ENHANCING CLASSIFICATION PERFORMANCE WITH TRUNCATED GENERALIZED PINBALL LOSS IN SUPPORT VECTOR MACHINES AND EFFICIENT OPTIMIZATION TECHNIQUES

SIWAKON SUPPALAP

A Thesis Submitted to the Graduate School of Naresuan University
in Partial Fulfillment of the Requirements
for the Doctor of Philosophy Degree in Mathematics
March 2025

This thesis entitled "Enhancing classification performance with truncated generalized pinball loss in support vector machines and efficient optimization techniques"

by Siwakon Suppalap

has been approved by the Graduate School as partial fulfillment of the requirements for the Doctor of Philosophy Degree in Mathematics of Naresuan University

Oral Defense Committee

..... Chair

(Professor Poom Kumam, Ph.D.)

..... Advisor

(Professor Rabian Wangkeeree, Ph.D.)

..... Internal Examiner

(Associate Professor Kasamsuk Ungchittrakool, Ph.D.)

..... Internal Examiner

(Associate Professor Rattanaporn Wangkeeree, Ph.D.)

..... External Examiner

(Assistant Professor Narit Hnoohom, Ph.D.)

Approved

.....

(Associate Professor Krongkarn Chootip, Ph.D.)

Dean of the Graduate School

14 MAR 2025

ACKNOWLEDGEMENT

I would like to express my deepest appreciation to my dissertation advisor, Professor Dr. Rabian Wangkeeree, for the initial idea, guidance, invaluable help, perfect supervision and encouragement that enabled me to complete my study successfully. Moreover, I want to thank my advisor for his teaching and for giving me good advice. This thesis would not have been completed without all the support that I have always received from him.

Furthermore, I would like to thank you Development and Promotion of Science and Technology Talents Project (DPST), Thai government scholarship for cost support throughout my graduate study, and I want to thank the Department of Mathematics, Faculty of Science, Naresuan University, for being helpful in providing facilities and materials for my thesis experiment.

A special thanks to my thesis committee: Professor Dr. Poom Kumam, Associate Professor Dr. Kasamsuk Ungchittrakoo, Associate Professor Dr. Rattanaporn Wangkeeree, and Assistant Professor Dr. Narit Hnoohom. I am pleased to express my gratitude to all of them for their generous assistance.

Finally, I gratefully appreciate my parents, my friends, and my graduate student family for their love, suggestions, support, and encouragement for me to go on with my Ph.D. studies.

Siwakon Suppalap

Title	ENHANCING CLASSIFICATION PERFORMANCE WITH TRUNCATED GENERALIZED PINBALL LOSS IN SUPPORT VECTOR MACHINES AND EFFICIENT OPTIMIZATION TECHNIQUES
Author	Siwakon Suppalap
Advisor	Professor Rabian Wangkeeree, Ph.D.
Academic Paper	Ph.D. Dissertation in Mathematics, Naresuan University, 2024.
Keywords	Support vector machine, Generalized Pinball Loss, Difference of Convex Functions Programming, Sparsity, Smooth Generalized Pinball Loss, Rescaled Pinball Loss, Modified BFGS Methods, Convex Optimization Problems, Nonconvex Optimization Problems, Nesterov Acceleration.

ABSTRACT

This thesis introduces loss functions in support vector machines that enhance flexibility while reducing the impact of outliers, thereby improving classification performance through several key contributions. Additionally, it also develops optimization techniques to effectively solve these problems.

First, to address the limitations of pinball loss SVMs (Pin-SVM) in handling noise sensitivity, instability, and lack of sparsity, the generalized pinball loss SVM (GP-SVM) was developed. While GP-SVM improves sparsity, its unbounded loss function reduces robustness to outliers. To overcome this, we propose a robust SVM with an asymmetric bounded loss function, the asymmetric truncated generalized pinball loss (ATGP-SVM). The model balances generalization and sparsity while reducing the impact of outliers. Solving the non-convex ATGP-SVM is achieved using DC (difference of convex functions) programming and the DC algorithm (DCA). Experimental results confirm its superior classification performance compared to existing models. However, its non-differentiability limits the optimization methods that can be employed.

Second, we introduce a smooth rescaled generalized pinball loss (SRGP) to further enhance robustness to handle outliers and ensure differentiability. This loss, characterized by asymmetry, non-convexity, sparsity, boundedness, and differentia-

bility, is applied in SRGP-SVM. Given that SRGP-SVM involves a differentiable non-convex optimization problem, we employ modified Broyden Fletcher Goldfarb Shanno (BFGS) method for solving SRGP-SVM to leverage its differentiability. SRGP-SVM demonstrates improved performance on diverse datasets.

Finally, while BFGS methods are widely used for convex and nonconvex optimization, they face challenges such as near-singularity, high computational costs, and inefficiency in large-scale problems. To address these, we propose a regularized stochastic BFGS method integrating Nesterov acceleration with an adaptive momentum coefficient. This coefficient adjusts dynamically based on selected dataset samples, preventing overshooting and reducing computational costs. We provide a theoretical analysis of its convergence and computational complexity. Numerical results on classification problems using SVMs demonstrate superior performance compared to existing approaches.

These contributions improve the efficiency, robustness, and applicability of support vector machines and optimization techniques, leading to further advancements.

LIST OF CONTENTS

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF TABLES (CONT.)

Table **Page**

LIST OF TABLES (CONT.)

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES (CONT.)

Figure **Page**

CHAPTER I

INTRODUCTION

The support vector machine (SVM) stands out as a premier supervised learning model equipped with specialized algorithms crafted for various tasks [1–25]. The SVM has attracted considerable attention due to its impressive performance in practical scenarios and its strong theoretical underpinnings. The core concept of the traditional support vector machine (CSVM) [26] revolves around identifying a hyperplane within the feature space that optimally separates different categories of observations, maximizing the margin between them. This approach utilizes the hinge loss function to enhance the distance between the closest points belonging to distinct classes, a characteristic that is inherently susceptible to noise. To overcome this drawback, the SVM model employing the pinball loss function (Pin-SVM) was introduced, aiming to alleviate sensitivity to noise and instability in re-sampling procedures. The pinball loss is based on quantile distance [27]. The Pin-SVM aims to maximize the quantile distance rather than minimizing the distance between two classes. This approach leads to lower sensitivity to noise but also entails a loss of sparsity.

Nevertheless, the sparsity of the Pin-SVM loss requires correction. In order to achieve sparsity, Sharma et al. [27] proposed a the ϵ-insensitive zone within the Pin-SVM (ϵ-Pin-SVM). This modification ensures that the loss in the insensitive zone is zero. Therefore, the ϵ-Pin-SVM is expected to yield a comparatively sparser loss and reduce computational costs because many components are zero, illustrating its sparsity characteristic. While ϵ-Pin-SVM enhances the sparsity of Pin-SVM, its reliance on a single value of ϵ specified within the insensitive zone may limit its performance. Inspired by these advancements, Reshma et al. [28] introduced the (ϵ_1, ϵ_2)-insensitive zone Pin-SVM, termed as the generalized pinball loss SVM (GP-SVM). This GP-SVM is resilient to noise and offers greater sparsity flexibility than the ϵ-Pin-SVM, as it controls the insensitive zone with two values. Notably, the generalized pinball loss encompasses various prevalent SVM-type models within its framework. Conversely, in many tasks, ensuring resilience against outliers remains a consistently important challenge. In real-world scenarios, samples frequently suf-

fer from a variety of noises beyond mere feature noise. These noisy samples often contain outliers, which can arise from errors in sampling or measurement. Feature noise typically denotes minor perturbations, whereas outliers deviate significantly from other samples, often due to incorrect labeling. Additionally, all the aforementioned SVM models are highly sensitive to outliers due to the unbounded nature of their loss functions. Their sensitivity to outliers can partially obstruct the progress of bounded loss functions, thus restricting their development.

The development of bounded functions has been developed along with the unbounded functions. Initially, Wu et al. [29] introduced an SVM methodology that involves truncating the hinge loss, ensuring that the loss function retains its essential properties. Naturally, it retains the same drawback as the original hinge loss function, remaining sensitive to noise. As a result, a truncated pinball loss function, referenced in [30, 31], was devised to address the issue of noise insensitivity while ensuring boundedness. It then evolved into a truncated ϵ-insensitive pinball loss function, derived from the ϵ-insensitive pinball loss function, to address sparsity more effectively. Since then, significant advancements have occurred in this category of boundary loss functions. Recently, the asymmetric truncated ϵ-insensitive pinball loss function [32] and the truncated generalized pinball loss function [33] have emerged in support vector regression (SVR) applications for regression tasks. Notably, the asymmetric truncated ϵ-insensitive pinball loss function can operate independently of both boundaries of the function. However, defining the insensitive zone using a single ϵ value may restrict the ability to achieve sparsity. While the truncated generalized pinball loss function offers greater flexibility in defining the (ϵ_1, ϵ_2)-insensitive zone, it still has a limitation in that the boundaries on both sides can only be defined by a single value, restricting the function's flexibility. Inspired by the advantages of asymmetric truncated ϵ-insensitive pinball loss function and truncated generalized pinball loss function, we intend to develop a novel bounded loss function that offers greater flexibility. Specifically, we propose an asymmetric truncated generalized pinball loss function that allows flexibility in defining both the boundaries and insensitive zones, serving as a general form of both loss functions. Then, we present a robust SVM with the asymmetric truncated generalized pinball function (ATGP-SVM) explicitly incorporating outliers suppression in the training process. It can be seen that ATGP-SVM can handle noise without losing

sparsity and can also freely set boundaries to handle outliers, which makes it very flexible.

However, defining the function boundary causes those original loss functions that were convex to become non-convex. The resulting optimization problem of ATGP-SVM is non-convex and generally challenging to solve using classical convex optimization techniques. However, we observe that ATGP-SVM offers an advantage in that it can be formulated as an optimization problem involving the DC (difference of two convex functions). Therefore, the DC algorithm (DCA) [34–36] can be applied to solve the ATGP-SVM. The DCA has been successfully employed to tackle various non-differentiable and non-convex optimization problems, often yielding global solutions and demonstrating greater robustness and efficiency.

Although the approach to creating bounded loss functions typically involves using truncated functions, these have been developed to effectively handle outliers through DC functions, which can be optimized using the DCA. However, these loss functions often exhibit abrupt behavior. Once the loss reaches a certain truncated threshold, it immediately becomes constant rather than transitioning smoothly, which is too aggressive. Consequently, the gradient of the loss function becomes zero when the function flattens out, posing challenges for first-order and second-order optimization techniques that rely on gradient information. To address this issue, the rescaling technique is employed. The rescaling technique [62, 63] is particularly popular due to its simplicity, allowing for the creation of smoothly scaled bounded functions while preserving the key properties of the original loss function. This technique has been applied to hinge loss, resulting in the development of the rescaled hinge loss in SVM (RH-SVM) [64] to limit the original hinge loss. However, it still remained sensitive to noise, similar to the original. To address this limitation, the rescaled pinball loss in SVM (RP-SVM) was introduced [65], which aimed to reduce noise sensitivity while retaining the bounded characteristics of the pinball loss. On the other hand, since the loss functions in all the previously discussed SVMs are non-differentiable, the corresponding unconstrained minimization problems in SVMs are also non-differentiable. Consequently, several first-order and second-order optimization methods [57], which are effective and grounded in solid theoretical foundations, may not be directly applicable to such problems. To overcome the limitations of the hinge loss in SVM, the smooth hinge loss in

SVM (SH-SVM) was introduced [58]. Similarly, the smooth pinball loss in SVM (SP-SVM) was proposed [59] to address the drawbacks of pinball loss. Building on these advancements, the smooth generalized pinball loss in SVM (SGP-SVM) was presented [60], offering a versatile framework that encompasses both SH-SVM and SP-SVM. These smooth loss functions are continuous and differentiable, making the corresponding unconstrained minimization problems in SVM differentiable as well, thus allowing the direct application of first-order and second-order optimization methods. Despite these advancements, the rescaling technique has yet to be applied to the smooth generalized pinball loss in SGP-SVM, which results in difficulties in handling outliers.

Therefore, we propose a new approach to integrate the rescaling technique with the smooth generalized pinball loss in SGP-SVM, enhancing its ability to handle outliers. Building on this, we develop a robust SVM model incorporating the smooth rescaled generalized pinball loss function, referred to as SRGP-SVM. This approach effectively handles noise, maintains sparsity, provides boundedness to manage outliers, and offers greater flexibility in parameter adjustment, while ensuring the differentiability of the objective functions. The differentiability of SRGP-SVM results in an unconstrained minimization problem, which can be efficiently addressed using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [66], a well-established second-order optimization technique that utilizes an approximate Hessian matrix update to improve its effectiveness in solving differentiable optimization problems. In many cases, BFGS outperforms gradient descent (GD), a widely used first-order optimization method [57]. Since SRGP-SVM constitutes a non-convex unconstrained minimization problem, the classical BFGS method, which is formulated for convex optimization, is not directly applicable. To address this limitation, we adopt a modified variant of BFGS, namely the modified BFGS method (M-BFGS), as presented in [67]. Nevertheless, these techniques still face challenges in large-scale problems, including high computational costs and inefficiency. To address these challenges, stochastic optimization techniques are often employed.

Stochastic optimization problems have attracted attention for their broad applications in machine learning [73–76], object detection [77–79], wireless communication [80], mixed logit modeling [81–83], and stock market forecasting [84–86]. A commonly used approach to solve these problems is stochastic gradient descent

(SGD) [87], which is stochastic variants of the GD method and particularly effective for stochastic convex optimization. By leveraging gradient information from sample data, SGD determines the search direction, providing a simple yet effective solution. SGD is guaranteed to converge to a minimum when using an unbiased stochastic gradient estimate and appropriate step sizes. However, it often requires many iterations to reach convergence. Consequently, SGD-based optimization techniques have become a central focus of recent research. Despite its effectiveness, SGD may struggle with ill-conditioned problems [88]. Consequently, research has increasingly focused on stochastic variants of the BFGS method, such as oBFGS [90, 91]. These methods adapt the classical BFGS approach to the stochastic setting by incorporating gradient estimates computed from randomly selected subsets of data, rather than the full dataset, which reduces computational cost. Although the oBFGS method theoretically ensures that the approximate Hessian matrix remains positive definite, meaning all eigenvalues are greater than zero, in practical computations, the smallest eigenvalue can approach zero. This situation leads to near-singularity issues when inverting the approximate Hessian matrix. To address this issue, a regularization term is incorporated into the calculation of the approximate Hessian matrix, which helps prevent the eigenvalues from becoming excessively small and alleviates the near-singularity problem. This approach led to the development of the regularized stochastic BFGS method (RES) [92]. Additionally, another approach to enhance the practical efficiency of the oBFGS method has garnered attention. One notable method is the stochastic Nesterov-accelerated quasi-Newton method (oNAQ) [93], which incorporates Nesterov's momentum with a constant momentum coefficient to improve the performance of oBFGS methods. Experimental results have shown that oNAQ can outperform both the oBFGS method and the first-order methods like SGD. Nevertheless, oNAQ still faces certain practical challenges. The first challenge is the near-singularity of the approximate Hessian matrix, similar to the issue encountered in oBFGS. The second challenge stems from applying momentum in every iteration. While momentum is intended to speed up convergence, there are cases where its application may lead to updates that are less favorable in direction compared to those without momentum. To address these challenges, the regularized stochastic Nesterov-accelerated quasi-Newton method (RES-NAQ) [94] was introduced. This method integrates the RES approach with Nesterov's ac-

celeration technique and incorporates an adaptive momentum coefficient. This coefficient dynamically adjusts between a constant value and zero based on the objective function's value. If the momentum fails to decrease the objective function, it is automatically deactivated. As a result, RES-NAQ can flexibly alternate between the characteristics of RES and oNAQ.

However, the constant value in the adaptive momentum coefficient of the RES-NAQ method lacks flexibility, potentially leading to overshooting issues, and evaluating the objective function on a large dataset results in significant computational costs. Moreover, addressing nonconvex optimization problems using stochastic quasi-Newton methods remains a challenging task. A framework for stochastic quasi-Newton methods applied to nonconvex stochastic optimization was proposed, and its complexity was analyzed under conditions related to step size and method output in [95,96]. Therefore, we introduced a regularized stochastic BFGS method for nonconvex optimization, which integrates Nesterov acceleration with a new adaptive momentum coefficient. This coefficient dynamically adjusts between a reduced momentum value and zero based on the objective function value from selected dataset samples, mitigating overshooting and lowering computational costs. We theoretically analyze its convergence and complexity and validate its effectiveness in both convex and nonconvex classification problems through extensive experiments with support vector machines.

This thesis is organized in the following way.

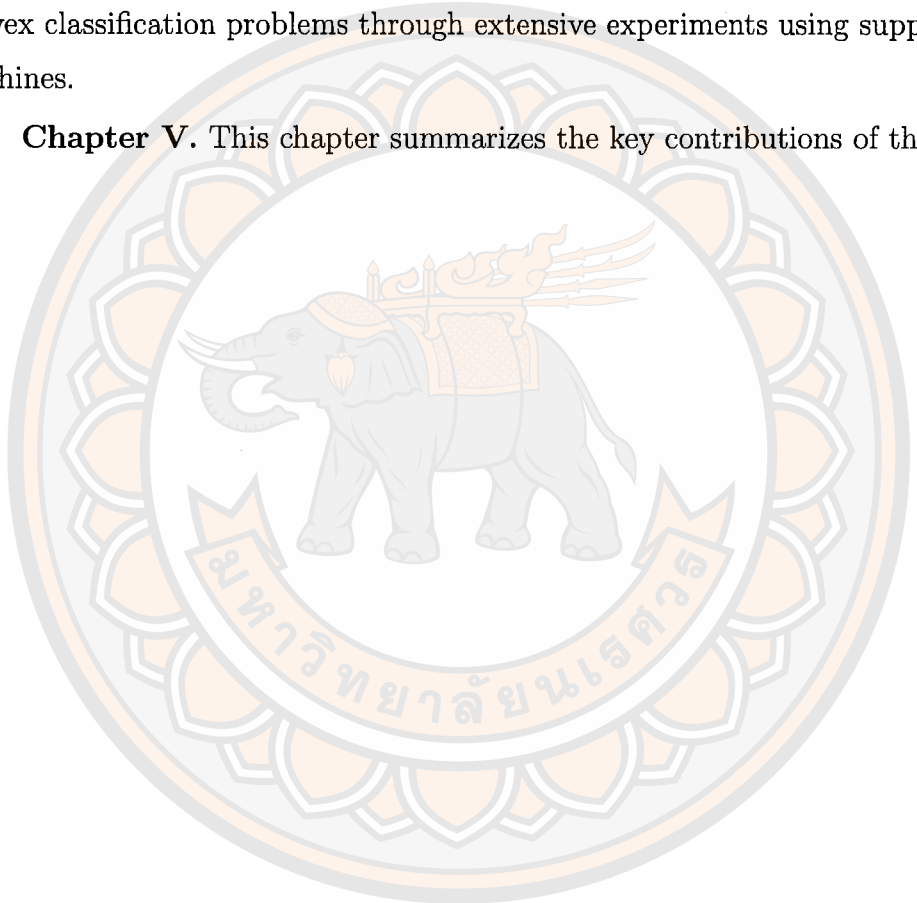
Chapter II. We review fundamental definitions, lemmas, and theorems relevant to the problems addressed within our framework. Additionally, we conduct a comprehensive review of prior research on support vector machines, DCA, and first-order and second-order optimization methods, establishing a robust foundation for our work.

Chapter III. We propose ATGP-SVM, a support vector machine with the asymmetric truncated generalized pinball loss function, optimized via DCA. This model incorporates asymmetry, boundedness, and non-convexity to address noise, sparsity, and outliers. Additionally, we introduce SRGP-SVM, which employs the smooth rescaled generalized pinball loss function, emphasizing its continuity, differentiability, and parameter adaptability. SRGP-SVM is optimized using a modi-

fied BFGS method. Extensive numerical experiments on synthetic and benchmark datasets validate the effectiveness of our approach in noisy environments.

