methods. In Table 6, observe that LIBSVM normally provides the highest accuracy for small to medium-scale datasets compared with other methods. However, once the size of the datasets (e.g., susy, hepmass, casas, and epsilon) becomes larger, LIBSVM cannot train the classifiers within three days due to the high time complexity of this method. Even though our method PLAME may not achieve the best performance for all datasets, PLAME can achieve the competitive accuracy (i.e., top-2 accuracy values) without using a huge amount of training time compared with the best method LIBSVM. Therefore, PLAME can support large-scale datasets that cannot be supported by LIBSVM. On the other hand, PLAME can achieve the best accuracy compared with feature approximation and function approximation methods.

4.5 Efficiency of All Methods

We proceed to compare the efficiency of all methods. Since all the approximation methods either directly adopt the linear SVM solver (e.g., LIBLINEAR) for the feature maps with higher dimensionality (i.e., feature approximation methods) or adopt the slightly modified linear SVM solver (i.e., function approximation methods), all of them can achieve similar response time for training the SVM models (cf. Figure 8), which are significantly faster than LIBSVM (based on the inefficient kernel (nonlinear) SVM solver). In practice, PLAME is at most 8.55x slower than the fastest method LIBLINEAR, which is roughly similar to the worst-case efficiency performance of PLAME (cf. Figure 5) with default setting (i.e., $P = 8$ with $\epsilon = 0.005$). As a remark, we omit the results of all feature approximation methods in casas and epsilon datasets, as the memory consumptions are more than 16GB. On the other hand, the training time of LIBSVM method in susy, hepmass, higgs, casas, and epsilon datasets is more than 259200 (sec) (i.e., three days).

4.6 Summary of the Experimental Results

In this section, we summarize the experimental results of both PLAME and existing methods. Even though PLAME may not achieve the best performance in terms of accuracy (cf. Table 6) and training time (cf. Figure 8), PLAME can achieve the competitive performance in these two aspects. Moreover, existing methods can fail at least one of these three conditions, (1) low classification error, (2) low memory space, and (3) low training time.

- High classification error for LIBLINEAR and PmSVM (cf. Table 6)
- High memory space (in order to achieve better accuracy (cf. Figure 7)) and high classification error (cf. Table 6) for feature approximation methods
- High training time for LIBSVM (cf. Figure 8)

Unlike the existing methods, PLAME can achieve the best trade-off between the accuracy (classification error), memory space, and training time in order to simultaneously fulfill these three conditions.

5 Related Work

Kernel-based SVM classification has been widely used in many application domains, including human activity recognition/detection [37], [41], [51], [52], [63], [79], pedestrian detection [4], [5], geoscience and remote sensing [8], [21]. For these types of applications, they normally adopt the additive kernels [59], e.g., χ^2 kernel, to improve the accuracy results. However, compared with training linear SVM models, it is time-consuming to train SVM models using additive kernel functions [54], [60], [64]. As such, many efficient techniques have been developed in the literature which can be divided into two camps, (1) feature approximation and (2) function approximation.

In the first camp, some researchers [33], [35], [47], [59], [60], [64] observe that there are many efficient linear SVM solvers, e.g., LIBLINEAR [23], Pegasos [56], and SVM-Perf [31], for training linear SVM models. As such, they propose to construct the high-dimensional feature maps such that the inner product between any two feature maps can capture the kernel function value (cf. Equation 1). However, all of these studies need to generate the new feature maps with much higher dimensions in order to achieve better accuracy which can consume huge memory resources, especially for large-scale datasets (e.g., GB-scale). Furthermore, these methods cannot provide competitive accuracy (cf. Table 6) compared with our method PLAME and the kernel SVM solver LIBSVM. In addition, given the absolute error ϵ, most of these studies, e.g., [47], [64], cannot provide the theoretical guarantee between the exact kernel value and the inner product between the feature maps. Moreover, there are also many other types of feature approximation methods [3], [9], [14], [34], [38], [48], [50], [53], [74], [76]. However, these methods focus on other types of kernel functions, e.g., Gaussian and polynomial kernels, which cannot support additive kernels.