Chapter IV. We propose a regularized stochastic BFGS method for non-convex optimization, combining Nesterov acceleration with an adaptive momentum coefficient. This method adjusts the coefficient flexibly to enhance convergence and reduce computational costs. We establish its convergence and complexity analysis theoretically and demonstrate its superior performance in both convex and non-convex classification problems through extensive experiments using support vector machines.

Chapter V. This chapter summarizes the key contributions of the thesis



CHAPTER II

PRELIMINARIES

Throughout this dissertation, we focus on binary classification within n-dimensional Euclidean space $\mathbb{R}^n$. All vectors are treated as column vectors, with the option to transpose them into row vectors denoted by the superscript $\top$. For vectors $\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$ and $\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$ in $\mathbb{R}^n$, the induced norm (Euclidean norm) of $\mathbf{x}$ is defined as $\|\mathbf{x}\| = \sqrt{\mathbf{x}^\top \mathbf{x}} = \sqrt{\sum_{i=1}^n x_i^2}$, and the scalar product of $\mathbf{x}$ and $\mathbf{y}$ is expressed as $\mathbf{x}^\top \mathbf{y} = \sum_{i=1}^n x_i y_i$.

2.1 Mathematical background

Definition 2.1.1. [104] Let A be an $n \times n$ real square matrix. A nonzero vector $\mathbf{x}$ is called an **eigenvector** of A if there exists a scalar λ such that $A\mathbf{x} = \lambda\mathbf{x}$. The scalar λ is called an **eigenvalue** corresponding to eigenvector $\mathbf{x}$.

Definition 2.1.2. [104] A **symmetric matrix** is a matrix A such that $A^\top = A$.

Definition 2.1.3. [104] A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called **positive semidefinite**, denoted by $\mathbf{A} \succeq 0$, if $\mathbf{x}^\top \mathbf{A}\mathbf{x} \geq 0$ for every $\mathbf{x} \in \mathbb{R}^n$. A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called **positive definite**, denoted by $\mathbf{A} \succ 0$, if $\mathbf{x}^\top \mathbf{A}\mathbf{x} > 0$ for every $0 \neq \mathbf{x} \in \mathbb{R}^n$.

Theorem 2.1.4. [104] A symmetric matrix A is positive definite (or positive semidefinite) if and only if all the eigenvalues of A are positive (or nonnegative).

Definition 2.1.5 (Spectral norm). [109] Let $\mathbf{A}$ be an $(m \times n)$-matrix. Then

$$\|\mathbf{A}\| := \max_{\mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|$$

is the 2-norm (or spectral norm) of $\mathbf{A}$.

In other words, the spectral norm is the largest factor by which a vector can be stretched in length under the mapping $\mathbf{x} \to \mathbf{A}\mathbf{x}$. Note that as a simple consequence,

$$\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\|\|\mathbf{x}\|, \forall \mathbf{x}.$$

Proposition 2.1.6 (Rayleigh's Inequalities). [105] If an $n \times n$ matrix $\mathbf{A}$ is real symmetric positive definite, then

$$\lambda_{\min}(\mathbf{A})\|\mathbf{x}\|^2 < \mathbf{x}^\top \mathbf{A}\mathbf{x} < \lambda_{\max}(\mathbf{A})\|\mathbf{x}\|^2,$$

where $\lambda_{\min}$ denotes the smallest eigenvalue of $\mathbf{A}$, and $\lambda_{\max}(\mathbf{A})$ denotes the largest eigenvalue of $\mathbf{A}$.

Now, let us delve into the fundamental concepts of convexity and the valuable characterizations of convex functions.

Definition 2.1.7. [106] A set $C \subseteq \mathbb{R}^n$ is called a **convex set** if for any $\mathbf{x}_1, \mathbf{x}_2 \in S$ and any $\lambda \in (0, 1)$, we have

$$\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in C.$$

Definition 2.1.8. [106] A function $f : C \to \mathbb{R}$ defined on a convex set $C \subseteq \mathbb{R}^n$ is **convex function** if and only if for all $\mathbf{x}_1, \mathbf{x}_2 \in C$ and all $\lambda \in (0, 1)$, we have

$$f((1-\lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2) \leq (1-\lambda)f(\mathbf{x}_1) + \lambda f(\mathbf{x}_2).$$

A function that does not satisfy above inequality is referred to as a **non-convex function**.

Next, we move on to a priori knowledge of a real-valued function that involves the function's differentiability and convexity. We begin with by recalling the partial derivative.

Definition 2.1.9. [106] Let $f : \mathbb{R}^n \to \mathbb{R}$ be some function, fix some $\mathbf{x} \in \mathbb{R}^n$. If the limit

$$\lim_{d \to 0^+} \frac{f(\mathbf{x} + d\mathbf{e}_i) - f(\mathbf{x})}{d}$$

exists, where $\mathbf{e}_i$ is the i-th unit vector, then it is called the i-th **partial derivative** of f at the vector $\mathbf{x}$ and and is denoted by $\frac{\partial f(\mathbf{x})}{\partial x_i}$.

Definition 2.1.10. [106] Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function and $\mathbf{x} \in \mathbb{R}^n$. Then $\mathbf{g} \in \mathbb{R}^n$ is said to be the **subgradient** of the function f at $\mathbf{x}$ if

$$f(\mathbf{z}) \geq f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{z} - \mathbf{x}), \quad \forall \mathbf{z} \in \mathbb{R}^n.$$

The set of subgradients of f at $\mathbf{x}$ is called the **subdifferential** of f at $\mathbf{x}$ and denoted $\partial f(\mathbf{x})$.

Definition 2.1.11. [106] Let $f : \mathbb{R}^n \to \mathbb{R}$ be a differentiable function. The function denoted by $\nabla f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ is said to be the **gradient** of the function f if

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right]^\top.$$

If f is continuously differentiable, the map $\mathbf{x} \mapsto \nabla f(\mathbf{x})$ is continuous over $\mathbb{R}^n$, then f is called a smooth function.

Next, given $f : \mathbb{R}^n \to \mathbb{R}$, if ∇f is differentiable, we say that f is twice differentiable, and we write the derivative of ∇f as

$$\nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{pmatrix}$$

The matrix $\nabla^2 f(\mathbf{x})$ is called the **Hessian matrix** of f at $\mathbf{x}$, and is often also denoted by $\mathbf{H}$.

Definition 2.1.12. [106] Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function and $\mathbf{x} \in \mathbb{R}^n$. Then $\mathbf{g} \in \mathbb{R}^n$ is said to be the **subgradient** of the function f at $\mathbf{x}$ if

$$f(\mathbf{z}) \geq f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{z} - \mathbf{x}), \quad \forall \mathbf{z} \in \mathbb{R}^n.$$

The set of subgradients of f at $\mathbf{x}$ is called the **subdifferential** of f at $\mathbf{x}$ and denoted $\partial f(\mathbf{x})$.

In optimization and mathematical analysis, critical points and stationary points are essential for understanding the behavior of functions. The following definitions offer clear mathematical descriptions of these points.

Definition 2.1.13. [106] A point $\mathbf{x}^* \in \mathbb{R}^n$ is called a **critical point** of a function $f : \mathbb{R}^n \to \mathbb{R}$ if

$$\nabla f(\mathbf{x}^*) = 0 \quad \text{or} \quad \nabla f(\mathbf{x}^*) \text{ does not exist.}$$

A **stationary point** is a particular type of critical point where the gradient is specifically zero. The following theorem provides the first-order necessary condition for stationary points:

Theorem 2.1.14 (First-order necessary condition). Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function. If $\mathbf{x}^*$ is a stationary point of f, then

$$\nabla f(\mathbf{x}^*) = 0.$$

This distinction between critical and stationary points is crucial for characterizing the behavior of a function and identifying potential optimization solutions.

Next, we introduce the order symbols O and o to discuss this property further [105]. Let g be a real-valued function defined in some neighbourhood of $\mathbf{0} \in \mathbb{R}^n$, with $g(\mathbf{x}) \neq \mathbf{0}$ if $\mathbf{x} \neq \mathbf{0}$. Let $f : S \to \mathbb{R}^m$ be defined in a domain $S \subset \mathbb{R}^n$ that includes $\mathbf{0}$. Then, we write

- $f(\mathbf{x}) = O(g(\mathbf{x}))$ to mean that the quotient $\|f(\mathbf{x})\|/|g(\mathbf{x})|$ is bounded near $\mathbf{0}$; that is, there exist numbers $K > 0$ and $\delta > 0$ such that if $\|\mathbf{x}\| < \delta$, $\mathbf{x} \in S$, then $\|f(\mathbf{x})\|/|g(\mathbf{x})| \leq K$. The symbol $O(g(\mathbf{x}))$ is used to represent a function bounded by a scaled version of g in a neighbourhood of 0.
- $f(\mathbf{x}) = o(g(\mathbf{x}))$ to mean that

  $$\lim_{\mathbf{x}\to\mathbf{0},\mathbf{x}\in S} \frac{\|f(\mathbf{x})\|}{|g(\mathbf{x})|} = 0.$$

  On the other hand, $o(g(\mathbf{x}))$ represents a function that goes to zero faster than $g(\mathbf{x})$ in the sense that $\lim_{\mathbf{x}\to\mathbf{0}} \|o(g(\mathbf{x}))\|/|g(\mathbf{x})| = 0$.

2.2 Statistical background

Next, we examine the statistical theories underlying our work, providing insights into the methodologies that inform our analysis. A key concept in this discussion is the probability space, which serves as a foundation for understanding random processes and comprises three main components: the sample space, the set of events, and the probability function. To clarify these components, we introduce the following definitions and theories.

Definition 2.2.1. [107] A sample space Ω is the set of all possible outcomes from an experiment. We denote ξ as an element in Ω.

Definition 2.2.2. [107] An event E is a subset of the sample space Ω. The set of all possible events is denoted as $\mathcal{F}$. If $\mathcal{F}$ is closed under complements, countable unions and contains the empty set $\emptyset$, then $\mathcal{F}$ is called a σ-field (or σ-algebra).

Definition 2.2.3. [107] A probability function is a function $P : \mathcal{F} \to [0, 1]$ of an event E to a real number in $[0, 1]$.

Building on the foundational concepts, we now explore more advanced topics in probability theory, including the idea of events occurring almost surely, the definition and properties of random variables, and the mathematical expectations associated with these variables.

Definition 2.2.4 (Almost Surely). [107] An event E in a probability space $(\Omega, \mathcal{F}, P)$ is said to hold almost surely (a.s.) if $P(E) = 1$.

This definition highlights the difference between events that are certain to occur and those that are merely likely, a crucial aspect in probabilistic modeling. A key development in probability theory is the introduction of random variables, which link outcomes to numerical values.

Definition 2.2.5. [107] A random variable X is a function $X : \Omega \to \mathbb{R}$ that maps an outcome $\xi \in \Omega$ to a number $X(\xi)$ on the real line.

Random variables enable the modeling and analysis of randomness in numerical terms, making them essential tools in statistics. We now turn to the concept of expectation, which is fundamental in understanding the average outcome of random processes.

Definition 2.2.6. [107] The expectation (or expected value) of a random variable X is defined as follows:

- For a discrete random variable X, the expectation is given by

$$\mathbb{E}[X] = \sum_{x \in X(\Omega)} x \cdot p_X(x),$$

where $X(\Omega)$ is the set of all possible values of X, and $p_X(x)$ is probability mass function of X.

- For a continuous random variable X, the expectation is given by

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x \cdot f_X(x)\, dx,$$

where $f_X(x)$ is the probability density function of X.

In our thesis, the notation $\mathbb{E}_X[\cdot]$ refers to the expectation taken with respect to the random variable X, while $\mathbb{E}[\cdot]$ denotes the expectation with respect to the distribution of a stochastic process. The expectation of a random variable has several important properties, which are outlined below. These properties apply to both discrete and continuous random variables.

Theorem 2.2.7. [107] The expectation of a random variable X has the following properties:

(E1) For any function g, $\mathbb{E}[g(X)] = \sum_{x \in X(\Omega)} g(x) pX(x)$;

(E2) For any function g and h, $\mathbb{E}[g(X) + h(X)] = \mathbb{E}[g(X)] + \mathbb{E}[h(X)]$;

(E3) For any constant c, $\mathbb{E}[cX] = c\mathbb{E}[X]$;

(E4) For any constant c, $\mathbb{E}[X + c] = \mathbb{E}[X] + c$;

(E5) $\mathbb{E}[X] \leq \mathbb{E}[Y]$ if $X \leq Y$ (a.s.);

(E6) $\mathbb{E}[\mathbb{E}[X]] = \mathbb{E}[X]$.

Definition 2.2.8. [107] The conditional expectation of a random variable X given $Y = y$, denoted as $\mathbb{E}[X|Y = y]$, is defined as follows:

1. For discrete random variables X and Y, the conditional expectation is given by

$$\mathbb{E}[X|Y = y] = \sum_{x \in X(\Omega)} x \cdot p_X(x|Y = y),$$

where $p_X(x|Y = y)$ is the conditional probability mass function of X given $Y = y$.

2. For continuous random variables X and Y, the conditional expectation is given by

$$\mathbb{E}[X|Y=y] = \int_{-\infty}^{\infty} x \cdot f_X(x|Y=y)\, dx,$$

where $f_X(x|Y=y)$ is the conditional probability density function of X given $Y=y$.

Theorem 2.2.9. [110] Let X, Y, Z be random variables, a, b be a constant or a deterministic function independent of Y, and g be a function. Assuming all the following expectations exist, we have

(C1) $\mathbb{E}[a|Y] = a$;

(C2) $\mathbb{E}[aX + bZ|Y] = a\mathbb{E}[X|Y] + b\mathbb{E}[Z|Y]$;

(C3) $\mathbb{E}[X|Y] \geq 0$ if $X \geq 0$;

(C4) $\mathbb{E}[X|Y] = \mathbb{E}[X]$ if X and Y are independent;

(C5) $\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X]$;

(C6) $\mathbb{E}[Xg(Y)|Y] = g(Y)\mathbb{E}[X|Y]$. In particular, $\mathbb{E}[g(Y)|Y] = g(Y)$;

(C7) $\mathbb{E}[X|Y, g(Y)] = \mathbb{E}[X|Y]$.

Next, we state the definition of a supermartingale, and the proposition establishes the convergence of a nonnegative supermartingale.

Definition 2.2.10. [99]. Let $\{\mathcal{F}_k\}$ be an increasing sequence of σ-algebras. If $\{X_k\}$ is a stochastic process satisfying (i) $\mathbb{E}[|X_k|] < \infty$, (ii) $X_k \in \mathcal{F}_k$ for all k, and (iii) $\mathbb{E}[X_{k+1} \mid \mathcal{F}_k] \leq X_k$ for all k, then $\{X_k\}$ is called a supermartingale.

Proposition 2.2.11. *(Theorem 5.2.9 in [99]). If $\{X_k\}$ is a nonnegative supermartingale, then $\lim_{k\to\infty} X_k \to X$ almost surely and $\mathbb{E}[X] \leq \mathbb{E}[X_0]$.*

2.3 Support vector machine

Let us now discuss the support vector machine to deal with the binary classification problem. We denote the training dataset is defined as $D = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times$

$\{1, -1\}|i = 1, 2, 3, \ldots, m\}$. In the SVM model, our goal is to identify a separating hyperplane represented by $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b = 0, \mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$; with the objective of maximizing the margin while minimizing the training error. The optimization problem can be defined as

$$\begin{aligned} \max_{\mathbf{w},b} \quad & \frac{2}{\|\mathbf{w}\|} \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad \text{for } i = 1, \ldots, m, \end{aligned}$$

or

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad \text{for } i = 1, \ldots, m. \end{aligned} \tag{2.3.1}$$

For linearly non-separable data, we introduce a non-negative variable called slack variable ξ_i to the objective function resulting in changing the primal problem (2.3.1) into

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i, \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \ldots, m, \end{aligned} \tag{2.3.2}$$

where $\boldsymbol{\xi} = (\xi_1, ..., \xi_m)^\top$, and $C > 0$ is penalty parameter.

In the framework of machine learning, the concept of kernels plays an essential role in enhancing the expressiveness and flexibility of algorithms, particularly in the context of support vector machine (SVM) and related techniques. A kernel can be viewed as a mathematical function that implicitly transforms data into a higher-dimensional space, allowing algorithms to capture intricate patterns and relationships that may not be discernible in the original feature space.

Definition 2.3.1. A function $K : X \times X \to \mathbb{R}$ is called a kernel over X, if

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle,$$

for any $\mathbf{x}, \mathbf{y} \in X$ and for some function ϕ from X to a higher dimensional feature space.

The following example gives kernel function commonly used in application.

Example 2.3.2. (Linear kernel) The linear kernel is the kernel K defined over $\mathbb{R}^n$ as

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}.$$

Example 2.3.3. (RBF kernel) For any constant $\sigma > 0$, a radial basis function (RBF) kernel or Gaussian kernel is the kernel K defined over $\mathbb{R}^n$ as

$$K(\mathbf{x}, \mathbf{y}) = \exp\Big(- \frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2} \Big).$$

The SVM problem can be rephrased as an unconstrained optimization problem:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} L(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)),$$

where $C > 0$ is a constant parameter, and L denotes the loss function. The decision function is determined by the sign of $\mathbf{w}^\top \mathbf{x} + b$. If this value is positive, $\mathbf{x}$ is classified as belonging to class +1; otherwise, it is classified as belonging to class -1. The essence of SVM lies in its loss function. Among the most popular and widely recognized loss functions is the hinge loss function. The hinge loss function, denoted as $L_{hinge} : \mathbb{R} \to \mathbb{R}$, is defined as follows:

$$L_{hinge}(u) = (u)_+,$$

where $(u)_+ = u$ if $u \geq 0$ and 0 otherwise. The hinge loss function is closely linked to the shortest distance between sets, rendering the classifier sensitive to noise and susceptible to instability during resampling. In response to these drawbacks, Sharma [27] proposes the utilization of a pinball loss function for SVM classification (Pin-SVM). This strategy introduces noise insensitivity by penalizing correctly classified samples. The pinball loss function, denoted as $L_\tau : \mathbb{R} \to \mathbb{R}$, is defined as follows:

$$L_\tau(u) = \begin{cases} u, & u \geq 0, \\ -\tau u, & u < 0, \end{cases}$$

where $\tau \geq 0$ is a hyperparameter (see Figure 1). However, the above formulation achieves noise insensitivity at the expense of sparsity. This is because the pinball

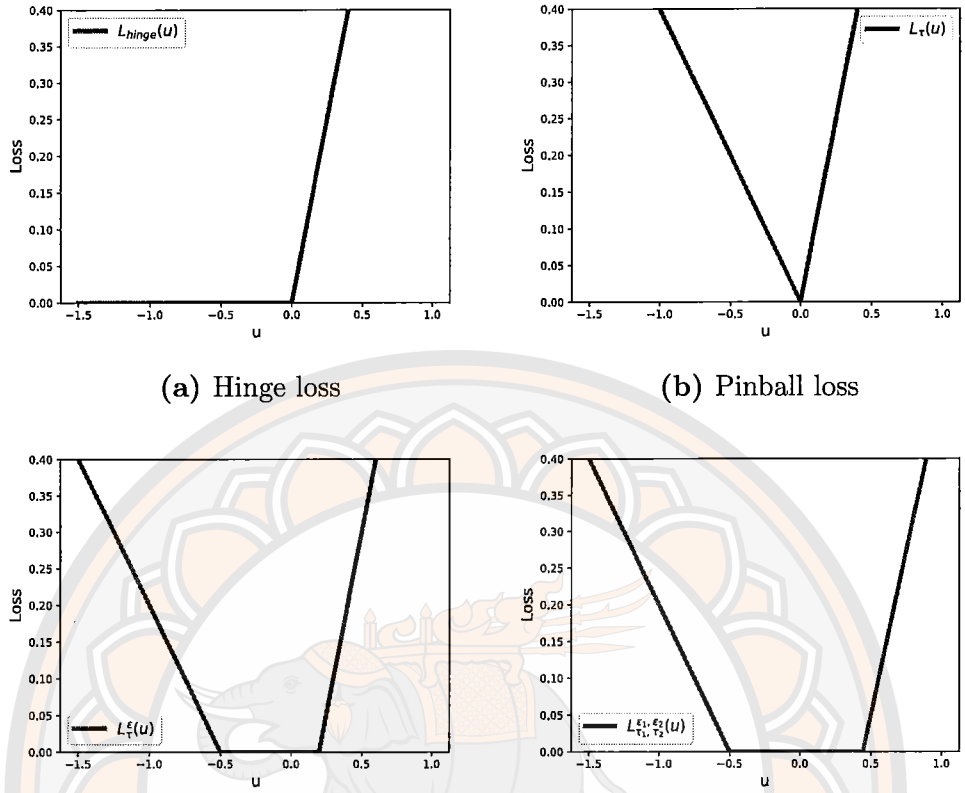
loss function's sub-gradient is non-zero almost everywhere. To promote sparsity, Sharma [27] further introduces an ϵ-insensitive zone into Pin-SVM (referred to as ϵ-insensitive zone pinball SVM). In this setup, the sub-gradient of the loss function becomes zero within the range $[\tau, \epsilon]$, thereby inducing sparsity in the model. The definition of the pinball loss with an ϵ-insensitive zone is as follows:

$$L_\tau^\epsilon(u) = \begin{cases} u - \epsilon, & \text{if } u > \epsilon, \\ 0, & \text{if } -\frac{\epsilon}{\tau} \leq u \leq \epsilon, \\ -\tau\left(u + \frac{\epsilon}{\tau}\right), & \text{if } u < -\frac{\epsilon}{\tau}. \end{cases}$$

In this context, the parameters are $\epsilon \geq 0$ and $\tau \geq 0$. However, in the formulation of the ϵ-insensitive zone pinball SVM, only one parameter, ϵ, requires optimal selection. In response to this limitation, Reshma [28] introduced the concept of ϵ-insensitive zone Pin-SVM, aiming to develop a generalized pinball loss SVM. This approach enables the creation of an asymmetric insensitive zone by allowing variations in ϵ. The generalized pinball loss function offers a broader scope compared to other existing loss functions, addressing concerns related to noise sensitivity and instability during resampling. Below is the provided definition of the generalized pinball loss function:

$$L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(u) = \begin{cases} \tau_1\left(u - \frac{\epsilon_1}{\tau_1}\right), & \text{if } u > \frac{\epsilon_1}{\tau_1}, \\ 0, & \text{if } -\frac{\epsilon_2}{\tau_2} \leq u \leq \frac{\epsilon_1}{\tau_1}, \\ -\tau_2\left(u + \frac{\epsilon_2}{\tau_2}\right), & \text{if } u < -\frac{\epsilon_2}{\tau_2}, \end{cases}$$

where $\tau_1, \tau_2, \epsilon_1, \epsilon_2 \geq 0$ represent user-defined parameters.



(a) Hinge loss **(b)** Pinball loss

(c) Epsilon-insensitive pinball loss **(d)** Generalized pinball loss

Figure 1 Unbounded loss functions with $\tau = 0.4$, $\tau_1 = 0.9$, $\tau_2 = 0.4$, $\epsilon = 0.2$, $\epsilon_1 = 0.4$, **and** $\epsilon_2 = 0.2$.

It is evident that these loss functions exhibit convexity (refer to Figure 1), a characteristic highly favored in numerous publications due to its computational advantages. However, their convex nature also brings about a lack of robustness to outliers, stemming from their unboundedness. This susceptibility can result in the domination of outliers within the corresponding classifier. To address this limitation, one approach is to enforce an upper bound on the loss functions, limiting their escalation beyond a certain threshold. Consequently, the convex loss functions transition into non-convex functions as a consequence of this constraint. In the next section, we discuss some of the non-convex loss functions that are derived from this approach. The first one is the truncated hinge loss function [29], denoted as $L_{th}^{\alpha} : \mathbb{R} \to \mathbb{R}$, defined as:

$$L_{th}^{\alpha}(u) = g_{th}(u) - h_{th}(u),$$

with $g_{th}(u) = L_{hinge}(u)$ and $h_{th}(u) = u$ if $u \geq \alpha$ and 0 otherwise (see Figure 2).

Nevertheless, it still exhibits the same susceptibility to noise issues as the original hinge loss functions. To address this concern, the truncated pinball loss and the asymmetric truncated pinball loss have been introduced. The truncated pinball loss is defined as follows [30]:

$$L_{tp}^{\alpha}(u) = \begin{cases} \tau\alpha, & u \leq -\alpha, \\ -\tau u, & -\alpha \leq u < 0, \\ u, & u \geq 0, \end{cases}$$

where $0 \leq \tau \leq 1$ and $\alpha > 0$ is a given scalar parameter. The Figure 3 demonstrates that the truncated pinball loss maintains a constant value for scores u less than α. This indicates that similar penalties are assigned to most accurately classified samples. Conversely, the asymmetrical truncated pinball loss function is defined as follows [31]:

$$\begin{aligned} L_{tp}^{\alpha_1,\alpha_2}(u) &= g_{tp}(u) - h_{tp}(u) \\ &= \begin{cases} \alpha_1, & u \geq \alpha_1, \\ u, & 0 \leq u < \alpha_1, \\ -\tau u, & -\alpha_2/\tau < u < 0, \\ \alpha_2, & u \leq -\alpha_2/\tau, \end{cases} \end{aligned}$$

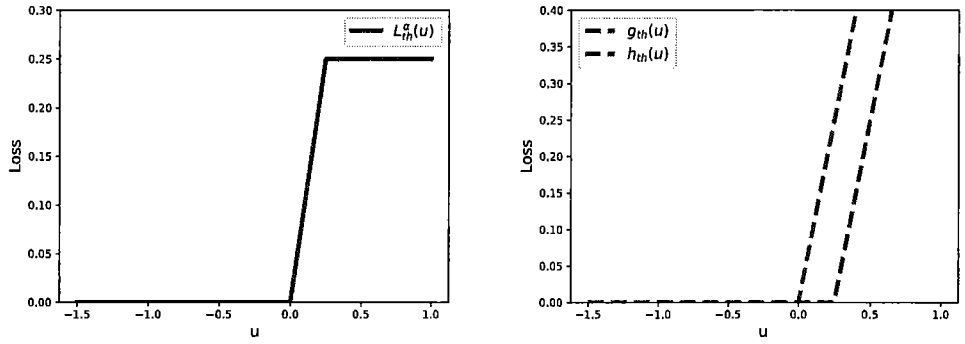
with

$$g_{tp}(u) = L_{\tau}(u) \tag{2.3.3}$$

and

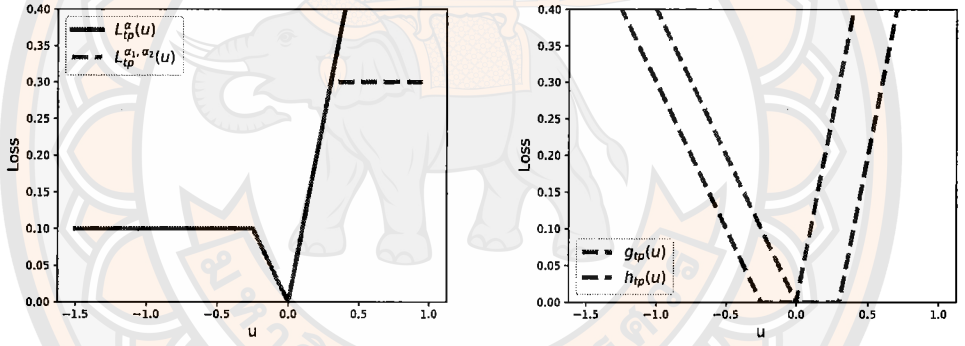
$$h_{tp}(u) = \begin{cases} u - \alpha_1, & u \geq \alpha_1, \\ 0, & -\alpha_2/\tau \leq u < \alpha_1, \\ -\tau u - \alpha_2, & u < -\alpha_2/\tau, \end{cases} \tag{2.3.4}$$

where the parameters are constrained as $0 \leq \tau \leq 1$, $\alpha_1 > 0$, and $\alpha_2 > 0$ (see Figure 3). Unlike truncated pinball loss function, asymmetrical truncated pinball loss function are bounded on both sides, which manage their robustness better.



(a) (b)

Figure 2 Truncated hinge loss and its decomposition functions with $\alpha_2 = 0.25$.



(a) (b)

Figure 3 Truncated pinball loss and asymmetrical truncated pinball loss and their decomposition functions with $\tau = 0.4$, $\alpha = 0.25$, $\alpha_1 = 0.3$, and $\alpha_2 = 0.1$.

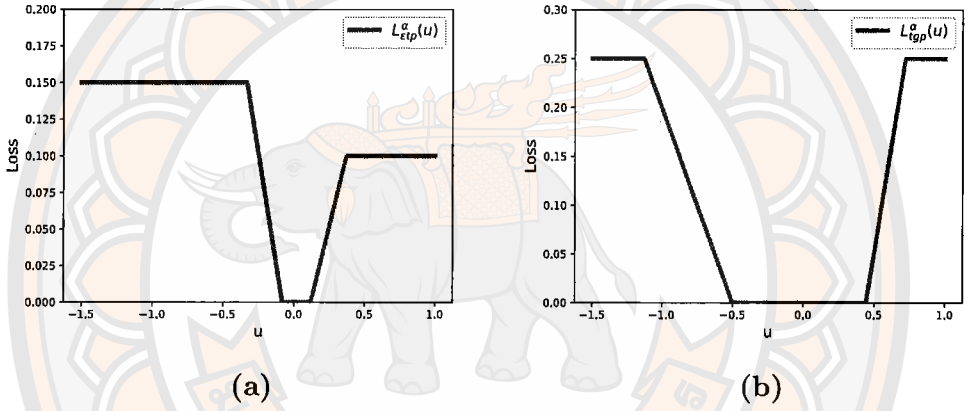
However, truncated pinball loss function still losses sparsity. Below is a definition of a more robust and efficient loss function:

$$L^{\alpha}_{\epsilon tp}(u) = \begin{cases} \tau\alpha, & u \geq \alpha + (1-\tau)\epsilon, \\ \tau(u - (1-\tau)\epsilon), & (1-\tau)\epsilon \leq u \leq \alpha + (1-\tau)\epsilon, \\ 0, & -\tau\epsilon \leq u \leq (1-\tau)\epsilon, \\ -(1-\tau)(u + \tau\epsilon), & -\alpha - \tau\epsilon \leq u \leq -\tau\epsilon, \\ -(1-\tau)(-\alpha), & u \leq -\alpha - \tau\epsilon, \end{cases}$$

where parameters $\tau \in (0,1), \alpha > 0$ (see Figure 4). The aforementioned loss function is referred to as the asymmetric truncated ϵ-insensitive pinball loss function [32]. The $L^{\alpha}_{\epsilon tp}$ exhibits independence from both boundaries of the function. However, limitations arise when determining only a single ϵ-insensitive zone, potentially restricting its effectiveness in managing sparsity. On the other hand, Wang [33] obtain the following truncated generalized pinball loss function:

$$L^{\alpha}_{tgp} = \min\{\alpha, L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}\},$$

where $\alpha > 0$ and $\tau_1, \tau_2, \epsilon_1, \epsilon_2 \geq 0$ are parameters (see Figure 4).

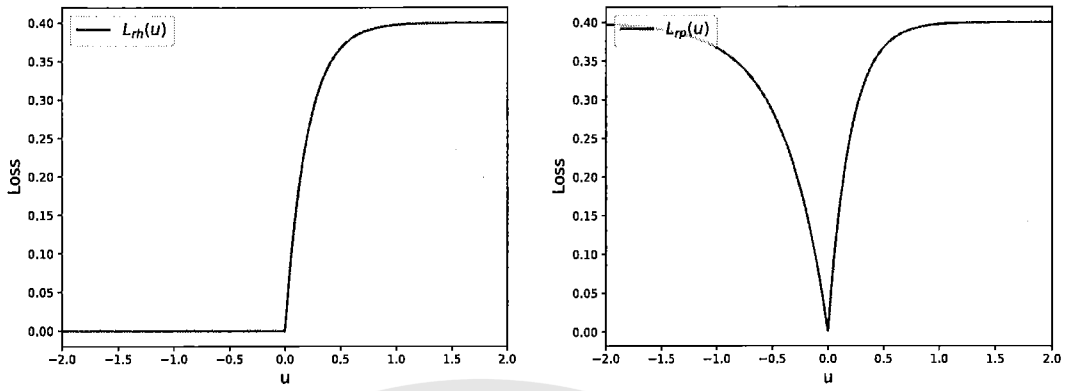


(a) (b)

Figure 4 Asymmetric truncated ϵ-insensitive pinball loss and truncated generalized pinball loss function with $\tau = 0.4$, $\alpha = 0.25$, $\epsilon = 0.2$, $\tau_1 = 0.9$, $\tau_2 = 0.4$, $\epsilon_1 = 0.4$, and $\epsilon_2 = 0.2$.

The truncated generalized pinball loss function can indeed be defined, and it offers a more flexible ϵ-insensitive zone. Unfortunately, a drawback is that the boundaries on both sides of the function can only be determined by a single value, thereby constraining the overall definition of the function's boundaries.

Although the approach to creating bounded loss functions typically involves using truncated functions, these have been developed to effectively handle outliers through DC functions, which can be optimized using the DCA. However, these loss functions often exhibit abrupt behavior. Once the loss reaches a certain truncated threshold, it becomes constant rather than transitioning smoothly. Consequently, the gradient of the loss function becomes zero when the function flattens out, posing



(a) Rescaled hinge loss **(b)** Rescaled pinball loss

Figure 5 Illustration of hinge loss, pinball loss, rescaled hinge loss, and rescaled pinball loss.

challenges for first-order and second-order optimization techniques that rely on gradient information. To address this issue, the rescaling technique is employed. The rescaling technique [62,63] is particularly popular due to its simplicity, allowing for the creation of smoothly scaled bounded functions while preserving the key properties of the original loss function. This technique has been applied to hinge loss, resulting in the development of the rescaled hinge loss in SVM (RH-SVM) [64] to limit the original hinge loss. The first one is the rescaled hinge loss function [64], denoted as $L_{rh} : \mathbb{R} \to \mathbb{R}$, defined as:

$$\begin{aligned} L_{rh}(u) &= \eta \left(1 - \exp\left(-\frac{L_{hinge}(u)}{\lambda}\right)\right) \\ &= \begin{cases} 0, & u < 0, \\ \eta \left(1 - \exp\left(-\frac{u}{\lambda}\right)\right), & u \geq 0, \end{cases} \end{aligned}$$

where η is a constant that regulates the bound of the loss function, and $\lambda > 0$ is a scale parameter that modulates its magnitude (see Figure 5a).

Nevertheless, it still exhibits the same susceptibility to noise issues as the original hinge loss functions. To address this concern, the rescaled pinball loss has been introduced. The rescaled pinball loss is defined as follows [65]:

$$L_{rp}(u) = \eta \left(1 - \exp\left(-\frac{L_{\tau}(u)}{\lambda}\right)\right)$$

$$= \begin{cases} \eta\left(1-\exp\left(-\frac{\tau u}{\lambda}\right)\right), & u<0, \\ \eta\left(1-\exp\left(-\frac{u}{\lambda}\right)\right), & u\geq 0, \end{cases}$$

where $0 \leq \tau \leq 1$ and η is a constant and $\lambda > 0$ is a scale parameter (see Figure 5b).

Nevertheless, unconstrained minimization problems in SVM are not differentiable due to the non-differentiable nature of the loss functions. Consequently, several first-order and second-order optimization methods [57], which are effective and grounded in solid theoretical foundations, may not be directly applicable to such problems. To overcome the limitations of the hinge loss in SVM, the smooth hinge loss in SVM (SH-SVM) was introduced [58]. Similarly, the smooth pinball loss in SVM (SP-SVM) was proposed [59] to address the drawbacks of pinball loss. Building on these advancements, the smooth generalized pinball loss in SVM (SGP-SVM) was presented [60], offering a versatile framework that encompasses both SH-SVM and SP-SVM. The smooth generalized pinball loss function [60] has been proposed:

$$L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(u,\mu) = \begin{cases} \tau_1(u-\frac{\epsilon_1}{\tau_1}) - \frac{\tau_1^2}{2}\mu, & \frac{\epsilon_1}{\tau_1} + \tau_1\mu \leq u, \\ \frac{1}{2\mu}(u-\frac{\epsilon_1}{\tau_1})^2, & \frac{\epsilon_1}{\tau_1} < u < \frac{\epsilon_1}{\tau_1} + \tau_1\mu, \\ 0, & \frac{\epsilon_2}{\tau_2} \leq u \leq \frac{\epsilon_1}{\tau_1}, \\ \frac{1}{2\mu}(u+\frac{\epsilon_2}{\tau_2})^2, & \frac{-\epsilon_2}{\tau_2} - \tau_2\mu < u < \frac{-\epsilon_2}{\tau_2}, \\ -\tau_2(u+\frac{\epsilon_2}{\tau_2}) - \frac{\tau_2^2}{2}\mu, & u \leq \frac{-\epsilon_2}{\tau_2} - \tau_2\mu, \end{cases}$$

where $\tau_1, \tau_2, \epsilon_1, \epsilon_2 \geq 0$ and $\mu > 0$. The $L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(u,\mu)$ tent to $L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(u)$ as $\mu \to 0^+$, meaning it acts as an approximation of the generalized pinball loss (refer to Figure 6). Specifically, when $\tau_1 = 1$ and $\tau_2 = \epsilon_1 = \epsilon_2 = 0$, the SGP-SVM reduces to SH-SVM. Similarly, SGP-SVM becomes SP-SVM when $\tau_1 = 1$, $\tau_2 = \tau$, and $\epsilon_1 = \epsilon_2 = 0$. These smooth loss functions are continuous and differentiable, making the corresponding unconstrained minimization problems in SVM differentiable as well, thus allowing the direct application of first-order and second-order optimization methods.

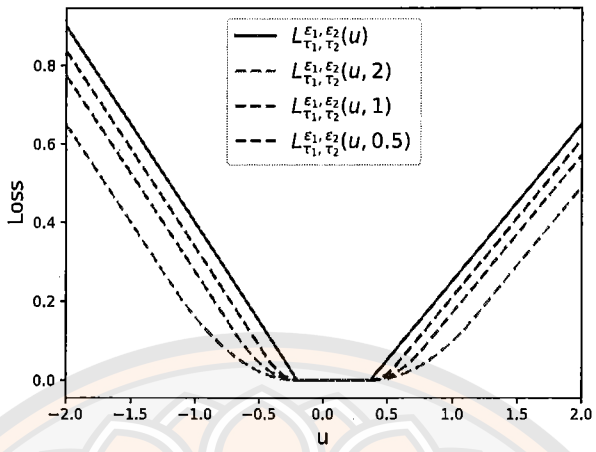


Figure 6 A comparison between generalized pinball loss and smooth generalized pinball loss.

2.4 DC algorithm

DC programming and DCA serve as the fundamental framework for non-convex continuous programming [34–36]. In essence, a DC program can be represented as follows:

$$\inf_{\mathbf{x}\in\mathbb{R}^n} \{g(\mathbf{x}) - h(\mathbf{x})\}, \quad (P_{dc})$$

where g and h are lower semicontinuous proper convex functions on $\mathbb{R}^n$. A function $\theta(\mathbf{x})$ is considered polyhedral convex provided that it satisfies the expression:

$$\theta(\mathbf{x}) = \max_{i\in\{1,2,\ldots,m\}} \{\mathbf{a}_i^T\mathbf{x} - b_i\} + \chi_\Omega(\mathbf{x}), \forall \mathbf{x} \in \mathbb{R}^n,$$

where $\mathbf{a}_i \in \mathbb{R}^n, b_i \in \mathbb{R}$, for $i = 1, 2, \ldots, m$.