In the second camp, which is mostly related to our work, some researchers propose using some simple functions to approximate the complicated kernel aggregation function $F(\mathbf{x})$ (cf. Equation 6). Both Wu et al. [69], [71] and Yang et al. [75] propose using a single quadratic function to approximate the one-dimensional kernel aggregation function $f(x^{(\ell)})$ (cf. Equation 9). Theoretically, $f(x^{(\ell)})$ may not be in the shape of quadratic function (i.e., no theoretical guarantee between this approximation and the exact value of $f(x^{(\ell)})$). As such, this approach [69], [71], [75] can achieve inferior accuracy results compared with our method (cf. the PmSVM method in Table 6). In addition, Maji et al. [45] propose to utilize piecewise polynomial function to approximate $f(x^{(\ell)})$. However, this approach cannot provide the approximation guarantee (like [69], [71], [75]) and only focus on the testing phase, which cannot be extended to support the time-consuming training phase. Recently, Chan et al. [10], [12] propose to combine the hierarchical indexing framework [24], [25] with the linear function [12] and the quadratic function [10] to approximate the kernel aggregation function $F(\mathbf{x})$ with different types of kernel functions. Although these studies [10], [12] can provide the theoretical guarantee between the approximate result and the exact value $F(\mathbf{x})$, these methods only focus on the testing phase, do not consider additive kernel functions, and suffer from high worst-case time complexities for evaluating $F(\mathbf{x})$ (e.g., $O(nd)$ time for [12] and $O(nd^2)$ time for [10]), which are far worse than our method ($O(Pd)$ time, where $P << n$). Furthermore, Baek et al. [4], [5] and Chan et al. [11] propose to precompute some $f(x^{(\ell)})$ values for each dimension in advance, which can further improve the efficiency for the testing phase in exact SVM. However, like [10], [12], [45], both Baek et al. [4], [5] and Chan et al. [11] cannot support the training phase. Even though some research studies in approximation theory [15], [62] also use some simple functions to approximate more complicated functions, none of these approaches, e.g., interpolation or curve fitting, can be easily extended to support efficient training for SVM models with theoretical guarantee (cf. Lemma 4), to the best of our knowledge.

Compared with the above two camps of research work, we can provide the approximation $K_{PL}(\mathbf{x_i}, \mathbf{x})$ for $K(\mathbf{x_i}, \mathbf{x})$ with bounded error (cf. Lemma 4), slight time overhead (cf. Theorem 1), and slight memory overhead (cf. Lemma 3) for training SVM models using PLAME.

In both database and machine learning communities, there are also many other studies [2], [20], [29], [32], [34], [39], [77] for speeding up the training and testing phases of SVM, which are not based on feature approximation and function approximation methods, e.g., support vector reduction [32], [39] and indexing [77]. However, all of these studies mainly focus on Gaussian, polynomial, and sigmoid kernels, which cannot be easily extended to additive kernels. Recently, many research studies also exploit the opportunities for adopting the parallel/distributed computation, e.g., [30], [68], and the graphics processing unit (GPU), e.g., [65], [66], to boost the efficiency for the training phase of SVM, which can consume more computational resources (e.g., 32 machines are used in [30]). Due to space limitations, we mainly focus on the single machine setting with CPU in this work (like [13]) and leave the combination of our method PLAME with these methods in the future work.

6 CONCLUSION

In this paper, we propose the new similarity measure, called PLAME, which can be used to approximate the additive kernel functions in order to efficiently and effectively train the additive kernel SVM model. We further show that (1) the approximation error between PLAME and the additive kernel function can be theoretically close and (2) training SVM with PLAME is slightly slower than training linear SVM (e.g., LIBLINEAR [23]) without incurring large additional space. Experimental studies on twelve datasets demonstrate that our method can retain the high accuracy, can achieve the small response time, and can incur nearly no space overhead compared with two types of state-of-the-art methods, including feature-approximation-based and function-approximation-based methods, and two types of commonly-used software, including LIBSVM [13] and LIBLINEAR [23].

In the future, we will further apply PLAME to boost the efficiency of other statistical and machine learning models, e.g., kernel k-means clustering [16], kernel regression [55], and kernel density estimation [10]–[12]. In addition, we will investigate how to apply the high-order polynomial function to accurately approximate the additive kernel functions in order to further boost both the efficiency and accuracy performances. Moreover, we will study the efficiency issues for other types of commonly used kernel functions, e.g., Gaussian kernel [55], isolation kernel [58], [72], graph kernels [61], and string kernels [42], in different machine learning models. Furthermore, like the previous studies [30], [65], [66], [68], we will also exploit the opportunities for using parallel/distributed computation and GPU to further improve the efficiency for training SVM models with PLAME.

7 APPENDIX

7.1 Proof of Lemma 5

Proof. Observe from Figure 9 that $L(x_i^{(\ell)}, x^{(\ell)})$ becomes larger once u is larger. Therefore, the objective function of the optimization problem (G) is monotonic increasing with respect to u. In addition, the search space is also larger if u is larger, based on the first constraint $\ell \leq x_i^{(\ell)} \leq u$ of (G). Since (G) is the maximization problem, we can conclude that $\mathbb{E}(l, u_1) \leq \mathbb{E}(l, u_2)$ if $u_1 \leq u_2$. □

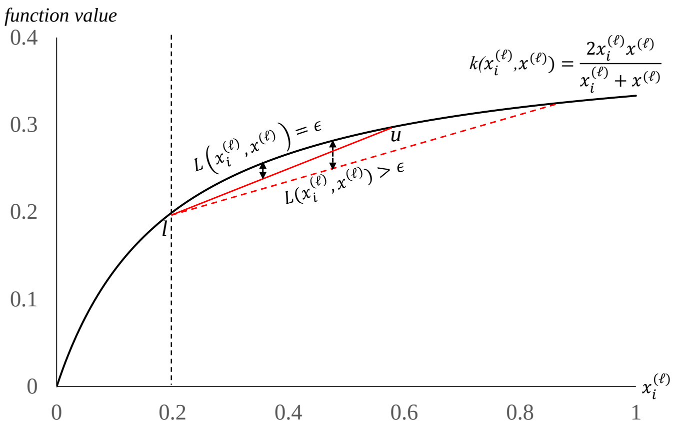


Fig. 9: $L(x_i^{(\ell)}, x^{(\ell)})$ is monotonic increasing with respect to u.

7.2 Proof of Theorem 2

Proof. We first let $\mathcal{I}^{\text{OCPM}}$ and $\mathcal{I}^{\text{OPT}}$ be two sets of intervals of the methods OCPM and optimal solution, respectively. Then, we denote $\mathcal{I}^{\text{OCPM}} = \{I_1^{\text{OCPM}}, I_2^{\text{OCPM}}, ...\}$ and $\mathcal{I}^{\text{OPT}} = \{I_1^{\text{OPT}}, I_2^{\text{OPT}}, ...\}$. After that, we also denote $I_k^{\text{OCPM}}.l$ ($I_k^{\text{OPT}}.l$) and $I_k^{\text{OCPM}}.u$ ($I_k^{\text{OPT}}.u$) as the start position and the end position of the interval for OCPM (OPT), respectively. Without loss of generality, we also let the k^{th} interval be just before the $(k+1)^{\text{th}}$ interval, i.e.,

$$I_k^{\text{OCPM}}.u = I_{k+1}^{\text{OCPM}}.l \text{ and } I_k^{\text{OPT}}.u = I_{k+1}^{\text{OPT}}.l$$

Since every method must cover the whole region, we have: $I_1^{\text{OCPM}}.l = I_1^{\text{OPT}}.l = L(\ell)$. Based on Lemma 5, we know that $\mathbb{E}(l, u)$ is the monotonic increasing function with respect to u. Therefore, the exponential search [6] can find the largest u such that it can still fulfill the absolute error condition (i.e., ϵ). As such, we have:

$$I_1^{\text{OCPM}}.u \geq I_1^{\text{OPT}}.u$$

Assume it is also true for the k^{th} interval that:

$$I_k^{\text{OCPM}}.u \geq I_k^{\text{OPT}}.u$$

Therefore, we have:

$$I_{k+1}^{\text{OCPM}}.l \geq I_{k+1}^{\text{OPT}}.l$$

Based on the correctness property of optimal solution, we also know that:

$$\mathbb{E}(I_{k+1}^{\text{OPT}}.l, I_{k+1}^{\text{OPT}}.u) \leq \epsilon$$

In Figure 10, since the red interval $[I_{k+1}^{\text{OCPM}}.l, I_{k+1}^{\text{OPT}}.u]$ is the subset of I_k^{OPT}, we have:

$$\mathbb{E}(I_{k+1}^{\text{OCPM}}.l, I_{k+1}^{\text{OPT}}.u) \leq \mathbb{E}(I_{k+1}^{\text{OPT}}.l, I_{k+1}^{\text{OPT}}.u) \leq \epsilon$$

We omit the proof of the first inequality, as it uses the same concept of Lemma 5.