The function $\chi_\Omega(\mathbf{x})$ denotes the indicator function of the non-empty convex set Ω and is defined as follows: $\chi_\Omega = 0$ if $\mathbf{x} \in \Omega$ and $+\infty$ otherwise. A DC program is termed a polyhedral DC program when either g or h is a polyhedral convex function. A point $\mathbf{x}^*$ satisfying the following generalized Kuhn-Tucker condition is termed a critical point of (P_{dc}):

$$\partial h(\mathbf{x}^*) \cap \partial g(\mathbf{x}^*) \neq \varnothing,$$

where ∂h denotes the subgradient of the convex function h. It can be inferred that if h is polyhedral convex, then such a critical point for (P_{dc}) typically constitutes a local solution. The necessary local optimality condition for (P_{dc}) is

$$\partial h\left(\mathbf{x}^*\right) \subset \partial g\left(\mathbf{x}^*\right) \neq \varnothing.$$

This condition is also sufficient for many significant classes of DC programs, including polyhedral DC programs. We utilize $g^*(\mathbf{y}) = \sup\left\{\mathbf{x}^T\mathbf{y} - g(\mathbf{x}), \mathbf{x} \in \mathbb{R}^n\right\}$ to denote the conjugate function of g. The Fenchel-Rockafellar dual of (P_{dc}) is defined as:

$$\inf_{\mathbf{y}\in\mathbb{R}^n}\left\{h^*(\mathbf{y}) - g^*(\mathbf{y})\right\}. \quad (D_{dc})$$

DCA is an iterative algorithm based on local optimality conditions and duality. The idea of DCA is simple: at each iteration, one replaces the second component h in the primal DC problem $(\boldsymbol{P}_{dc})$ by its affine minorization, $h(\mathbf{x}^k) + (\mathbf{x} - \mathbf{x}^k)^T\mathbf{y}^k$, to generate the convex program

$$\min_{\mathbf{x}\in\mathbb{R}^n}\left\{g(\mathbf{x}) - h(\mathbf{x}^k) - (\mathbf{x} - \mathbf{x}^k)^T\mathbf{y}^k,\ \mathbf{y}^k \in \partial h(\mathbf{x}^k)\right\}, \quad (2.4.1)$$

whose the solution set is $\partial g^*(\mathbf{y}^k)$.

Similarly, the second DC component g^* of the dual DC program (D_{dc}) is replaced by its affine minorization, $g^*(\mathbf{y}^k) + (\mathbf{y} - \mathbf{y}^k)^T\mathbf{x}^{k+1}$, to obtain a convex program whose the solution set is $\partial h(\mathbf{x}^k)$. In practice, a simplified version of the DCA is often utilized. This simplified scheme involves constructing two sequences, $\{\mathbf{x}^k\}$ and $\{\mathbf{y}^k\}$, where $\mathbf{y}^k \in \partial h(\mathbf{x}^k)$. Subsequently, $\mathbf{x}^{k+1}$ is obtained as a solution to the convex program (2.4.1). The simplified DCA scheme is outlined as follows:

Algorithm 1 DCA

Initialization: Choose an initial point $\mathbf{x}^0 \in \mathbb{R}^n$ and let $k = 0$.

Repeat

1: Calculate $\mathbf{y}^k \in \partial h(\mathbf{x}^k)$.

2: Solve convex program (2.4.1) to obtain $\mathbf{x}^{k+1}$.

3: Let $k = k + 1$.

Until some stopping criterion is satisfied.

2.5 First-order and second-order optimization methods

In this section, we address the following optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^n} F(\mathbf{w}) = \mathbb{E}[f(\mathbf{w},\theta)], \tag{2.5.1}$$

where $f : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}$ is a continuously differentiable and potentially nonconvex function, $\theta \in \mathbb{R}^d$ denotes a random variable with distribution function P, and $\mathbb{E}[\cdot]$ represents the expectation with respect to θ.

Frequently, the distribution function P may be unknown, making it impractical to directly optimize expectations over it. To overcome this challenge, we can approximate the expectation using a finite set of observed data points. Therefore, the training set is represented as a collection of samples $\Theta = [\theta_1, \ldots, \theta_N]$, where $\theta_i \in \mathbb{R}^d$ denotes the random variable in the ith sampling, and N denotes the number of data samples. It is assumed that Θ is independent of $\mathbf{w}$. Consequently, a special case of (2.5.1) that arises frequently in machine learning is the empirical risk minimization problem,

$$F(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w},\theta_i). \tag{2.5.2}$$

The gradient of (2.5.2) at $\mathbf{w}$ is calculated as follows:

$$\nabla F(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \nabla f(\mathbf{w},\theta_i).$$

Additionally, stochastic optimization methods are typically employed when dealing with large datasets. At each iteration k, the stochastic optimization method defines the function associated with the sample set $\xi_k = [\xi_{k,1}, \ldots, \xi_{k,m_k}]$, where $\xi_k \subset \Theta$ and $m_k < N$, as follows:

$$\hat{f}(\mathbf{w}_k,\xi_k) := \frac{1}{m_k}\sum_{i=1}^{m_k} f(\mathbf{w}_k,\xi_{k,i}),$$

and $\xi_{k,i}$ denotes the random variable generated by the ith sampling in the kth iteration. Subsequently, the stochastic gradient of $\hat{f}$ at $\mathbf{w}$, given the samples ξ_k, is defined by:

$$\mathbf{g}(\mathbf{w}_k,\xi_k) := \nabla\hat{f}(\mathbf{w}_k,\xi_k) = \frac{1}{m_k}\sum_{i=1}^{m_k} \nabla f(\mathbf{w}_k,\xi_{k,i}). \tag{2.5.3}$$

2.5.1 First-order methods

In first-order optimization methods, the gradient of the objective function with respect to the parameters is used to guide the optimization process. To lay the groundwork for understanding this class of methods, we begin by explaining the concept of gradient descent (GD). In GD, the initial point is denoted by $\mathbf{w}_0$, and the step size for each k^{th} iteration is given by α_t. Therefore, GD is used to update $\mathbf{w}$ according to the following:

$$\mathbf{z}_{k+1} = -\alpha_k \nabla F(\mathbf{w}_k),$$
$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{z}_{k+1}.$$

The transition from traditional gradient descent (GD) to stochastic gradient descent (SGD) marks a significant change in how model parameters are updated. Unlike GD, which relies on the average gradient computed from the entire dataset, SGD introduces an element of randomness into the optimization process. This is accomplished by selecting random subsets of data for each iteration, which improves the method's efficiency, particularly when working with large datasets.

This indicates that, on average, the stochastic gradient provides a reliable estimate of the gradient. Therefore, the stochastic gradient is used to update $\mathbf{w}$ according to the following:

$$\mathbf{z}_{k+1} = -\alpha_k \mathbf{g}(\mathbf{w}_k, \xi_k),$$
$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{z}_{k+1}.$$

Although SGD is effective, it can encounter difficulties with ill-conditioned problems [88].

2.5.2 Second-order methods

Recall that GD relies solely on first derivatives (gradients) to determine a suitable search direction. However, this strategy is not always the most effective. Utilizing higher derivatives can result in an iterative algorithm that performs better than GD. In this regard, Newton's method, also known as the Newton-Raphson

method, incorporates first and second derivatives, often demonstrating superior performance to GD, particularly when the initial point is close to the minimizer.

The essence of Newton's method is to locally approximate the function F being minimized, at every iteration, by a quadratic function. This approximation uses the minimizer of the quadratic function as the starting point for the next iteration. For a twice continuously differentiable objective function $F : \mathbb{R}^n \to \mathbb{R}$, we can use the Taylor series expansion of F around the current point $\mathbf{w}_t$, disregarding terms of order three and higher. This results in the following quadratic approximation:

$$F(\mathbf{w}) \approx F(\mathbf{w}_t) + \nabla F(\mathbf{w}_t)^\top (\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^\top \mathbf{H}_t (\mathbf{w} - \mathbf{w}_t). \tag{2.5.4}$$

Here, $\mathbf{w}$ denotes the step vector at the point $\mathbf{w}_t$, and $\mathbf{H}_t$ denotes the Hessian matrix of F at $\mathbf{w}_t$. Applying the first-order necessary condition to (2.5.4) yields:

$$0 = \nabla F(\mathbf{w}_t) + \mathbf{H}_t(\mathbf{w} - \mathbf{w}_t).$$

If $\mathbf{H}_t \succ 0$, then m_t achieves a minimum at

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1} \nabla F(\mathbf{w}_t). \tag{2.5.5}$$

The vector $-\mathbf{H}_t^{-1}\nabla F(\mathbf{w}_t)$ is called the Newton direction, and the algorithm induced by the update formula (2.5.5) is called Newton's method. If the objective function is quadratic, then this approximation is exact, and the method identifies the true minimizer in one step. However, a challenge with Newton's method is need to compute Hessian matrix in each iteration. For an n-dimensional problem, where we have an objective function $F : \mathbb{R}^n \to \mathbb{R}$, the computational complexity of calculating Hessian matrix is generally $O(n^2)$. This can be prohibitively expensive for large-scale problems. To address the previously mentioned issue, one approach is to develop variations of the quasi-Newton method. The conventional deterministic quasi-Newton method is characterized by

$$\mathbf{z}_{k+1} = -\alpha_k \mathbf{B}_k^{-1} \nabla F(\mathbf{w}_k),$$
$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{z}_{k+1},$$

where $\alpha_k > 0$ represents the step size, and $\mathbf{B}_k$ approximates the Hessian matrix $\nabla^2 F(\mathbf{w}_k)$. A widely used quasi-Newton method is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [89]. The BFGS method updates $\mathbf{B}_{k+1}$ as follows:

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^\top \mathbf{B}_k}{\mathbf{s}_k^\top \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k}, \tag{2.5.6}$$

where $\mathbf{s}_k = \mathbf{w}_{k+1} - \mathbf{w}_k$ and $\mathbf{y}_k = \nabla F(\mathbf{w}_{k+1}) - \nabla F(\mathbf{w}_k)$.

The primary adaptation of the BFGS method for stochastic settings involves substituting the exact gradient $\nabla F(\mathbf{w}_k)$ with the stochastic gradient as defined in (2.5.3). This substitution is the core concept of the stochastic BFGS method (oBFGS). However, in stochastic settings, $\mathbf{B}_k$ may encounter near-singularity issues, which can prevent the stochastic oBFGS method from ensuring a descent direction. To address this issue, a regularized stochastic BFGS method (RES) [92] has been proposed. To describe RES, we begin by modifying the matrix $\mathbf{B}_{k+1}$ by introducing a positive constant δ. RES provides an explicit form for $\mathbf{B}_{k+1}$, given by:

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^\top \mathbf{B}_k}{\mathbf{s}_k^\top \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k} + \delta \mathbf{I},$$

where $\mathbf{s}_k = \mathbf{w}_{k+1} - \mathbf{w}_k$, $\mathbf{y}_k = \mathbf{g}(\mathbf{w}_{k+1}, \xi_k) - \mathbf{g}(\mathbf{w}_k, \xi_k) - \delta \mathbf{s}_k$ and assume that $\mathbf{y}_k^\top \mathbf{s}_k > 0$. As a result, if $\mathbf{B}_k \succ 0$ for each iteration k, implying that $\mathbf{B}_{k+1} \succ \delta \mathbf{I}$. To mitigate the near-singularity issues of $\mathbf{B}_k$, it is necessary for the constant δ to be large. However, increasing δ leads to the largest eigenvalue of $\mathbf{B}_k^{-1}$ approaching zero, which means that updates must be made more slowly. To address this issue, a constant Γ is introduced, and $\Gamma \mathbf{I}$ is added to $\mathbf{B}_k^{-1}$, resulting in updated iterates $\mathbf{w}_k$ given by:

$$\begin{aligned} \mathbf{z}_{k+1} &= -\alpha_k (\mathbf{B}_k^{-1} + \Gamma \mathbf{I}) \mathbf{g}(\mathbf{w}_k, \xi_k), \\ \mathbf{w}_{k+1} &= \mathbf{w}_k + \mathbf{z}_{k+1}, \end{aligned}$$

where $\alpha_k > 0$ denotes the step size. On the other hand, since the oBFGS method has encountered considerable modifications to attain higher convergence. Consequently, the Nesterov-accelerated stochastic quasi-Newton approach (oNAQ) [93] was introduced to incorporate acceleration according to Nesterov's technique [97]. Unlike the oBFGS method, the oNAQ approach uses a quadratic approximation at the point $\mathbf{w}_k + \mu \mathbf{z}_k$, where μ represents the momentum coefficient. Therefore, the oNAQ method determines a new iteration at the k-th step by using

$$\begin{aligned} \mathbf{z}_{k+1} &= \mu \mathbf{z}_k - \alpha_k \mathbf{B}_k^{-1} \mathbf{g}(\mathbf{w}_k + \mu \mathbf{z}_k, \xi_k), \\ \mathbf{w}_{k+1} &= \mathbf{w}_k + \mathbf{z}_{k+1}, \end{aligned}$$

where $\mathbf{B}_k$ is updated by

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^\top \mathbf{B}_k}{\mathbf{s}_k^\top \mathbf{B}_k \mathbf{s}_k} + \frac{\mathbf{y}_k \mathbf{y}_k^\top}{\mathbf{y}_k^\top \mathbf{s}_k} \tag{2.5.7}$$

with $\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu\mathbf{z}_k)$, $\mathbf{y}_k = \mathbf{g}(\mathbf{w}_{k+1}, \xi_k) - \mathbf{g}(\mathbf{w}_k + \mu\mathbf{z}_k, \xi_k) + \lambda\mathbf{s}_k$, $\lambda\mathbf{s}_k$ is used to guarantee numerical stability, $\lambda > 0$ and $\mu \in (0, 1)$. Moreover, extensive experiments revealed that oNAQ effectively accelerated performance within the range of $\mu \in (0, 1)$ while considerably decreasing both computational and memory demands. Nonetheless, oNAQ presents notable limitations in certain scenarios. Specifically, the parameter μ, which is employed for acceleration, remains fixed and is recalculated for each iteration, potentially leading to overshooting in the solution.

As a result, the regularized stochastic Nesterov's accelerated quasi-Newton method (RES-NAQ) [94] was introduced, merging the principles of RES with those of oNAQ. This method incorporates an innovative momentum coefficient that adjusts dynamically between a constant and zero, based on the observed values of the objective function. According to the study discussed earlier, it was noted that adjusting μ when $F(\mathbf{w}_k + \mu\mathbf{z}_k) < F(\mathbf{w}_k)$ at each iteration k is beneficial. Consequently, the momentum coefficient is defined as follows:

$$\mu_k = \begin{cases} \mu, & \text{if } F(\mathbf{w}_k + \mu\mathbf{z}_k) < F(\mathbf{w}_k), \\ 0, & \text{otherwise}, \end{cases} \tag{2.5.8}$$

where $\mu \in (0, 1)$. Thus, for every iteration k, we have

$$F(\mathbf{w}_k + \mu_k\mathbf{z}_k) \leq F(\mathbf{w}_k). \tag{2.5.9}$$

Furthermore, regularized parameters are included to update the Hessian matrix $\mathbf{B}_k$ similarly to RES, to prevent near-singularity problems. Therefore, at iteration k with the initial $\mathbf{z}_0 = \mathbf{0}$, the proposed RES-NAQ method is updated by

$$\mathbf{z}_{k+1} = \mu_k\mathbf{z}_k - \alpha_k(\mathbf{B}_k^{-1} + \Gamma\mathbf{I})\mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k),$$
$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{z}_{k+1},$$

where $\alpha_k > 0$ is the step size and $\mathbf{B}_k$ is an approximate Hessian matrix updated by

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^\top\mathbf{B}_k}{\mathbf{s}_k^\top\mathbf{B}_k\mathbf{s}_k} + \frac{\mathbf{y}_k\mathbf{y}_k^\top}{\mathbf{y}_k^\top\mathbf{s}_k} + \delta\mathbf{I}, \tag{2.5.10}$$

where $\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k\mathbf{z}_k)$ and $\mathbf{y}_k = \mathbf{g}(\mathbf{w}_{k+1}, \xi_k) - \mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k) - \delta\mathbf{s}_k$. Evaluating the objective function with large datasets can be computationally expensive, and addressing nonconvex optimization problems presents significant challenges.

In nonconvex optimization problems, the positivity condition $\mathbf{s}_k^T\mathbf{y}_k > 0$ for the correction pairs might not always be satisfied, potentially resulting in a BFGS matrix that is not positive definite. To address this challenge, the damped RES method [96] was introduced to ensure the positive definiteness of the BFGS matrix in nonconvex optimization problems. To begin with, the following stochastic gradient difference $\hat{\mathbf{y}}_k$ is proposed to modify $\mathbf{y}_k$. Specifically, $\mathbf{y}_k$ is modified to $\hat{\mathbf{y}}_k = \phi_k\mathbf{y}_k + (1-\phi_k)(\mathbf{B}_k+\gamma\mathbf{I})\mathbf{s}_k - \delta\mathbf{s}_k$, where $\mathbf{y}_k = \mathbf{g}(\mathbf{w}_{k+1},\xi_k) - \mathbf{g}(\mathbf{w}_k,\xi_k)$ and ϕ_k is the damped parameter that satisfies

$$\phi_k = \begin{cases} \dfrac{0.8\mathbf{s}_k^\top(\mathbf{B}_k+\gamma\mathbf{I})\mathbf{s}_k - \delta\mathbf{s}_k^\top\mathbf{s}_k}{\mathbf{s}_k^\top(\mathbf{B}_k+\gamma\mathbf{I})\mathbf{s}_k - \mathbf{s}_k^\top\mathbf{y}_k}, & \text{if } \mathbf{s}_k^\top\mathbf{y}_k \le 0.2\mathbf{s}_k^\top(\mathbf{B}_k+\gamma\mathbf{I})\mathbf{s}_k + \delta\mathbf{s}_k^\top\mathbf{s}_k, \\ 1, & \text{otherwise,} \end{cases} \tag{2.5.11}$$

where γ and δ are positive constants such that $0.8\gamma < \delta$. Therefore, their proposed Hessian approximation updating scheme

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^\top\mathbf{B}_k}{\mathbf{s}_k^\top\mathbf{B}_k\mathbf{s}_k} + \frac{\hat{\mathbf{y}}_k\hat{\mathbf{y}}_k^\top}{\hat{\mathbf{y}}_k^\top\mathbf{s}_k} + \delta\mathbf{I}.$$

2.6 Evaluation Metrics

In evaluating model performance, several key metrics are commonly used for quantitative assessment:

- Accuracy (ACC) represents the rate of correctly identified observations from both classes. It is calculated as

$$\frac{TP+TN}{TP+FN+TN+FP}.$$

- Matthews correlation coefficient (MCC) is a comprehensive measure for classification models. It is computed as

$$\frac{TP\cdot TN - FP\cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}.$$

  The MCC value ranges from -1 to 1, with 1 representing a perfect prediction, 0 indicating a random prediction, and -1 signifying complete disagreement between predictions and actual observations.

- F1-score (F1) is another comprehensive metric for classification models, defined as

$$\frac{2 \times TP}{2 \times TP + FP + FN}.$$

The F1-score ranges from 0 to 1, with 1 representing perfect precision and recall, while 0 indicates the lowest performance.

In this context, TP and TN refer to true positives and true negatives, respectively, while FN and FP denote false negatives and false positives. Higher values of ACC, MCC, and F1-score indicate superior model performance.