Recall that the exponential search [6] can find the largest u such that it can achieve the absolute error guarantee (ϵ). Therefore, based on the above inequality $\mathbb{E}(I_{k+1}^{\text{OCPM}}.l, I_{k+1}^{\text{OPT}}.u) \leq \epsilon$, we can conclude:

$$I_{k+1}^{\text{OCPM}}.u \geq I_{k+1}^{\text{OPT}}.u$$

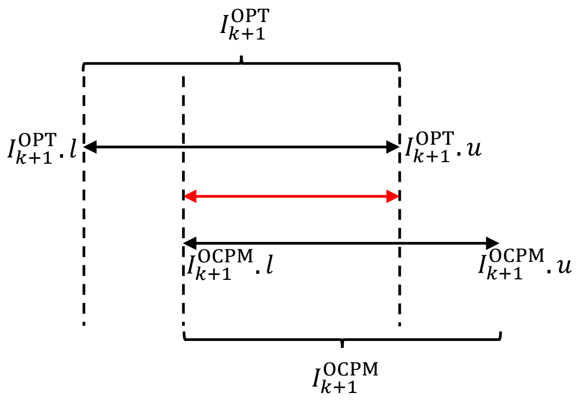


Fig. 10: The $(k+1)^{\text{th}}$ intervals (black) for both OCPM and OPT methods.

By induction, we show that $I_v^{\text{OCPM}}.u \geq I_v^{\text{OPT}}.u$ for each v^{th} interval in the sets $\mathcal{I}^{\text{OCPM}}$ and $\mathcal{I}^{\text{OPT}}$ which also means that OCPM needs smaller (or at most equal) number of intervals compared with OPT. However, OPT is the optimal solution which must produce the smallest number of intervals. As such, OCPM produces the minimum number of intervals (i.e., achieves optimal solution). $\square$

7.3 Proof of Lemma 6

To achieve the result in Lemma 6, we need to prove two properties which are: (1) there exists the local maximum solution for optimization problem (G) and (2) this local maximum solution is the global maximum solution.

In the optimization problem (G), we notice that the constraint region (i.e., $l \leq x_i^{(\ell)} \leq u$ and $\mathcal{L} \leq x^{(\ell)} \leq \mathcal{U}$) is bounded and closed, i.e., compact space [57]. As such, by Weierstrass theorem (Theorem 2.2 in [26]), we can show the property (1).

In order to show the property (2), we find the Hessian matrix of $L(x_i^{(\ell)}, x^{(\ell)})$, i.e.,

$$H = \begin{bmatrix} \frac{\partial^2 L}{\partial {x_i^{(\ell)}}^2} & \frac{\partial^2 L}{\partial x_i^{(\ell)} \partial x^{(\ell)}} \\ \frac{\partial^2 L}{\partial x_i^{(\ell)} \partial x^{(\ell)}} & \frac{\partial^2 L}{\partial {x^{(\ell)}}^2} \end{bmatrix}$$

If $x_i^{(\ell)}$ and $x^{(\ell)}$ fulfill the constraints of the optimization problem (G), we can achieve the following expressions, by some algebraic and calculus operations:

$$\frac{\partial^2 L}{\partial {x_i^{(\ell)}}^2} \leq 0 \text{ and } |H| \geq 0$$

As such, the Hessian matrix H is negative definite matrix. Therefore, $L(x_i^{(\ell)}, x^{(\ell)})$ is the concave function once $x_i^{(\ell)}$ and $x^{(\ell)}$ are within the constraint region of the optimization problem (G). Hence, we prove the property (2).

7.4 Additional Experiments for Other Kernels

In this section, we further conduct additional experiments for testing the accuracy and efficiency for all methods using other additive kernel functions, which are (1) JS kernel and (2) Hellinger kernel. Due to space limitations, we omit the results for adopting the intersection kernel and only test the performance of these methods using two datasets, which are cod-rna (with a small size) and casas (with a large size), in Table 4.

Accuracy of all methods using other additive kernels: In the first experiment, we follow the default settings in Section 4.4 (e.g., using x7 dimensionality for all feature approximation methods and choosing $\epsilon = 0.005$ for PLAME) to test the accuracy of all methods (cf. Table 7). Like Section 4.4, our method, PLAME, achieves the top-2 accuracy results no matter which kernel functions we adopt. Although LIBSVM achieves the best performance in the small-scale dataset, i.e., cod-rna, this method cannot train the SVM model within three days for the large-scale dataset, i.e., casas. In addition, all feature approximation methods, namely VLFeat, PL, NN, LD, and ENL, need to consume more than 16GB space for training SVM models in the casas dataset. Therefore, PLAME is the only method that (1) is scalable to large-scale datasets, (2) does not incur huge space overhead, and (3) retains high accuracy results. As a remark, since the Chebyshev method can only support the χ^2 kernel function, we omit the results of this method in Table 7.

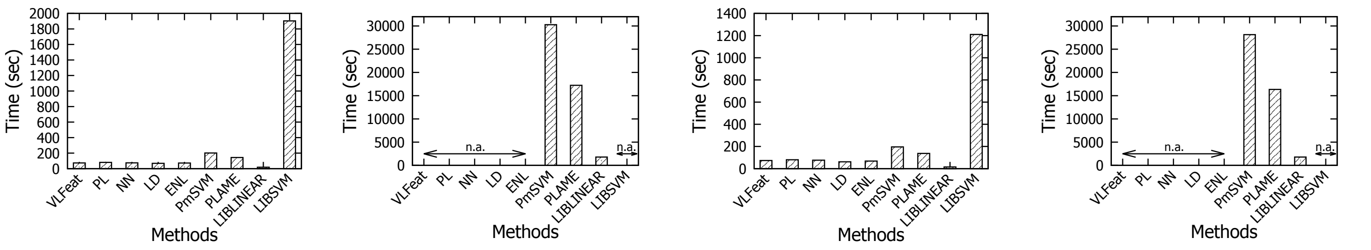
TABLE 7: Accuracy of all methods with different kernel functions (n.a.: The running time of this method takes more than 3 days or the memory consumption of this method is more than 16GB.), where the top-2 accuracy values are in bold type for each dataset.

Method	JS kernel		Hellinger kernel	
	cod-rna	casas	cod-rna	casas
LIBSVM	**0.956**	n.a.	**0.948**	n.a.
LIBLINEAR	0.623	0.71	0.623	**0.71**
VLFeat	0.783	n.a.	0.774	n.a.
PL	0.714	n.a.	0.714	n.a.
NN	0.611	n.a.	0.657	n.a.
LD	0.728	n.a.	0.712	n.a.
ENL	0.441	n.a.	0.437	n.a.
PmSVM	0.675	**0.718**	0.692	0.707
PLAME	**0.918**	**0.773**	**0.921**	**0.767**

Efficiency of all methods using other additive kernels: In the second experiment, we proceed to test the efficiency of all methods for training SVM models in the cod-rna and casas datasets using the JS kernel and Hellinger kernel. Figure 11 shows the results of all methods. Like Section 4.5, since the feature approximation methods and function approximation methods mainly adopt the linear SVM solver and adopt the slightly modified linear SVM solver, respectively, to train SVM models, all these methods achieve similar response time for using the small-scale dataset, i.e., cod-rna. In addition, all feature approximation methods and LIBSVM need to consume more than 16GB space and take more than 3 days, respectively, for training SVM models in the large-scale dataset. As such, PLAME is the only method that (1) takes small training time and (2) does not consume huge memory space.

Acknowledgement

This work was supported by the NSFC grant 62202401, the Science and Technology Development Fund Macau SAR 0015/2019/AKP, 0031/2022/A, SKL-IOTSC-2021-2023, the Research Grant of University of Macau MYRG2022-00252-FST, Wuyi University Hong Kong and Macau joint Research Fund 2021WGALH14, the University of Hong Kong (Projects 104005858 and 10400599), and the Guangdong–Hong Kong-Macau Joint Laboratory Program 2020 (Project No: 2020B1212030009).



(a) cod-rna (JS kernel) (b) casas (JS kernel) (c) cod-rna (Hellinger kernel) (d) casas (Hellinger kernel)

Fig. 11: Response time (sec) for all methods in the training phase with other additive kernels, fixing the dimensionality of feature approximation methods as x7 of the original dimensionality.

References

[1] UCI machine learning repository. http://archive.ics.uci.edu/ml/index.php.

[2] F. Angiulli and A. Astorino. Scaling up support vector machines using nearest neighbor condensation. *IEEE Trans. Neural Networks*, 21(2):351–357, 2010.

[3] H. Avron, V. Sindhwani, J. Yang, and M. W. Mahoney. Quasi-monte carlo feature maps for shift-invariant kernels. *J. Mach. Learn. Res.*, 17:120:1–120:38, 2016.