2.7 Statistical Analysis

To justify the competitive performance of the experimental results, statistical analysis is employed. In this research, nonparametric statistics are considered. The most well-known procedure for testing the differences between more than two related samples is the Friedman test, accompanied by post hoc analysis [47]. The related samples in classification are the performances of the methods measured across the same data sets. It considers that the null hypothesis being tested is that all methods obtain similar results with nonsignificant differences. The Friedman statistic is computed as follows:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right],$$

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2},$$

where $R_j = \frac{1}{N} \sum_j r_j^i$ and r_j^i denotes the rank of the j^{th} algorithm on the i^{th} dataset out of N datasets. F_F follows an F-distribution with degrees of freedom $(k-1)(N-1)$, where k is the number of methods and N is the number of datasets. If the null hypothesis is rejected, a post hoc test can be conducted.

Moreover, the Nemenyi post hoc test is also employed and is computed using the critical difference (CD) value, given by the formula:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

where q_α is the critical value for a chosen significance level (e.g., 0.05), k is the number of groups being compared, and N is the total number of observations. In summary, the Nemenyi test provides critical values to determine the critical difference, which is used to assess the significance of pairwise differences between groups.

CHAPTER III

SUPPORT VECTOR MACHINE WITH BOUNDED GENERALIZED PINBALL LOSS

In this chapter, we will discuss the development of bounded loss functions in SVMs related to generalized pinball loss. The chapter is divided into two sections: the first introduces the asymmetric truncated generalized pinball loss, which provides a flexible two-sided bound for the loss and sparsity while minimizing the impact of outliers. The second presents the smooth rescaled generalized pinball loss function, which reduces the effect of outliers and is differentiable, enabling the direct application of both first-order and second-order optimization techniques.

3.1 Robust support vector machine with asymmetric truncated generalized pinball loss

Exploring the advantages of the asymmetric truncated ϵ-insensitive pinball loss function and the truncated generalized pinball loss function, we aim to craft a versatile, bounded loss function. Our proposed loss function, the asymmetric truncated generalized pinball loss function ($L_{tgp}^{\alpha_1,\alpha_2}$), effectively incorporates both loss functions. The $L_{tgp}^{\alpha_1,\alpha_2}$ is given by

$$L_{tgp}^{\alpha_1,\alpha_2}(u) = g_{tgp}(u) - h_{tgp}(u) \tag{3.1.1}$$

$$= \begin{cases} \alpha_1 - \epsilon_1, & u \geq \alpha_1/\tau_1, \\ \tau_1(u - \epsilon_1/\tau_1), & \epsilon_1/\tau_1 \leq u \leq \alpha_1/\tau_1, \\ 0, & -\epsilon_2/\tau_2 \leq u \leq \epsilon_1/\tau_1, \\ -\tau_2(u + \epsilon_2/\tau_2), & -\alpha_2/\tau_2 \leq u \leq -\epsilon_2/\tau_2, \\ \alpha_2 - \epsilon_2, & u \leq -\alpha_2/\tau_2, \end{cases}$$

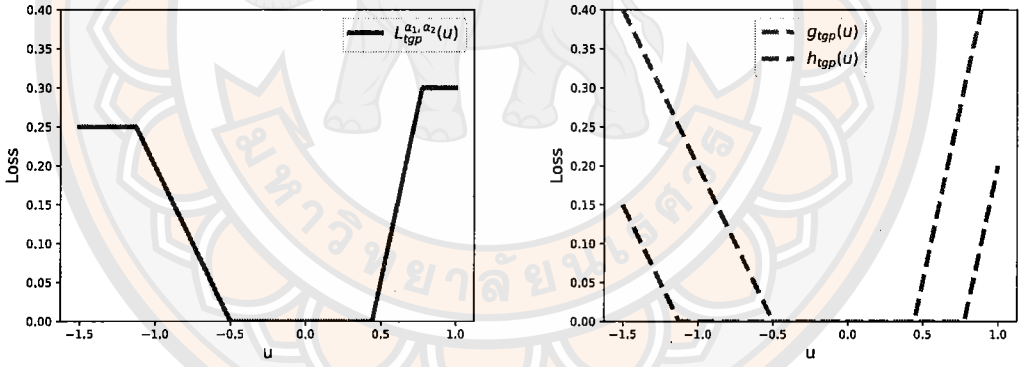
with

$$g_{tgp}(u) = \begin{cases} \tau_1(u - \epsilon_1/\tau_1), & u \geq \epsilon_1/\tau_1, \\ 0, & -\epsilon_2/\tau_2 \leq u \leq \epsilon_1/\tau_1, \\ -\tau_2(u + \epsilon_2/\tau_2), & u \leq -\epsilon_2/\tau_2, \end{cases} \tag{3.1.2}$$

and

$$h_{tgp}(u) = \begin{cases} \tau_1(u - \alpha_1/\tau_1), & u \geq \alpha_1/\tau_1, \\ 0, & -\alpha_2/\tau_2 \leq u \leq \alpha_1/\tau_1, \\ -\tau_2(u + \alpha_2/\tau_2), & u \leq -\alpha_2/\tau_2, \end{cases} \tag{3.1.3}$$

where parameters $0 \leq \tau_1 \leq 1$, $0 \leq \tau_2 \leq 1$ and $\alpha_1 > \epsilon_1 > 0, \alpha_2 > \epsilon_2 > 0$ (see Figure 15). From Figure 8, it can be seen that the $L_{tgp}^{\alpha_1,\alpha_2}$ is more flexible in selecting bounds than the L_{tgp}^{α}.



(a) (b)

Figure 7 Asymmetric truncated generalized pinball loss and its decomposition functions with $\tau_1 = 0.9$, $\tau_2 = 0.4$, $\alpha_1 = 0.7$, $\alpha_2 = 0.45$, $\epsilon_1 = 0.4$, and $\epsilon_2 = 0.2$.

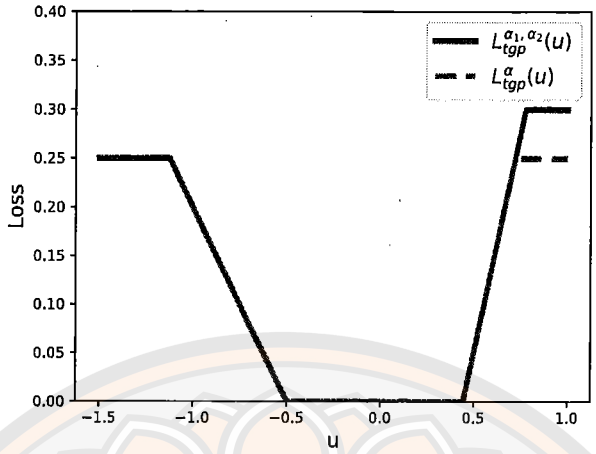


Figure 8 A comparison between truncated generalized pinball loss and asymmetric truncated generalized pinball loss with $\tau_1 = 0.9$, $\tau_2 = 0.4$, $\alpha = 0.25$, $\alpha_1 = 0.7$, $\alpha_2 = 0.45$, $\epsilon_1 = 0.4$ and $\epsilon_2 = 0.2$.

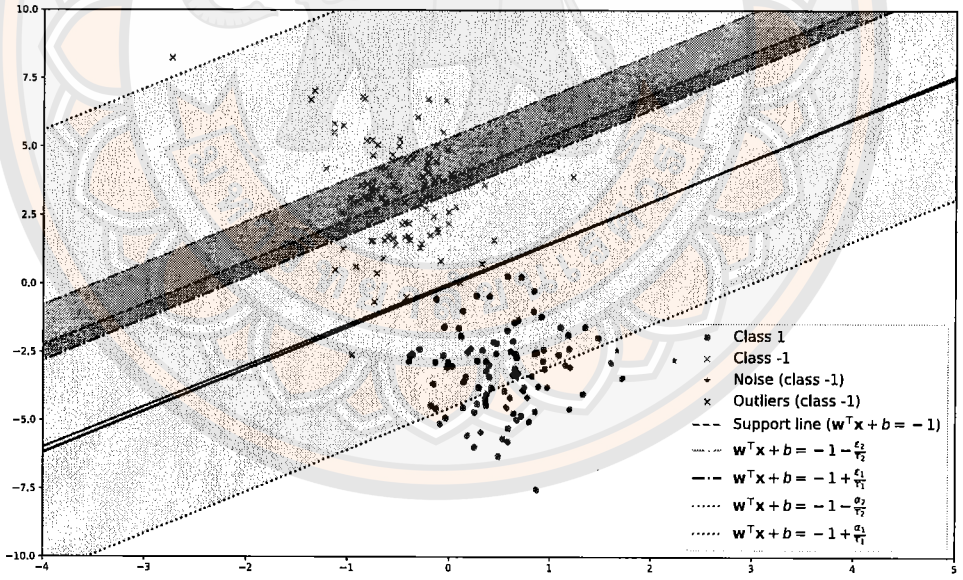


Figure 9 Effectiveness of the proposed $L_{tgp}^{\alpha_1,\alpha_2}$ loss function in handling noise and outliers while preserving sparsity.

Through the incorporation of the proposed $L_{tgp}^{\alpha_1,\alpha_2}$ into SVM, we introduce a novel robust SVM formulation that utilizes the $L_{tgp}^{\alpha_1,\alpha_2}$ loss. This formulation,

denoted as ATGP-SVM, is presented as follows:

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} L_{tgp}^{\alpha_1,\alpha_2}(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)). \tag{3.1.4}$$

Below are additional principal attributes of the $L_{tgp}^{\alpha_1,\alpha_2}(u)$ function:

- The proposed $L_{tgp}^{\alpha_1,\alpha_2}(u)$ is a non-negative, asymmetric, bounded and non-convex loss function that adheres to the inequality:

  $$0 \leq L_{tgp}^{\alpha_1,\alpha_2}(u) \leq \max\{\alpha_1 - \epsilon_1, \alpha_2 - \epsilon_2\},$$

  $$\lim_{u\to+\infty} L_{tgp}(u) = \alpha_1 - \epsilon_1, \lim_{u\to-\infty} L_{tgp}(u) = \alpha_2 - \epsilon_2,$$

  and

  $$L_{tgp}^{\alpha_1,\alpha_2}(u) \neq L_{tgp}^{\alpha_1,\alpha_2}(-u),$$

  when $\tau_1 \neq \tau_2$, $\epsilon_1 \neq \epsilon_2$ and $\alpha_1 \neq \alpha_2$. This indicates that $L_{tgp}^{\alpha_1,\alpha_2}(u)$ is less sensitive to outliers than traditional losses such as hinge loss, pinball loss, ϵ-insensitive pinball loss and generalized pinball loss. Additionally, the asymmetric feature enhances the model's versatility and applicability to various types of noise.

- In certain scenarios, it can be demonstrated that $L_{tgp}^{\alpha_1,\alpha_2} = L_{\epsilon tp}^{\alpha}$ when $\tau_1 = \tau, \tau_2 = (1-\tau), \epsilon_1 = \epsilon_2 = \tau(1-\tau)\epsilon, \alpha_1 = \tau\alpha+\tau(1-\tau)\epsilon, \alpha_2 = (1-\tau)\alpha+\tau(1-\tau)\epsilon$ and $L_{tgp}^{\alpha_1,\alpha_2} = L_{tgp}^{\alpha}$ when $\alpha_1 = \alpha+\epsilon_1, \alpha_2 = \alpha+\epsilon_2$. Moreover, $L_{tgp}^{\alpha_1,\alpha_2}$ encompasses and extends existing loss functions.

We will see that $L_{tgp}^{\alpha_1,\alpha_2}$ has a more flexible insensitive zone than $L_{\epsilon tp}^{\alpha}$ since the insensitive zone of $L_{\epsilon tp}^{\alpha}$ is defined by a single value of ϵ, while the insensitive zone of $L_{tgp}^{\alpha_1,\alpha_2}$ is defined by the values of ϵ_1, ϵ_2, which are more independent and therefore can handle the sparsity problem better. And $L_{tgp}^{\alpha_1,\alpha_2}$ has a more flexible bound than L_{tgp}^{α} since $L_{tgp}^{\alpha_1,\alpha_2}$ the boundaries of both sides of the function can be determined independently by α_1, α_2. This will be able to deal with outlier more effectively. Furthermore, $L_{tgp}^{\alpha_1,\alpha_2}$ can appropriately control the amount of loss using τ_1, while τ_2 can also adjust the loss and reduce the impact of noise, similar to $L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}$. In Figure 9, the gray tube represents the (ϵ_1, ϵ_2)-insensitive zone, where the loss

value is zero. This reduces the loss along the support line, which is not significant. It also decreases computation time, as many components are zero, illustrating the sparsity property of $L_{tgp}^{\alpha_1,\alpha_2}$. The purple and brown areas represent the loss function boundary. The black solid line represents the classifier obtained by ATGP-SVM, and the green solid line represents Bayes classifier. If data points are outside this boundary, the loss values for these points are positive constants $\alpha_1 - \epsilon_1$ and $\alpha_2 - \epsilon_2$, respectively. The flexibility in defining these bounds can help limit the influence of both noise and outliers. Therefore, Figure 9 clearly demonstrates the effectiveness of the proposed $L_{tgp}^{\alpha_1,\alpha_2}$ in addressing noise and outliers while preserving sparsity.

3.1.1 Sparsity

In this subsection, we show that ATGP-SVM preserves sparsity. For a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, the optimality condition for (3.1.4) can be written as

$$\mathbf{0} \in \frac{\mathbf{w}}{C} - \sum_{i=1}^{m} \partial L_{tgp}^{\alpha_1,\alpha_2}(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))y_i\mathbf{x}_i,$$

where $\mathbf{0}$ denotes the vector of which all the components equal zero, and the subgradient of $L_{tgp}^{\alpha_1,\alpha_2}(u)$-loss is as follows:

$$\partial L_{tgp}^{\alpha_1,\alpha_2}(u) = \begin{cases} 0 & u > \alpha_1/\tau_1 \\ [0, \tau_1], & u = \alpha_1/\tau_1, \\ \tau_1 & \epsilon_1/\tau_1 < u < \alpha_1/\tau_1, \\ [0, \tau_1] & u = \epsilon_1/\tau_1, \\ 0 & -\epsilon_2/\tau_2 < u < \epsilon_1/\tau_1, \\ [-\tau_2, 0] & u = -\epsilon_2/\tau_2, \\ -\tau_2 & -\alpha_2/\tau_2 < u < -\epsilon_2/\tau_2, \\ [-\tau_2, 0] & u = -\alpha_2/\tau_2, \\ 0 & u < -\alpha_2/\tau_2. \end{cases}$$

For given the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, it does not contribute to the classification hyperplane when $1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) > \alpha_1/\tau_1, -\epsilon_2/\tau_2 < 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) < \epsilon_1/\tau_1$ and $1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) < -\alpha_2/\tau_2$. Therefore, the ATGP-SVM demonstrates sparsity to some degree.

3.1.2 DC algorithm for solving ATGP-SVM

By introducing the ATGP-SVM, it adeptly handles outliers and offers boundary flexibility. It's worth noting that defining the function boundary turns convex loss functions into non-convex forms, posing an optimization challenge for ATGP-SVM. However, $L_{tgp}^{\alpha_1,\alpha_2}$-loss lies in representing the difference of two convex functions, allowing us to leverage the DC algorithm (DCA), known for its effectiveness in non-convex optimization, particularly in large-scale settings. Therefore, we will utilize DCA as a suitable method for optimizing ATGP-SVM . By employing DC decomposition in the $L_{tgp}^{\alpha_1,\alpha_2}$-loss, ATGP-SVM can be formulated as a DC programming problem:

$$\begin{aligned}\min_{\mathbf{w},b} &\underbrace{\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} g_{tgp}(1-y_i(\mathbf{w}^T\mathbf{x}_i+b))}_{\text{convex}} \\ &\underbrace{- C\sum_{i=1}^{m} h_{tgp}(1-y_i(\mathbf{w}^T\mathbf{x}_i+b))}_{\text{convex}}\end{aligned} \tag{3.1.5}$$

with DC decomposition

$$G(\mathbf{w},b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} g_{tgp}(1-y_i(\mathbf{w}^T\mathbf{x}_i+b)) \tag{3.1.6}$$

and

$$H(\mathbf{w},b) = C\sum_{i=1}^{m} h_{tgp}(1-y_i(\mathbf{w}^T\mathbf{x}_i+b)). \tag{3.1.7}$$

Evidently, both $G(\mathbf{w},b)$ and $H(\mathbf{w},b)$ exhibit convexity, with $H(\mathbf{w},b)$ being a polyhedral convex function. Therefore, problem (3.1.5) qualifies as a polyhedral DC programming problem. As outlined in Section 2, the implementation of DCA for problem (3.1.5) involves computing the sequence $\{\mathbf{x}^k, k=1,2,\ldots\}$, where $\mathbf{x}^{k+1}$ is obtained by solving a series of quadratic programming problems:

$$\begin{aligned}(\mathbf{w}_{k+1},b_{k+1})^T = \operatorname*{argmin}_{\mathbf{w},b} &\frac{1}{2}\|\mathbf{w}\|^2 \\ &+ C\sum_{i=1}^{m} g_{tgp}(1-y_i(\mathbf{w}^T\mathbf{x}_i+b)) \\ &- C\sum_{i=1}^{m}\left[h_{tgp}(1-y_i(\mathbf{w}_k^T\mathbf{x}_i+b_k))\right]\end{aligned}$$

$$- C\gamma_k^T \left((\mathbf{w}, b)^T - (\mathbf{w}_k, b_k)^T \right),$$

$$\gamma_k \in \partial \sum_{i=1}^{m} h_{tgp}(1 - y_i(\mathbf{w}_k^T \mathbf{x}_i + b_k))$$

it can be simplified as,

$$\begin{aligned}
&\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i - C\gamma_k^T (\mathbf{w}, b)^T, \\
&\text{s.t. } \tau_1(1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \leq (\xi_i + \epsilon_1), i = 1, 2, \cdots, m, \\
&-\tau_2(1 - y_i(\mathbf{w}^T \mathbf{x}_i + b)) \leq (\xi_i + \epsilon_2), i = 1, 2, \cdots, m, \\
&\qquad \xi \geq 0, i = 1, 2, \cdots, m,
\end{aligned} \tag{3.1.8}$$

where

$$\gamma_k \in \partial \sum_{i=1}^{m} h_{tgp}(1 - y_i(\mathbf{w}_k^T \mathbf{x}_i + b_k))$$

and the subgradient of h_{tgp} is given by

$$\begin{aligned}
&\partial h_{tgp}(u_{i,k}) \\
&= \begin{cases}
-\tau_1 y_i(\mathbf{x}_i^\top, 1), & u_{i,k} > \alpha_1/\tau_1, \\
\left[0, -\tau_1 y_i(\mathbf{x}_i^\top, 1)\right], & u_{i,k} = \alpha_1/\tau_1, \\
0 & -\alpha_2/\tau_2 < u_{i,k} < \alpha_1/\tau_1, \\
\left[0, \tau_2 y_i(\mathbf{x}_i^\top, 1)\right] & u_{i,k} = -\alpha_2/\tau_2, \\
\tau_2 y_i(\mathbf{x}_i^\top, 1) & u_{i,k} < -\alpha_2/\tau_2,
\end{cases}
\end{aligned} \tag{3.1.9}$$

where $u_{i,k} = 1 - y_i(\mathbf{w}_k^T \mathbf{x}_i + b_k), i = 1, 2, \cdots, m$. Based on the preceding analysis, the solution to ATGP-SVM can be outlined as follows:

Algorithm 2 DCA for solving ATGP-SVM

Input: Training set $T = \{\mathbf{x}_i, y_i\}_{i=1}^{m}, \mathbf{x}_i \in \mathbb{R}^n$. Let $\varepsilon > 0$ be a small real number, and I be the maximum number of iterations. Merge variables w and b into β. Initialize $\beta_0 = (\mathbf{w}_0, b_0)^T = (\beta_1, \beta_2, \ldots, \beta_{n+1})^T$ and set $k = 1$.

1: **Step 1**: Using β_{k-1}, calculate the subgradient of h_{tgp} from (3.1.9) and determine γ_k.

2: **Step 2**: Using γ_k, calculate β_k by solving the quadratic programming problem from (3.1.8).