[4] J. Baek, J. Hyun, and E. Kim. A pedestrian detection system accelerated by kernelized proposals. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–13, 2019.

[5] J. Baek, J. Kim, and E. Kim. Fast and efficient pedestrian detection via the cascade implementation of an additive kernel support vector machine. *IEEE Trans. Intelligent Transportation Systems*, 18(4):902–916, 2017.

[6] J. L. Bentley and A. C. Yao. An almost optimal algorithm for unbounded searching. *Inf. Process. Lett.*, 5(3):82–87, 1976.

[7] M. Boroumand and J. J. Fridrich. Applications of explicit non-linear feature maps in steganalysis. *IEEE Trans. Inf. Forensics Secur.*, 13(4):823–833, 2018.

[8] Ü. Budak, U. Halici, A. Sengür, M. Karabatak, and Y. Xiao. Efficient airport detection using line segment detector and fisher vector representation. *IEEE Geosci. Remote Sensing Lett.*, 13(8):1079–1083, 2016.

[9] B. Bullins, C. Zhang, and Y. Zhang. Not-so-random features. In *ICLR*, 2018.

[10] T. N. Chan, R. Cheng, and M. L. Yiu. QUAD: Quadratic-bound-based kernel density visualization. In *SIGMOD*, pages 35–50, 2020.

[11] T. N. Chan, L. H. U, R. Cheng, M. L. Yiu, and S. Mittal. Efficient algorithms for kernel aggregation queries. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[12] T. N. Chan, M. L. Yiu, and L. H. U. KARL: Fast kernel aggregation queries. In *ICDE*, pages 542–553, 2019.

[13] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[14] M. Chen and K. Lin. Efficient kernel approximation for large-scale support vector machine classification. In *SDM*, pages 211–222, 2011.

[15] E. Cheney and W. Light. *A Course in Approximation Theory*. Mathematics Series. Brooks/Cole Publishing Company, 2000.

[16] R. Chitta, R. Jin, T. C. Havens, and A. K. Jain. Approximate kernel k-means: solution to large scale kernel clustering. In *SIGKDD*, pages 895–903, 2011.

[17] O. Chum. Low dimensional explicit feature maps. In *ICCV*, pages 4077–4085, 2015.

[18] O. Chum. Optimizing explicit feature maps on intervals. *Image Vis. Comput.*, 66:36–47, 2017.

[19] D. J. Cook. Learning setting-generalized activity models for smart spaces. *IEEE Intelligent Systems*, 27(1):32–38, 2012.

[20] M. Cossalter, R. Yan, and L. Zheng. Adaptive kernel approximation for large-scale non-linear SVM prediction. In *ICML*, pages 409–416, 2011.

[21] B. Demir and L. Bruzzone. Histogram-based attribute profiles for classification of very high resolution remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(4):2096–2107, April 2016.

[22] D. K. Duvenaud, H. Nickisch, and C. E. Rasmussen. Additive gaussian processes. In *NIPS*, pages 226–234, 2011.

[23] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[24] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *ACM SIGMOD*, pages 945–959, 2017.

[25] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, pages 203–211, 2003.

[26] O. Güler. *Foundations of Optimization*. Graduate Texts in Mathematics. Springer New York, 2010.

[27] Y. Guo, Y. Li, and Z. Shao. DSRF: A flexible trajectory descriptor for articulated human action recognition. *Pattern Recognit.*, 76:137–148, 2018.

[28] C. Hsieh, K. Chang, C. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML*, pages 408–415, 2008.

[29] C. Hsieh, S. Si, and I. S. Dhillon. Fast prediction for large-scale kernel machines. In *NIPS*, pages 3689–3697, 2014.

[30] C. Hsieh, S. Si, and I. S. Dhillon. Communication-efficient distributed block minimization for nonlinear kernel machines. In *SIGKDD*, pages 245–254, 2017.

[31] T. Joachims. Training linear svms in linear time. In *SIGKDD*, pages 217–226, 2006.

[32] H. G. Jung and G. Kim. Support vector number reduction: Survey and experimental evaluations. *IEEE Trans. Intelligent Transportation Systems*, 15(2):463–476, 2014.

[33] S. Kim and S. Choi. Binary embedding with additive homogeneous kernels. In *AAAI*, pages 2094–2100, 2017.

[34] Q. V. Le, T. Sarlós, and A. J. Smola. Fastfood - computing hilbert space expansions in loglinear time. In *ICML*, pages 244–252, 2013.

[35] F. Li, G. Lebanon, and C. Sminchisescu. Chebyshev approximations to the histogram χ^2 kernel. In *CVPR*, pages 2424–2431, 2012.

[36] W. Li, M. Coats, J. Zhang, and S. Mckenna. Discriminating dysplasia: Optical tomographic texture analysis of colorectal polyps. *Medical image analysis*, 26:57–69, 09 2015.

[37] W. Li and N. Vasconcelos. Complex activity recognition via attribute dynamics. *International Journal of Computer Vision*, 122(2):334–370, 2017.

[38] Y. Li, K. Zhang, J. Wang, and S. Kumar. Learning adaptive random features. In *AAAI*, pages 4229–4236, 2019.

[39] X. Liang. An effective method of pruning support vector machine classifiers. *IEEE Trans. Neural Networks*, 21(1):26–38, 2010.

[40] B. Liu, H. Cai, Z. Ju, and H. Liu. RGB-D sensing based human action and interaction analysis: A survey. *Pattern Recognit.*, 94:1–12, 2019.

[41] M. Liu, H. Liu, and C. Chen. 3d action recognition using multiscale energy-based global ternary image. *IEEE Trans. Circuits Syst. Video Techn.*, 28(8):1824–1838, 2018.

[42] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. J. C. H. Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, 2002.

[43] S. Maji and A. C. Berg. Max-margin additive classifiers for detection. In *ICCV*, pages 40–47, 2009.

[44] S. Maji, A. C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *CVPR*, 2008.

[45] S. Maji, A. C. Berg, and J. Malik. Efficient classification for additive kernel svms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):66–77, 2013.

[46] M. Mutny and A. Krause. Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. In *NeurIPS*, pages 9019–9030, 2018.

[47] O. Pele, B. Taskar, A. Globerson, and M. Werman. The pairwise piecewise-linear embedding for efficient non-linear classification. In *ICML*, pages 205–213, 2013.

[48] J. Pennington, F. X. Yu, and S. Kumar. Spherical random features for polynomial kernels. In *NIPS*, pages 1846–1854, 2015.

[49] F. Perronnin, J. Sánchez, and Y. Liu. Large-scale image categorization with explicit data embedding. In *CVPR*, pages 2297–2304, 2010.

[50] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *SIGKDD*, pages 239–247, 2013.

[51] J. Qi, Z. Wang, X. Lin, and C. Li. Learning complex spatio-temporal configurations of body joints for online activity recognition. *IEEE Trans. Human-Machine Systems*, 48(6):637–647, 2018.

[52] H. Qian, S. J. Pan, and C. Miao. Sensor-based activity recognition via learning from distributions. In *AAAI*, pages 6262–6269, 2018.

[53] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, pages 1177–1184, 2007.

[54] V. Risojevic and Z. Babic. Unsupervised quaternion feature learning for remote sensing image classification. *IEEE J Sel. Topics in Appl. Earth Observ. and Remote Sensing*, 9(4):1521–1531, 2016.

[55] B. Schölkopf and A. J. Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond*. Adaptive computation and machine learning series. MIT Press, 2002.

[56] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *ICML*, pages 807–814, 2007.

[57] B. Thomson, J. Bruckner, and A. Bruckner. *Elementary Real Analysis*. Prentice-Hall, 2008.

[58] K. M. Ting, Y. Zhu, and Z. Zhou. Isolation kernel and its effect on SVM. In *SIGKDD*, pages 2329–2337, 2018.

[59] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(3):480–492, 2012.

[60] A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *CVPR*, pages 2320–2327, 2012.

[61] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, 2010.

[62] I. A. S. Vladislav Kirillovich Dziadyk. *Theory of Uniform Approximation of Functions by Polynomials*. Walter De Gruyter, 2008.

[63] L. Wang, X. Zhao, Y. Si, L. Cao, and Y. Liu. Context-associative hierarchical memory model for human activity recognition and prediction. *IEEE Trans. Multimedia*, 19(3):646–659, 2017.

[64] Z. Wang, X. Yuan, Q. Liu, and S. Yan. Additive nearest neighbor feature maps. In *ICCV*, pages 2866–2874, 2015.

[65] Z. Wen, J. Shi, B. He, J. Chen, and Y. Chen. Efficient multi-class probabilistic svms on gpus. *IEEE Trans. Knowl. Data Eng.*, 31(9):1693–1706, 2019.