3: **Step 3**: If $|\beta_k - \beta_{k-1}| < \varepsilon$ or $k > I$, then stop and $\mathbf{x}^{k+1}$ is the computed solution. Otherwise, set $k = k + 1$ and return to Step 1.

Theorem 3.1.1. *Algorithm 2 generates the sequence* $\{\boldsymbol{x}^k\}$ *such that* $\{G(\boldsymbol{x}^k) - H(\boldsymbol{x}^k)\}$ *decreases monotonically, and after a finite number of iterations, the sequence* $\{\boldsymbol{x}^k\}$ *converges to* $\boldsymbol{x}^*$*, which is a critical point of the DC programming (3.1.5).*

Proof. The initial statement directly follows from the convergence properties of the general DC program. The feasible region of DC programming (3.1.5) can be defined as a convex polyhedral set. Furthermore, the function $H(\mathbf{w}, b)$ demonstrates polyhedral convexity. Thus, DC programming (3.1.5) encompasses a DC polyhedral program, ensuring that the sequences $\{\mathbf{x}^k\}$ generated by Algorithm 2 converge to a point $\mathbf{x}^*$ after a finite number of iterations. Considering the convergence properties of DC polyhedral programming, we infer that the limit point $\mathbf{x}^*$ serves as a critical point of DC programming (3.1.5). □

3.1.3 Complexity

In this section, we analyze the complexity of ATGP-SVM. Let n represent the dimension of the feature for each data. To obtain the classifier, the inverse of an $(n+1) \times (n+1)$ matrix is computed during quadratic programming, resulting in a complexity of $\mathcal{O}((n+1)^3)$ per iteration. Considering the maximum number of iterations I, the overall complexity is $\mathcal{O}(I \cdot (n+1)^3)$.

3.1.4 Numerical experiments

Numerical experiments were conducted on various datasets to evaluate the effectiveness of the proposed method. The experiments comprised two parts. Initially, experiments were performed on synthetic datasets, followed by the application of the proposed method to benchmark UCI datasets, compared with baseline methods. The experiments were executed on a MacBook Air equipped with an Apple M1 chip, featuring an 8-core CPU (3.2GHz) and 8 GB of memory. Five-fold cross-validation was employed for each dataset under consideration. In evaluating the comparison of the proposed method with baseline methods, this study utilizes ACC, MCC, and F1-score.

Additionally, we provide the Python code used in this study. Before running the code, it is necessary to set up the required modules. Note that all the code presented here is derived from the BaseEstimator class, which includes self in function inputs. If you plan to use the code, we recommend reviewing it carefully. The Python code in this task was implemented using the following environment: pandas 1.3.2, numpy 1.20.3, matplotlib 3.5.3, scikit-learn 0.24.2, scipy 1.6.2, and cvxopt 1.3.0. The following Python code defines the class of loss functions in ATGP-SVM:

```
class ATGPIN:
    def __init__(self,tau1=0.1, tau2 =0.1, alpha1=2, alpha2=3,
    ↪ epsilon1 = 0.1, epsilon2= 0.1):
        self.tau1 = tau1
        self.tau2 = tau2
        self.alpha1 = alpha1
        self.alpha2 = alpha2
        self.epsilon1 = epsilon1
        self.epsilon2 = epsilon2
    def g(self,u, tau1, tau2, epsilon1, epsilon2):
        if epsilon1 / tau1 <= u:
            return tau1 * ( u -(epsilon1/tau1))
        elif -epsilon2 / tau2 <= u <= epsilon1 / tau1:
            return 0
        elif u <=  -epsilon2 / tau2:
            return -tau2 * ( u +(epsilon2/tau2))

    def h(self,u, tau1, tau2, alpha1, alpha2):
```

```
18          if alpha1 / tau1 <= u:
19              return tau1 * ( u -(alpha1/tau1))
20          elif -alpha2 / tau2 <= u <= alpha1/ tau1:
21              return 0
22          elif u <=  -alpha2 / tau2:
23              return -tau2* ( u +(alpha2/tau2))
24
25      def sum_g(self,w, b, X, y, tau1, tau2, epsilon1, epsilon2):
26          total_loss_g = 0
27          N = X.shape[0]
28          for i in range(N):
29              u = 1 - ((y[i] * (np.dot(X[i], w))+b))
30              g_value = ATGPIN.g(self,u, tau1, tau2, epsilon1,
                    ↪ epsilon2)
31              total_loss_g += g_value
32          return total_loss_g
33
34      def sum_h(self,w,b,X, y, tau1, tau2, alpha1, alpha2):
35          total_loss_h = 0
36          N = X.shape[0]
37          for i in range(N):
38              u = 1 - (y[i] * (np.dot(X[i], w)+b))
39              h_value = ATGPIN.h(self,u, tau1, tau2, alpha1, alpha2)
40              total_loss_h += h_value
41          return total_loss_h
42
43      def subgradient_h(self,u, X, y, tau1, tau2, alpha1, alpha2):
44          value_to_add = 1
45          X1 = np.append(X, value_to_add)
46          p = X1.shape[0]
47          if u > alpha1 / tau1:
48              return -tau1*(y * X1)
49          elif -(alpha2 / tau2) <= u and u <= alpha1/tau1:
50              return  np.zeros(p)
51          else:
52              return tau2*(y * X1)
53
54      def sum_subgradient_h(self,w, b, X, y, tau1, tau2, alpha1,
            ↪ alpha2):
55          total_sgh = np.zeros(len(w)+1)
```

```
        N = X.shape[0]
        for i in range(N):
            u = 1 - (y[i] * ((X[i] @ w )+b))
            subgradient_h_value =
                ↪ ATGPIN.subgradient_h(self,u,X[i],y[i], tau1,
                ↪ tau2, alpha1, alpha2)
            total_sgh += subgradient_h_value
        return total_sgh
```

The following Python code implements the DC algorithm for solving ATGP-SVM in this experiment:

```
class ATGP_SVM(BaseEstimator):
    def __init__(self, tau1=0.1,  tau2 =0.1, epsilon1 = 0.1,
    ↪ epsilon2 = 0.1, alpha1=2, alpha2=2,  C=1,
    ↪ num_iterations=50):
        self.C = C
        self.tau1 = tau1
        self.tau2 = tau2
        self.epsilon1 = epsilon1
        self.epsilon2 = epsilon2
        self.alpha1 = alpha1
        self.alpha2 = alpha2
        self.num_iterations = num_iterations
        self.w = None
        self.b = None

    def fit(self, X, y):
        m, n = X.shape
        #theta =np.ones(n)
        w = np.zeros(n)
        b = np.zeros(1)
        for _ in range(self.num_iterations):
            sum_subgradient_h_ATGPIN =   Loss_funtions.ATGPIN()
            gamma = sum_subgradient_h_ATGPIN.sum_subgradient_h(w,
                ↪ b, X, y, self.tau1, self.tau2, self.alpha1,
                ↪ self.alpha2)
            w = cp.Variable(n)
            b = cp.Variable(1)
            xi = cp.Variable(m)
```

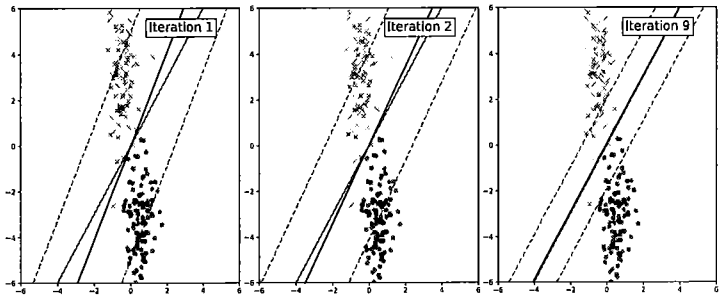
```
            objective = cp.Minimize((0.5 * cp.norm(w)**2) + self.C
                ↪ *( cp.sum(xi)) - self.C *( cp.matmul(gamma[:-1],
                ↪ w)) - self.C * (gamma[-1] * b))
            constraints = [
                xi >= (self.tau1 * (np.ones(m) - (np.diag(y) @ ((
                    ↪ (X @ w ) + b))))) - self.epsilon1,
                xi >= (-self.tau2 * (np.ones(m) - (np.diag(y) @
                    ↪ ((X @ w + b))))) -self.epsilon2,
                xi >= np.zeros(m)
            ]
            problem = cp.Problem(objective, constraints)
            problem.solve(solver=cp.ECOS, verbose=False)
            w = w.value
            b = b.value
        self.w = w
        self.b = b
        return self.w, self.b

    def predict(self, X):
        # Compute the decision function
        decision = np.dot(X, self.w) + self.b
        predictions = np.sign(decision)
        predictions = np.where(predictions == 0, -1, predictions)
        return predictions

    def score(self, X, y):
        y_pred = np.dot(X, self.w) + self.b
        y_pred = np.sign(y_pred)
        y_pred = np.where(y_pred == 0, -1, y_pred)
        accuracy = (y_pred == y).mean()
        return accuracy
```

3.1.5 Experiments on synthetic datasets

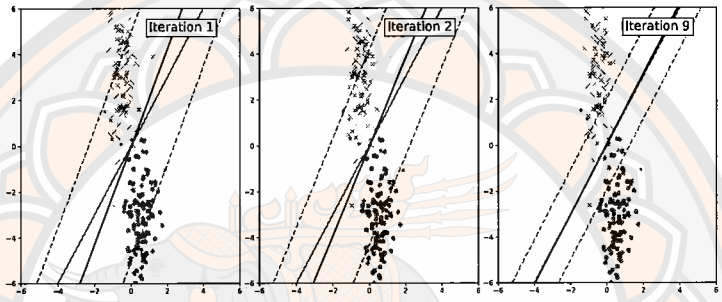
We test our approach using a two-dimensional scenario where equal samples are drawn from two Gaussian distributions $x_i, i \in \{i : y_i = 1\} \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $x_i, i \in \{i : y_i = -1\} \sim \mathcal{N}(\mu_2, \Sigma_2)$, where $\mu_1 = [0.5, -3]^\top, \mu_2 = [-0.5, 3]^\top$

and $\Sigma_1 = \Sigma_2 = [0.2\ 3; 0\ 3]$. We generated training data, each set consisting of 100 samples. In this experiment, the Bayes classifier is defined by the equation $f(x) = 2.5x(1) - x(2)$, where $x = [x(1), x(2)]^\top \in \mathbb{R}^2$. The Bayes classifier is illustrated in Figure 10. To add complexity to the dataset, we introduced noise to the training data. The labels were chosen from $\{-1, 1\}$ with equal probability and the positions follow Gaussian distribution $N(\mu_n, \Sigma_n)$ with $\mu_n = [0, 0]^T$ and $\Sigma_n = [1\ -0.8; -0.8\ 1]$. We consider the ratio of noise data r, which can take values of 0.00, 0.10, 0.15, and 0.20, to control the level of noise relative to the total number of samples. A value of 0.00 indicates no noise data. The labels around the decision boundaries are affected by the noise. The Bayes classifier remains unchanged. Figure 10 illustrates the outcomes achieved by Bayes and ATGP-SVM following each iteration. The red dotted line represents the support line, the black solid line represents the classifier obtained by ATGP-SVM, and the green solid line represents the Bayes classifier As depicted in Figure 10, a discernible trend of diminishing noise influence emerges after each iteration. It is evident that ATGP-SVM demonstrates insensitivity to noise along the decision boundary. Notably, ATGP-SVM exhibits rapid convergence, as the classifier obtained after the tenth iteration closely resembles the Bayes classifier. These findings suggest that ATGP-SVM is highly robust to noise, achieving convergence within a few iterations.

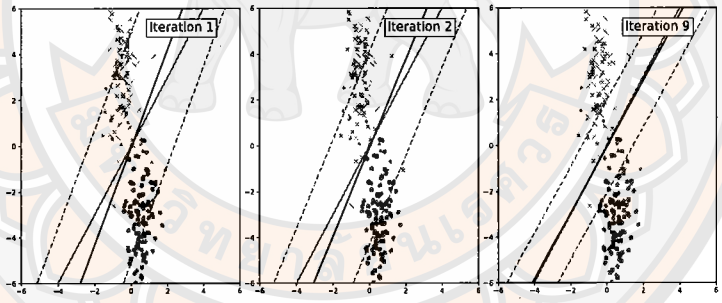
Moreover, we also consider $\mu_n = [0, -20]^T$, which creates an outlier to compare performance between ATGP-SVM and TGP-SVM in Table 1. We generate a training set for each class of 200 (called DATA-I) and 400 (called DATA-II). We consider the ratio of noise data r from 0.05 to 0.20. We fix the parameters as follows: τ_1 is 0.2 and τ_2 is 0.1, ϵ_1 is 0.01, ϵ_2 is 0.015 for ATGP-SVM and TGP-SVM, and α is 0.5 for TGP-SVM and α_1 is 0.5, and α_2 is 0.02 for ATGP-SVM. From Table 1, we can see that at a noise level of 0.05, TGP-SVM has better performance than ATGP-SVM, but as the noise level increases, ATGP-SVM can perform better than TGP-SVM in both cases. Additionally, we consider α_1 and α_2 from the set $\{0.02, 0.1, 0.2, 0.5\}$, which demonstrates the impact of α_1 and α_2 on the accuracy of ATGP-SVM for the synthetic dataset with outliers, as depicted in Figures 11 and 12. These figures illustrate that as the level of outliers increases, smaller values of α_1 and α_2 should be used to reduce the effect of outliers, thereby improving the model's performance.



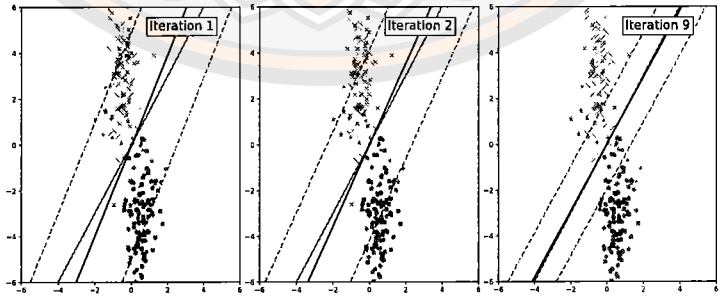
(a) $r = 0.00$



(b) $r = 0.10$

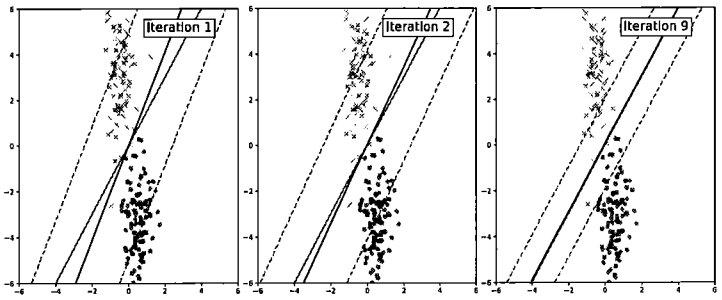


(c) $r = 0.15$

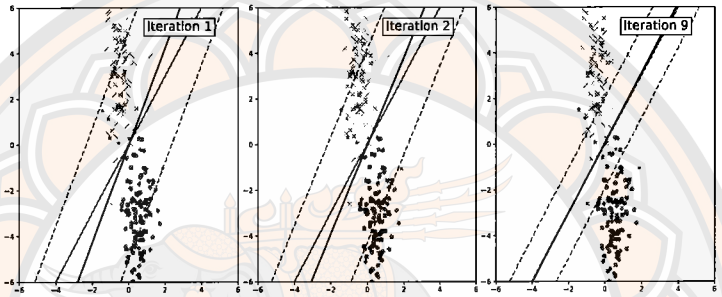


(d) $r = 0.20$

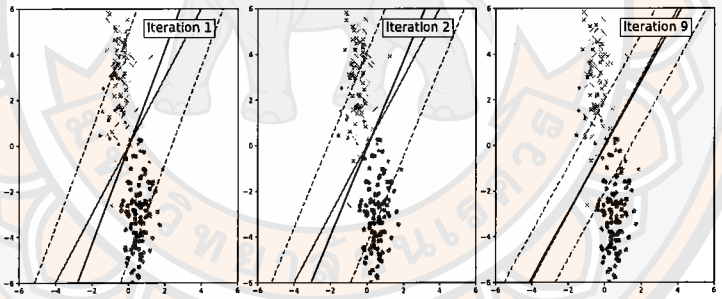
Figure 10: Comparison of results obtained by Bayes classifier and ATGP-SVM on synthetic data.



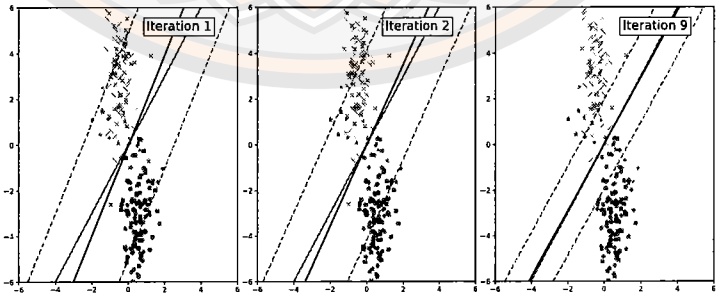
(a) $r = 0.00$



(b) $r = 0.10$

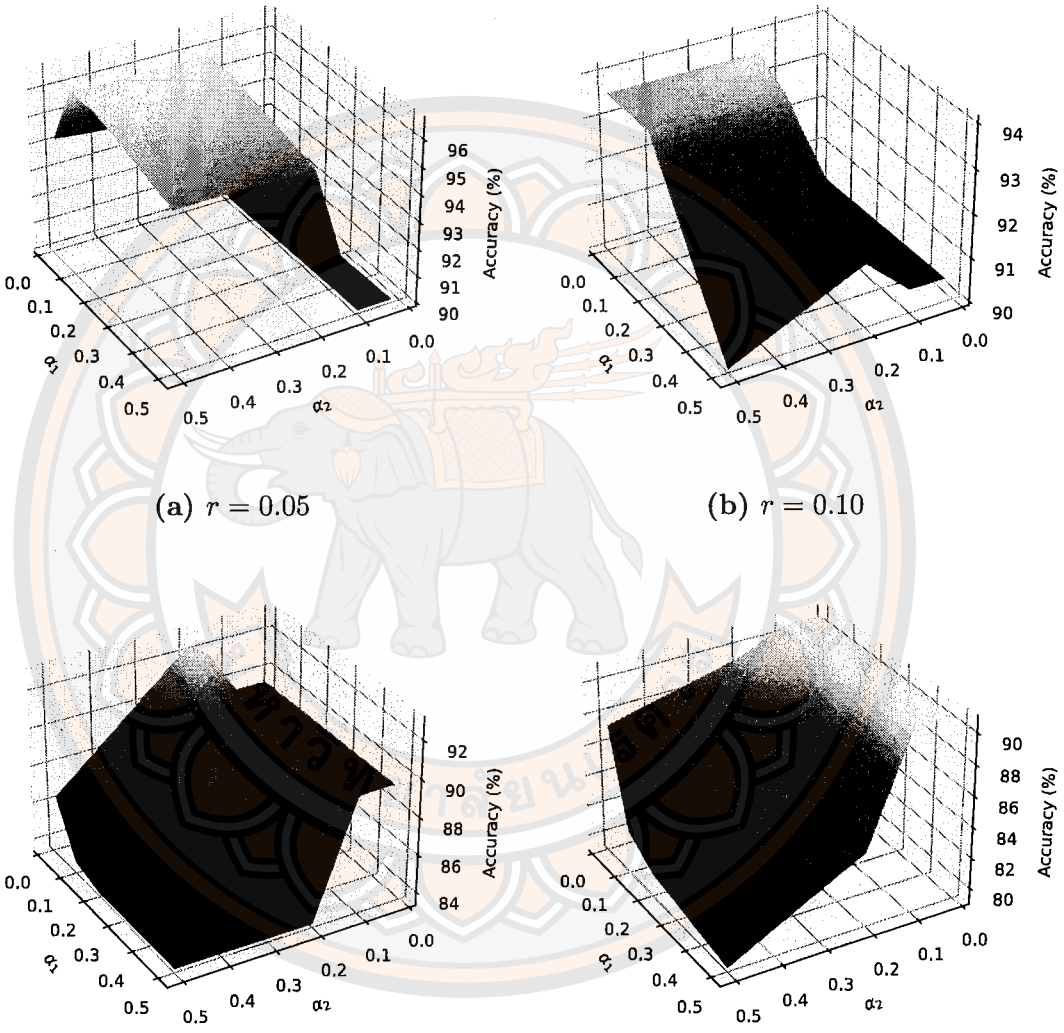


(c) $r = 0.15$



(d) $r = 0.20$

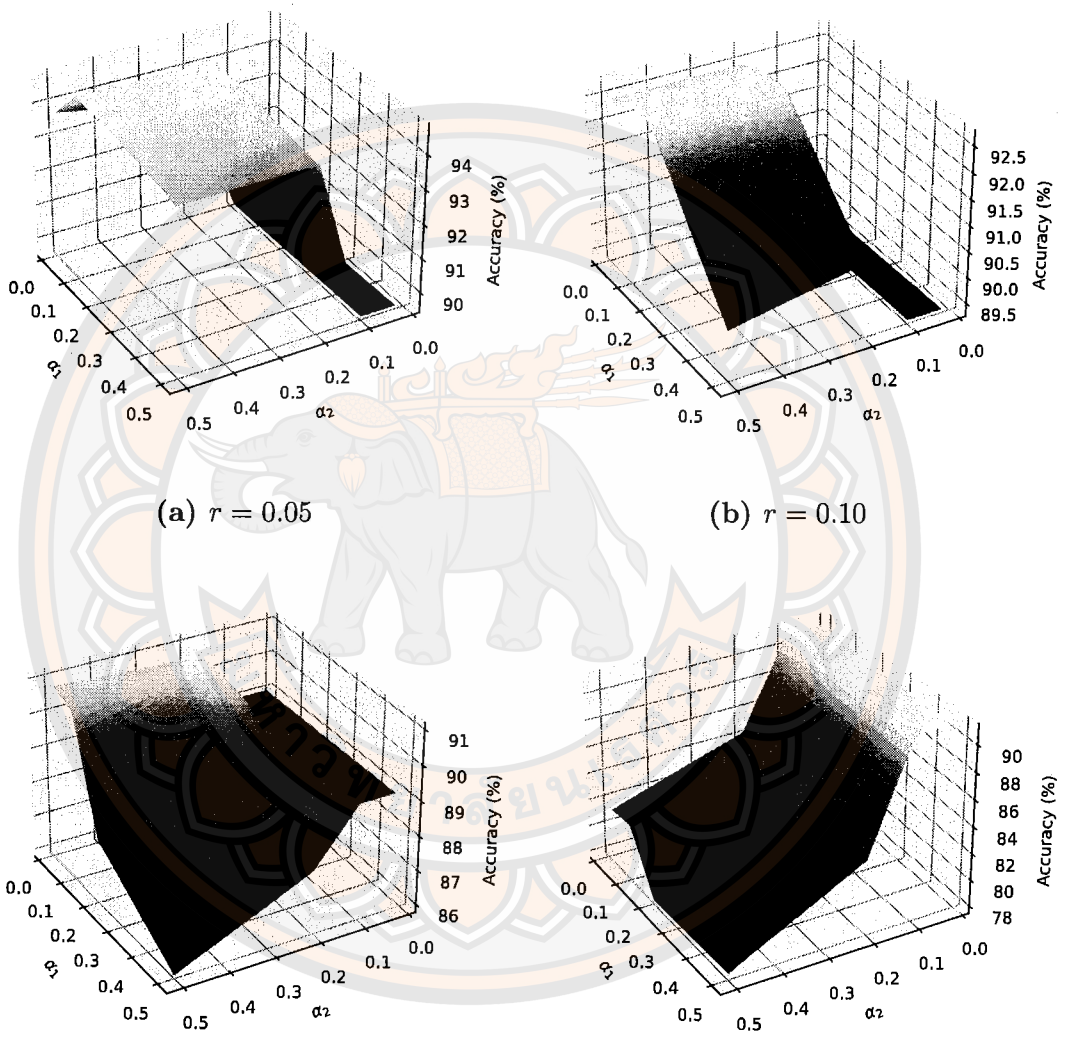
Figure 10 Comparison of results obtained by Bayes classifier and ATGP-SVM on synthetic data.



(a) $r = 0.05$ (b) $r = 0.10$

(c) $r = 0.15$ (d) $r = 0.20$

Figure 11 The variation in test accuracy attributed to α_1 and α_2 over DATA-I.



(a) $r = 0.05$ **(b)** $r = 0.10$

(c) $r = 0.15$ **(d)** $r = 0.20$

Figure 12 The variation in test accuracy attributed to α_1 and α_2 over DATA-II.

Table 1 Comparison of ATGP-SVM and TGP-SVM on synthetic datasets with outliers.

Datasets		Criteria	ACC (%)	MCC (%)	F_1 (%)
DATA-I	r = 0.05	ATGP-SVM	95.24 ± 2.13	90.65 ± 4.01	95.07 ± 2.59
		TGP-SVM	**95.71± 2.78**	**91.59 ± 5.33**	**95.50 ± 3.13**
	r = 0.10	ATGP-SVM	**93.18 ± 1.44**	**86.97 ± 2.88**	**93.47 ± 1.54**
		TGP-SVM	92.73 ± 1.70	86.17 ± 3.28	93.09 ± 1.62
	r = 0.15	ATGP-SVM	**90.87 ± 1.63**	**82.37 ± 2.92**	91.50 ± 1.11
		TGP-SVM	86.52 ± 5.74	73.18 ± 12.04	87.16 ± 4.44
	r = 0.20	ATGP-SVM	**88.75 ± 1.67**	**79.33 ± 2.78**	**89.64 ± 1.50**
		TGP-SVM	85.42 ± 4.37	71.74 ± 8.74	86.22 ± 4.29
DATA-II	r = 0.05	ATGP-SVM	95.00 ± 1.58	90.12 ± 3.15	95.07 ± 1.70
		TGP-SVM	**95.24± 1.68**	**90.60 ± 3.34**	**95.38 ± 1.71**
	r = 0.10	ATGP-SVM	**92.95 ± 3.17**	**86.00 ± 6.21**	**93.10 ± 3.44**
		TGP-SVM	92.50 ± 3.71	85.14 ± 7.26	92.68 ± 3.98
	r = 0.15	ATGP-SVM	**90.86 ± 2.34**	**82.01 ± 4.92**	**91.48 ± 2.09**
		TGP-SVM	89.35 ± 2.52	78.90 ± 5.43	89.92 ± 2.67
	r = 0.20	ATGP-SVM	**86.04 ± 3.27**	**73.22 ± 5.41**	**86.51 ± 3.40**
		TGP-SVM	85.00 ± 3.70	71.13 ± 6.48	85.44 ± 3.99

3.1.6 Experiments on UCI benchmark datasets

We conducted experiments on UCI benchmark datasets (Ionosphere [37], Pima [38], Diabetes [39], Bupa [40], Sonar [41], Heart Failure Clinical Records [42], Breast [43], Rice [44], WDBC [71], and NHANES [46]) under both noisy and noiseless settings. In the noisy setting, the training datasets are affected by zero-mean Gaussian noise with variances of 0.1 and 1.0. Additionally, we considered kernel methods. Rather than having direct access to the feature vector data, support vector machines can utilize an approximate feature map. We evaluated both a linear kernel and a nonlinear kernel (e.g., RBF kernel). To assess the effectiveness of the proposed ATGP-SVM on the UCI datasets, we selected GP-SVM, TPin-SVM, and CSVM as baseline methods. Furthermore, we measured the average computation time for each method.

In this investigation, we conducted parameter optimization aiming to maximize accuracy through five-fold cross-validation on each dataset. Upon parameter selection, the optimal values were employed to scrutinize the experimental

results. The precise optimal parameter values are outlined in the corresponding experimental sections. We established the iteration parameters as $\epsilon = 10^{-4}$ and $I = 10$. In all methods, parameter C is fixed at 1. In order to facilitate the calculation, with $\alpha_2 = \alpha_1/5$ we search α_1 from set $\{0.2, 0.5, 1, 2, 3\}$ and τ, τ_1, τ_2 from set $\{0.1, 0.2, 0.3, 0.9\}$ and ϵ_1 from set $\{0.01, 0.025, 0.05, 0.075, 0.1\}$ and ϵ_2 from set $\{0.002, 0.005, 0.01, 0.015, 0.02\}$. The optimal parameters are determined through five-fold cross-validation to maximize accuracy (ACC). The optimal parameters for the linear kernel in both noiseless and noisy cases are presented in Tables 2, 3, and 4. The optimal parameters for the RBF kernel in noiseless and noisy cases are shown in Tables 5, 6, and 7.

Table 2 The optimal parameters of ATGP-SVM, TPin-SVM and GP-SVM on UCI data without noise using a linear kernel.

Datasets	GP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$	TPin-SVM τ, α_1, α_2	ATGP-SVM $\tau_1, \tau_2, \epsilon_1$ $\epsilon_2, \alpha_1, \alpha_2$
Ionosphere	0.1, 0.1, 0.075, 0.02	0.1, 3, 0.6	0.1, 0.1, 0.075, 0.02 0.2, 0.04
Pima	0.1, 0.3, 0.05, 0.015	0.3, 2, 0.4	0.1, 0.3, 0.05, 0.015 2, 0.4
Diabetes	0.2, 0.1, 0.01, 0.002	0.2, 3, 0.6	0.2, 0.1, 0.01, 0.002 2, 0.4
Bupa	0.1, 0.3, 0.1, 0.015	0.3, 2, 0.4	0.1, 0.3, 0.1, 0.015 2, 0.4
Sonar	0.1, 0.2, 0.1, 0.01	0.1, 2, 0.4	0.1, 0.2, 0.1, 0.01 1, 0.2
Heart Failure Clinical Records	0.1, 0.9, 0.1, 0.075	0.1, 2, 0.4	0.1, 0.9, 0.1, 0.02 1, 0.2
Breast	0.3, 0.2, 0.1, 0.015	0.3, 3, 0.6	0.3, 0.2, 0.1, 0.02 3, 0.6
Rice	0.1, 0.2, 0.1, 0.01	0.2, 1, 0.2	0.1, 0.2, 0.1, 0.01 3, 0.6
WDBC	0.1, 0.2, 0.075, 0.002	0.1, 2, 0.4	0.1, 0.2, 0.075, 0.002 0.2, 0.04
NHANES	0.1, 0.9, 0.1, 0.015	0.1, 3, 0.6	0.1, 0.9, 0.1, 0.01 2, 0.4

Initially, we consider the noiseless case for the linear kernel in Table 8. It can be observed that:

Table 3 The optimal parameters of ATGP-SVM, TPin-SVM and GP-SVM on UCI data with noise ($\sigma^2 = 0.1$) using a linear kernel.

Datasets	GP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$	TPin-SVM τ, α_1, α_2	ATGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ α_1, α_2
Ionosphere	0.2, 0.1, 0.075, 0.015	0.3, 2, 0.4	0.2, 0.1, 0.05, 0.015 1, 0.2
Pima	0.3, 0.1, 0.075, 0.015	0.1, 2, 0.4	0.3, 0.1, 0.075, 0.015 0.5, 0.1
Diabetes	0.1, 0.1, 0.01, 0.005	0.9, 2, 0.4	0.1, 0.1, 0.01, 0.005 0.2, 0.04
Bupa	0.1, 0.9, 0.1, 0.002	0.5, 2, 0.4	0.1, 0.9, 0.1, 0.002 0.2, 0.04
Sonar	0.1, 0.2, 0.1, 0.01	0.1, 2, 0.4	0.1, 0.2, 0.1, 0.01 1, 0.2
Heart Failure Clinical Records	0.1, 0.9, 0.075, 0.015	0.1, 1, 0.2	0.1, 0.9, 0.1, 0.02 2, 0.4
Breast	0.9, 0.1, 0.1, 0.015	0.3, 3, 0.6	0.3, 0.1, 0.1, 0.015 2, 0.4
Rice	0.1, 0.2, 0.1, 0.01	0.2, 1, 0.2	0.2, 0.2, 0.1, 0.01 2, 0.4
WDBC	0.9, 0.1, 0.01, 0.002	0.1, 2, 0.4	0.9, 0.1, 0.01, 0.002 1, 0.2
NHANES	0.1, 0.9, 0.1, 0.015	0.1, 3, 0.6	0.1, 0.1, 0.075, 0.015 3, 0.6

- ATGP-SVM and GP-SVM frequently achieve the highest accuracy, MCC values, and F1-Scores across datasets, demonstrating strong overall classification performance.
- TPin-SVM shows slightly better accuracy for datasets such as "Breast" and "Rice."
- CSVM outperforms in accuracy and MCC for datasets like "Heart Failure Clinical Records."
- CSVM is typically the fastest, requiring the least computational time across datasets.

In summary, ATGP-SVM and GP-SVM excel in accuracy, MCC, and F1-

Table 4 The optimal parameters of ATGP-SVM, TPin-SVM and GP-SVM on UCI data with noise ($\sigma^2 = 1.0$) using a linear kernel.

Datasets	GP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$	TPin-SVM τ, α_1, α_2	ATGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ α_1, α_2
Ionosphere	0.3, 0.1, 0.075, 0.015	0.3, 2, 0.4	0.3, 0.1, 0.075, 0.015 1, 0.2
Pima	0.3, 0.1, 0.075, 0.015	0.1, 2, 0.4	0.3, 0.1, 0.075, 0.015 0.5, 0.1
Diabetes	0.9, 0.1, 0.01, 0.002	0.1, 2, 0.4	0.1, 0.1, 0.075, 0.002 0.2, 0.04
Bupa	0.1, 0.1, 0.1, 0.002	0.1, 2, 0.4	0.1, 0.1, 0.1, 0.002 0.2, 0.04
Sonar	0.9, 0.1, 0.075, 0.02	0.1, 2, 0.4	0.1, 0.2, 0.01, 0.002 0.2, 0.04
Heart Failure Clinical Records	0.1, 0.9, 0.075, 0.015	0.1, 1, 0.2	0.1, 0.9, 0.1, 0.02 2, 0.4
Breast	0.9, 0.1, 0.1, 0.015	0.1, 2, 0.4	0.3, 0.3, 0.1, 0.02 2, 0.4
Rice	0.1, 0.2, 0.1, 0.01	0.2, 1, 0.2	0.2, 0.2, 0.1, 0.01 2, 0.4
WDBC	0.9, 0.2, 0.1, 0.002	0.2, 2, 0.4	0.9, 0.2, 0.1, 0.02 2, 0.4
NHANES	0.1, 0.9, 0.1, 0.01	0.1, 3, 0.6	0.1, 0.9, 0.1, 0.002 0.2, 0.04

Score, while TPin-SVM shows slightly better accuracy for some datasets, and CSVM is preferred for time-sensitive tasks. The results for ATGP-SVM and GP-SVM are similar, with the same optimal parameter values τ_1, τ_2, ϵ_1, and ϵ_2 for both methods, and ATGP-SVM's parameters α_1 and α_2 are notably large. This suggests that the $L_{tgp}^{\alpha_1,\alpha_2}$-loss tends to resemble the $L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}$-loss when α_1 and α_2 are large. Both methods exhibit strong performance compared to other methods on datasets without noise. Next, we consider the noisy cases, as detailed in Tables 9 and 10:

- An increase in noise variance (σ^2) leads to a discernible reduction in ACC, MCC, and F1-Score across all methodologies.
- ATGP-SVM consistently demonstrates superior performance across multiple

Table 5 The optimal parameters of ATGP-SVM, TPin-SVM and GP-SVM on UCI data without noise using RBF kernel.

Datasets	GP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$	TPin-SVM τ, α_1, α_2	ATGP-SVM $\tau_1, \tau_2, \epsilon_1$ $\epsilon_2, \alpha_1, \alpha_2$
Ionosphere	0.9, 0.2, 0.1, 0.02	0.2, 3, 0.6	0.9, 0.2, 0.1, 0.02 2, 0.04
Pima	0.9, 0.1, 0.1, 0.02	0.2, 3, 0.6	0.9, 0.1, 0.1, 0.02 3, 0.6
Diabetes	0.1, 0.9, 0.1, 0.02	0.1, 3, 0.6	0.1, 0.9, 0.1, 0.02 3, 0.6
Bupa	0.1, 0.9, 0.1, 0.02	0.1, 3, 0.6	0.1, 0.9, 0.1, 0.02 3, 0.6
Sonar	0.1, 0.9, 0.1, 0.02	0.5, 1, 0.2	0.1, 0.9, 0.1, 0.02 3, 0.6
Heart Failure Clinical Records	0.1, 0.9, 0.075, 0.015	0.3, 2, 0.4	0.1, 0.9, 0.075, 0.015 3, 0.6
Breast	0.3, 0.2, 0.1, 0.015	0.3, 3, 0.6	0.3, 0.2, 0.1, 0.015 3, 0.6
Rice	0.1, 0.2, 0.1, 0.01	0.2, 1, 0.2	0.1, 0.2, 0.1, 0.01 2, 0.4
WDBC	0.9, 0.1, 0.1, 0.02	0.1, 3, 0.6	0.9, 0.1, 0.1, 0.02 3, 0.6
NHANES	0.1, 0.9, 0.1, 0.015	0.1, 3, 0.6	0.1, 0.9, 0.1, 0.02 2, 0.4

datasets and noise levels, maintaining high accuracy, MCC, and F1-Scores even with increased noise variance.

- TPin-SVM also shows strong performance, particularly in datasets like Pima and Diabetes, remaining competitive compared to other SVM variants.
- CSVM and GP-SVM exhibit higher sensitivity to noise levels, which is evident from the reduced performance measurements, possibly due to parameter limitations.
- CSVM remains the fastest, requiring the least computational time across datasets.

When comparing ATGP-SVM with all baseline methods, ATGP-SVM gener-

Table 6 The optimal parameters of ATGP-SVM, TPin-SVM and GP-SVM on UCI data with noise ($\sigma^2 = 0.1$) using RBF kernel.

Datasets	GP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$	TPin-SVM τ, α_1, α_2	ATGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ α_1, α_2
Ionosphere	0.9, 0.2, 0.1, 0.02	0.2, 3, 0.6	0.9, 0.1, 0.1, 0.02 2, 0.4
Pima	0.9, 0.2, 0.1, 0.02	0.2, 3, 0.6	0.9, 0.1, 0.1, 0.02 3, 0.6
Diabetes	0.1, 0.9, 0.1, 0.02	0.1, 3, 0.6	0.1, 0.9, 0.1, 0.02 1, 0.2
Bupa	0.1, 0.9, 0.1, 0.02	0.1, 2, 0.4	0.1, 0.9, 0.1, 0.015 0.5, 0.1
Sonar	0.1, 0.9, 0.075, 0.02	0.9, 1, 0.2	0.5, 0.9, 0.075, 0.02 2, 0.4
Heart Failure Clinical Records	0.1, 0.9, 0.075, 0.075	0.3, 2, 0.4	0.1, 0.9, 0.1, 0.02 0.2, 0.04
Breast	0.3, 0.2, 0.1, 0.015	0.3, 3, 0.6	0.3, 0.2, 0.1, 0.015 0.5, 0.1
Rice	0.1, 0.5, 0.1, 0.01	0.3, 2, 0.4	0.1, 0.9, 0.1, 0.01 1, 0.2
NHANES	0.1, 0.9, 0.1, 0.01	0.1, 3, 0.6	0.1, 0.9, 0.1, 0.02 1, 0.2

ally outperforms the baseline methods, especially in scenarios with significant noise interference. This further validates the effectiveness of the $L_{tgp}^{\alpha_1,\alpha_2}$-loss. We then consider the noiseless case for the RBF kernel, as shown in Table 11. It can be observed that:

- ATGP-SVM and GP-SVM often achieve high accuracy and F1-Scores, particularly on datasets like Ionosphere and Pima.
- TPin-SVM shows strong performance in the Diabetes dataset, achieving the highest accuracy and F1-Score.
- CSVM excels in achieving the highest F1-Scores in several datasets, including Diabetes and Heart Failure Clinical Records.
- CSVM generally has the shortest processing time across most datasets, mak-

Table 7 The optimal parameters of ATGP-SVM, TPin-SVM and GP-SVM on UCI data with noise ($\sigma^2 = 1.0$) using a RBF kernel.