[66] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen. Thundersvm: A fast SVM library on gpus and cpus. *J. Mach. Learn. Res.*, 19:21:1–21:5, 2018.

[67] S. J. Wright. Coordinate descent algorithms. *Math. Program.*, 151(1):3–34, June 2015.

[68] H. Wu, X. Huang, Q. Luo, and Z. Yang. Ppd: A scalable and efficient parallel primal-dual coordinate descent algorithm. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[69] J. Wu. Power mean SVM for large scale visual classification. In *CVPR*, pages 2344–2351, 2012.

[70] J. Wu, W. Tan, and J. M. Rehg. Efficient and effective visual codebook generation using additive kernels. *Journal of Machine Learning Research*, 12:3097–3118, 2011.

[71] J. Wu and H. Yang. Linear regression-based efficient SVM learning for large-scale classification. *IEEE Trans. Neural Netw. Learning Syst.*, 26(10):2357–2369, 2015.

[72] B. Xu, K. M. Ting, and Z. Zhou. Isolation set-kernel and its application to multi-instance learning. In *SIGKDD*, pages 941–949, 2019.

[73] J. Xu, J. Han, F. Nie, and X. Li. Multi-view scaling support vector machines for classification and feature selection. *IEEE Trans. Knowl. Data Eng.*, 32(7):1419–1430, 2020.

[74] Y. Xu, C. Miao, Y. Liu, H. Song, Y. Hu, and H. Min. Kernel-target alignment based non-linear metric learning. *Neurocomputing*, 411:54–66, 2020.

[75] H. Yang and J. Wu. Practical large scale classification with additive kernels. In *ACML*, pages 523–538, 2012.

[76] F. X. Yu, A. T. Suresh, K. M. Choromanski, D. N. Holtmann-Rice, and S. Kumar. Orthogonal random features. In *NIPS*, pages 1975–1983, 2016.

[77] H. Yu, I. Ko, Y. Kim, S. Hwang, and W. Han. Exact indexing for support vector machines. In *SIGMOD*, pages 709–720, 2011.

[78] Z. Yu, X. Jiang, F. Zhou, J. Qin, D. Ni, S. Chen, B. Lei, and T. Wang. Melanoma recognition in dermoscopy images via aggregated deep convolutional features. *IEEE Trans. Biomed. Engineering*, 66(4):1006–1016, 2019.

[79] H. Zhang and L. E. Parker. Bio-inspired predictive orientation decomposition of skeleton trajectories for real-time human activity prediction. In *ICRA*, pages 3053–3060, May 2015.

[80] J. Zhao, L. Wang, R. S. Cabral, and F. D. la Torre. Feature and region selection for visual learning. *IEEE Trans. Image Process.*, 25(3):1084–1094, 2016.

Tsz Nam Chan received the bachelor's degree in electronic and information engineering and the PhD degree in computer science from the Hong Kong Polytechnic University in 2014 and 2019, respectively. He worked as the postdoctoral researcher in The University of Hong Kong from Sep 2018 to Aug 2020. He is currently a research assistant professor in the Hong Kong Baptist University. His research interests include spatiotemporal data management, large-scale data visualization, and kernel methods for machine learning. He is a member of IEEE.

Zhe Li is currently a senior R&D engineer in the Database Department of Alibaba Cloud. Specifically, he focuses on the storage layer of AnalyticDB, a commercial cloud-oriented data warehouse. He received his Ph.D. degree in computer science from The Hong Kong Polytechnic University in 2022, supervised by Dr. Ken Yiu. He used to be a member of PolyU Database Research Group. Before that, he obtained his MSc degree in computer science from The Hong Kong University of Science and Technology and BEng degree in computer science from Sun Yat-sen University.

Leong Hou U completed his B.Sc. in Computer Science and Information Engineering at Taiwan Chi Nan University, his M.Sc. in E-commerce at University of Macau, and his Ph.D. in Computer Science at University of Hong Kong. He is now an Associate Professor at University of Macau. His research interests include spatial and spatio-temporal databases, advanced query processing, crowdsourced query processing, information retrieval, data mining and optimization problems.

Reynold Cheng is a professor of the Department of Computer Science in the University of Hong Kong (HKU). He obtained his PhD from Department of Computer Science of Purdue University in 2005. Dr. Cheng was granted an Outstanding Young Researcher Award 2011-12 by HKU.