Datasets	GP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$	TPin-SVM τ, α_1, α_2	ATGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ α_1, α_2
Ionosphere	0.9, 0.2, 0.1, 0.02	0.1, 3, 0.6	0.9, 0.1, 0.1, 0.015 2, 0.4
Pima	0.9, 0.1, 0.1, 0.02	0.1, 3, 0.6	0.9, 0.1, 0.1, 0.02 2, 0.4
Diabetes	0.1, 0.9, 0.1, 0.02	0.1, 2, 0.4	0.1, 0.9, 0.1, 0.02 0.5, 0.1
Bupa	0.1, 0.9, 0.1, 0.02	0.1, 2, 0.4	0.1, 0.9, 0.1, 0.02 0.2, 0.04
Sonar	0.1, 0.9, 0.075, 0.015	0.9, 1, 0.2	0.5, 0.9, 0.075, 0.02 1, 0.2
Heart Failure Clinical Records	0.9, 0.2, 0.1, 0.1	0.2, 2, 0.4	0.9, 0.3, 0.1, 0.02 3, 0.6
Breast	0.3, 0.2, 0.1, 0.015	0.3, 3, 0.6	0.1, 0.3, 0.1, 0.015 1, 0.2
Rice	0.2, 0.5, 0.1, 0.01	0.2, 2, 0.4	0.1, 0.9, 0.1, 0.015 2, 0.4
WDBC	0.9, 0.1, 0.1, 0.02	0.1, 2, 0.4	0.9, 0.1, 0.1, 0.02 2, 0.4
NHANES	0.1, 0.9, 0.1, 0.015	0.1, 3, 0.6	0.1, 0.5, 0.1, 0.01 3, 0.6

ing it the most efficient in terms of computation. GP-SVM also performs well in terms of processing time on several datasets.

In summary, while ATGP-SVM and GP-SVM are competitive in accuracy and F1-Scores, CSVM stands out for its efficiency and speed, similar to the noiseless case for the linear kernel. Finally, we review the noisy case for the RBF kernel in Tables 12 and 13. It can be observed that:

- ATGP-SVM frequently performs the best or is on par with other methods in terms of accuracy and F1-Score across datasets and noise levels.
- TPin-SVM and GP-SVM generally show competitive results but do not consistently outperform ATGP-SVM.

- CSVM tends to have lower performance in terms of accuracy and F1-Score compared to ATGP-SVM, TPin-SVM, and GP-SVM but often exhibits better time efficiency.

Overall, ATGP-SVM frequently achieves the highest accuracy and F1-Score across various datasets and noise conditions, while CSVM usually excels in computation time.

Based on the experimental outcomes, it is clear that ATGP-SVM performs commendably even on datasets without noise and excels particularly well with increased noise levels, outperforming other baseline methods in such scenarios. ATGP-SVM's flexibility to adjust parameters and the boundedness of its loss function contribute to its superior performance. Regarding time complexity, ATGP-SVM and TPin-SVM take nearly ten times the computation time of GP-SVM and CSVM due to the requirement of 10 iterations ($I = 10$), which involves calculating the quadratic programming problem 10 times. In contrast, GP-SVM and CSVM compute the quadratic programming problem only once. However, the number of iterations can be reduced to lower computation time while maintaining model performance under certain conditions.

Table 8 Comparison of ATGP-SVM, TPin-SVM, GP-SVM and CSVM on UCI data without noise using a linear kernel.

Dataset	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time (Seconds)
Ionosphere (350 × 34)	ATGP-SVM	**86.34 ± 7.44**	**70.19 ± 9.01**	**88.80 ± 7.02**	0.3818
	TPin-SVM	84.91 ± 6.94	65.67 ± 5.65	87.81 ± 6.84	0.3866
	GP-SVM	**86.34 ± 7.44**	**70.19 ± 9.01**	**88.80 ± 7.02**	0.0364
	CSVM	85.77 ± 7.37	67.32 ± 8.70	88.21 ± 7.31	0.0181
Pima (768 × 8)	ATGP-SVM	**78.12 ± 2.16**	**49.82 ± 5.44**	63.11 ± 3.89	0.2577
	TPin-SVM	77.73 ± 3.63	49.15 ± 8.65	**63.67 ± 5.77**	0.2570
	GP-SVM	**78.12 ± 2.16**	**49.82 ± 5.44**	63.11 ± 3.89	0.2190
	CSVM	77.34 ± 3.24	48.32 ± 7.73	63.54 ± 5.26	0.0153
Diabetes (1151 × 19)	ATGP-SVM	78.08 ± 5.31	**49.16 ± 7.42**	62.77 ± 3.96	0.2559
	TPin-SVM	**78.08 ± 4.22**	49.08 ± 5.26	**63.53 ± 2.23**	0.2487
	GP-SVM	78.08 ± 5.31	**49.16 ± 7.42**	62.77 ± 3.96	0.0243
	CSVM	77.67 ± 4.12	48.20 ± 4.89	63.25 ± 1.57	0.0178
Bupa (345 × 7)	ATGP-SVM	**66.38 ± 2.81**	**32.16 ± 4.87**	**55.94 ± 3.25**	0.1309
	TPin-SVM	65.80 ± 2.69	31.59 ± 2.06	53.20 ± 2.18	0.1269
	GP-SVM	**66.38 ± 2.81**	**32.16 ± 4.87**	**55.94 ± 3.25**	0.0113
	CSVM	65.22 ± 2.90	30.43 ± 1.77	51.16 ± 3.37	0.0077
Sonar (208 × 60)	ATGP-SVM	66.09 ± 2.35	31.75 ± 4.93	**54.73 ± 2.73**	0.4468
	TPin-SVM	**67.25 ± 4.26**	**33.90 ± 3.83**	53.97 ± 3.71	0.4336
	GP-SVM	66.09 ± 2.35	31.75 ± 4.93	**54.73 ± 2.73**	0.0419
	CSVM	65.22 ± 2.90	30.43 ± 1.77	51.16 ± 3.37	0.0210
Heart Failure Clinical Records (299 × 12)	ATGP-SVM	79.93 ± 6.06	53.10 ± 9.88	63.35 ± 8.88	0.4987
	TPin-SVM	80.61 ± 5.61	54.60 ± 7.98	66.69 ± 5.86	0.4825
	GP-SVM	79.93 ± 6.06	53.10 ± 9.88	63.35 ± 8.88	0.0457
	CSVM	**80.62 ± 6.26**	**54.69 ± 9.66**	**66.88 ± 6.85**	0.0242
Breast (116 × 9)	ATGP-SVM	71.45 ± 7.30	**43.91 ± 15.25**	**70.77 ± 9.18**	0.0785
	TPin-SVM	**72.32 ± 7.28**	46.82 ± 13.07	70.24 ± 10.04	0.0751
	GP-SVM	71.45 ± 7.30	**43.91 ± 15.25**	**70.77 ± 9.18**	0.0069
	CSVM	67.14 ± 9.59	32.05 ± 21.42	62.14 ± 15.23	0.0050
Rice (3810 × 7)	ATGP-SVM	92.89 ± 0.96	85.45 ± 2.09	93.81 ± 0.74	2.4929
	TPin-SVM	**93.07 ± 0.82**	**85.82 ± 1.77**	**94.02 ± 0.65**	2.0061
	GP-SVM	92.89 ± 0.96	85.45 ± 2.09	93.81 ± 0.74	0.2319
	CSVM	92.97 ± 0.61	85.65 ± 1.34	93.85 ± 0.52	0.1060
WDBC (569 × 30)	ATGP-SVM	96.49 ± 2.15	92.52 ± 3.84	95.31 ± 2.39	0.5001
	TPin-SVM	95.96 ± 2.75	91.53 ± 5.17	94.88 ± 2.71	0.5228
	GP-SVM	96.49 ± 2.15	92.52 ± 3.84	95.31 ± 2.39	0.0478
	CSVM	**97.01 ± 1.31**	**93.41 ± 2.21**	**95.91 ± 1.20**	0.0298
NHANES (6287 × 7)	ATGP-SVM	**99.69 ± 0.22**	**98.80 ± 0.94**	**98.97 ± 0.83**	1.2258
	TPin-SVM	97.85 ± 0.61	91.74 ± 2.69	92.96 ± 2.36	0.5228
	GP-SVM	**99.69 ± 0.22**	**98.80 ± 0.94**	**98.97 ± 0.83**	0.1092
	CSVM	99.52 ± 0.43	98.09 ± 1.84	98.37 ± 1.59	0.0654

Table 9 Comparison of ATGP-SVM, TPin-SVM, GP-SVM and CSVM on UCI data with noise using a linear kernel.

Dataset		Criteria	ACC (%)	MCC (%)	F_1 (%)	Time (Seconds)
Ionosphere	$\sigma^2 = 0.1$	ATGP-SVM	**84.36 ± 7.84**	**66.00 ± 8.04**	**87.74 ± 7.16**	0.3668
		TPin-SVM	82.94 ± 7.81	62.60 ± 4.89	86.80 ± 7.18	0.3765
		GP-SVM	84.07 ± 7.49	65.14 ± 7.20	87.57 ± 6.95	0.0351
		CSVM	81.79 ± 9.22	58.86 ± 13.37	85.38 ± 8.82	0.0190
	$\sigma^2 = 1.0$	ATGP-SVM	**76.08 ± 11.30**	**46.89 ± 17.98**	**81.40 ± 10.05**	0.3715
		TPin-SVM	73.81 ± 11.02	41.00 ± 18.01	79.58 ± 10.12	0.3895
		GP-SVM	75.52 ± 11.57	45.66 ± 17.95	80.92 ± 10.34	0.0344
		CSVM	73.52 ± 10.13	41.19 ± 17.38	78.99 ± 9.87	0.0181
Pima	$\sigma^2 = 0.1$	ATGP-SVM	**77.34 ± 2.68**	**48.48 ± 6.37**	**64.24 ± 4.68**	0.2550
		TPin-SVM	77.21 ± 2.70	48.20 ± 6.42	63.98 ± 4.56	0.2542
		GP-SVM	77.08 ± 3.28	47.64 ± 7.83	62.96± 5.42	0.0240
		CSVM	77.08 ± 2.68	47.85 ± 6.38	63.56 ± 4.46	0.0149
	$\sigma^2 = 1.0$	ATGP-SVM	**71.88 ± 2.22**	33.30 ± 6.38	47.64 ± 7.19	0.2504
		TPin-SVM	71.88 ± 3.49	**33.97 ± 9.00**	**50.64 ± 7.40**	0.2437
		GP-SVM	71.49± 2.28	32.32 ± 6.41	44.71 ± 5.04	0.0226
		CSVM	71.36 ± 3.17	32.48 ± 8.47	49.26 ± 7.26	0.0157
Diabetes	$\sigma^2 = 0.1$	ATGP-SVM	**77.26 ± 4.69**	**47.25± 6.26**	62.35± 3.17	0.2535
		TPin-SVM	77.26± 4.83	47.25± 6.68	**62.56± 3.28**	0.2537
		GP-SVM	76.71 ± 4.22	45.63± 4.84	60.25± 2.17	0.0265
		CSVM	77.12 ± 4.56	47.01 ± 5.54	61.92± 2.88	0.0199
	$\sigma^2 = 1.0$	ATGP-SVM	**73.01 ± 4.12**	36.56 ± 4.18	52.99 ± 3.01	0.2513
		TPin-SVM	72.74 ± 4.34	**36.67 ± 4.87**	**53.13 ± 4.22**	0.2456
		GP-SVM	72.88 ± 5.35	36.07 ± 6.39	50.94± 5.00	0.0216
		CSVM	72.74± 4.00	35.96± 4.32	52.72 ± 3.20	0.0240
Bupa	$\sigma^2 = 0.1$	ATGP-SVM	62.90 ± 4.45	26.13± 8.36	48.64 ± 6.50	0.1260
		TPin-SVM	**64.93 ± 3.82**	**30.04 ± 6.27**	**52.32 ± 5.21**	0.1199
		GP-SVM	62.90± 4.45	26.13± 8.36	48.64± 6.50	0.0110
		CSVM	62.03± 3.82	25.84± 7.28	46.51 ± 4.86	0.0075
	$\sigma^2 = 1.0$	ATGP-SVM	57.10± 4.64	14.34± 9.99	**36.61 ± 7.11**	0.1216
		TPin-SVM	**57.97± 3.43**	**15.82± 4.09**	34.97 ± 10.63	0.1191
		GP-SVM	57.10 ± 4.64	14.34 ± 9.99	**36.61 ± 7.11**	0.0108
		CSVM	55.36 ± 5.61	10.59 ± 9.64	25.31 ± 14.62	0.0074
Sonar	$\sigma^2 = 0.1$	ATGP-SVM	57.68 ± 2.66	15.63 ± 2.42	34.78± 10.92	0.4517
		TPin-SVM	57.10 ± 3.62	14.60 ± 4.50	31.80 ± 15.43	0.4406
		GP-SVM	57.39 ± 3.38	15.50 ± 4.84	33.59 ± 13.20	0.0437
		CSVM	**58.26 ± 2.49**	**17.56 ± 6.76**	**38.54 ± 6.05**	0.0286
	$\sigma^2 = 1.0$	ATGP-SVM	**56.81± 8.57**	**7.80 ± 9.56**	17.10 ± 21.20	0.4513
		TPin-SVM	55.36 ± 6.82	7.76 ± 6.64	**27.30 ± 12.61**	0.4662
		GP-SVM	54.49 ± 6.46	3.98 ± 5.27	20.03± 17.71	0.0435
		CSVM	54.20± 6.71	3.18 ± 6.14	19.93± 17.70	0.0226

Table 10 Comparison of ATGP-SVM, TPin-SVM, GP-SVM and CSVM on UCI data with noise using a linear kernel (continued).

Dataset		Criteria	ACC (%)	MCC (%)	F_1 (%)	Time (Seconds)
Heart Failure Clinical Records	$\sigma^2 = 0.1$	ATGP-SVM	79.94 ± 5.36	52.32 ± 10.58	**65.40± 9.65**	0.1585
		TPin-SVM	79.28 ± 4.24	50.87 ± 5.89	64.76 ± 4.76	0.1443
		GP-SVM	**80.28 ± 5.27**	**52.62 ± 7.79**	65.09 ± 5.70	0.0133
		CSVM	78.94 ± 5.39	49.60 ± 7.33	63.18 ± 5.25	0.0079
	$\sigma^2 = 1.0$	ATGP-SVM	**76.95± 6.03**	**43.42 ± 13.75**	**57.89 ± 11.90**	0.1495
		TPin-SVM	74.92 ± 4.60	39.38 ± 8.02	55.79± 6.51	0.1391
		GP-SVM	76.60 ± 4.14	40.90 ± 9.76	54.66± 10.19	0.0129
		CSVM	75.94± 5.05	39.91 ± 10.51	55.16± 10.32	0.0082
Breast	$\sigma^2 = 0.1$	ATGP-SVM	**69.78 ± 7.91**	**38.46 ± 20.34**	**64.67 ± 18.42**	0.0790
		TPin-SVM	**69.78 ± 7.91**	35.33 ± 22.66	60.92 ± 22.18	0.0758
		GP-SVM	**69.78 ± 7.91**	37.96 ± 18.53	63.58 ± 17.39	0.0067
		CSVM	68.08 ± 7.14	31.88 ± 20.83	59.36 ± 21.13	0.0052
	$\sigma^2 = 1.0$	ATGP-SVM	**61.27± 8.77**	**23.24 ± 18.41**	53.50 ± 15.81	0.0765
		TPin-SVM	59.57 ± 10.07	14.71 ± 21.08	48.04 ± 17.39	0.0737
		GP-SVM	61.23 ± 5.93	22.89 ± 15.04	**54.39± 14.28**	0.0066
		CSVM	59.49± 5.78	20.00 ± 13.68	51.05± 13.05	0.0048
Rice	$\sigma^2 = 0.1$	ATGP-SVM	**91.81 ± 0.60**	83.28 ± 1.31	**92.87± 0.56**	1.8641
		TPin-SVM	91.81 ± 0.66	**83.33 ± 1.48**	92.80 ± 0.52	1.8221
		GP-SVM	91.68 ± 0.79	83.02 ± 1.70	92.75 ± 0.69	0.2122
		CSVM	91.63 ± 0.84	82.93 ± 1.78	92.69 ± 0.77	0.1034
	$\sigma^2 = 1.0$	ATGP-SVM	**86.46± 1.42**	**72.24 ± 3.06**	**88.34 ± 1.20**	1.8013
		TPin-SVM	86.09 ± 1.37	71.60± 2.82	87.87± 1.28	1.8020
		GP-SVM	86.35 ± 1.43	72.05 ± 3.05	88.18± 1.23	0.2062
		CSVM	86.40± 1.40	72.16 ± 3.03	88.21± 1.18	0.0993
WDBC	$\sigma^2 = 0.1$	ATGP-SVM	94.91 ± 3.06	89.00 ± 5.38	93.34 ± 2.07	0.4987
		TPin-SVM	94.38 ± 3.07	87.83 ± 5.49	92.44± 2.80	0.5413
		GP-SVM	95.08 ± 3.45	89.46 ± 6.70	**93.68 ± 3.51**	0.0492
		CSVM	**95.26 ± 2.39**	**89.75 ± 4.50**	93.56 ± 2.64	0.0326
	$\sigma^2 = 1.0$	ATGP-SVM	**90.86 ± 4.62**	**80.67 ± 6.90**	**87.46 ± 3.45**	0.4534
		TPin-SVM	90.51 ± 4.88	80.00 ± 7.38	86.95 ± 3.94	0.4532
		GP-SVM	90.69± 4.52	80.27 ± 6.44	87.17 ± 2.94	0.0436
		CSVM	89.10 ± 2.83	76.63± 5.46	84.51 ± 4.79	0.0277
NHANES	$\sigma^2 = 0.1$	ATGP-SVM	94.34 ± 1.19	78.02 ± 5.19	80.91 ± 4.48	0.9473
		TPin-SVM	94.34 ± 1.29	78.00± 5.01	80.85 ± 4.35	0.8948
		GP-SVM	94.16 ± 1.43	78.02 ± 5.52	81.24± 4.76	0.0979
		CSVM	**94.34± 1.17**	**78.20 ± 4.78**	**81.30± 4.07**	0.0550
	$\sigma^2 = 1.0$	ATGP-SVM	**87.05 ± 1.79**	**45.96 ± 4.45**	**51.92 ± 3.73**	0.9425
		TPin-SVM	86.65 ± 1.52	38.26 ± 3.74	37.12 ± 5.24	0.9336
		GP-SVM	87.01 ± 1.76	45.80 ± 4.21	51.82 ± 3.62	0.0033
		CSVM	86.92± 1.30	40.54 ± 1.90	41.22 ± 3.60	0.0452

Table 11 Comparison of ATGP-SVM, TPin-SVM, GP-SVM and CSVM on UCI data without noise using a RBF kernel.

Dataset	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time (Seconds)
Ionosphere (350 × 34)	ATGP-SVM	**90.04 ± 2.35**	**79.00 ± 3.82**	**92.65 ± 1.93**	0.4342
	TPin-SVM	89.75 ± 2.94	78.44 ± 3.87	92.45 ± 1.99	0.4555
	GP-SVM	**90.04 ± 2.35**	**79.00 ± 3.82**	**92.65 ± 1.93**	0.0458
	CSVM	89.47 ± 2.89	77.88 ± 4.95	92.26 ± 2.30	0.0245
Pima (768 × 8)	ATGP-SVM	**74.87 ± 2.32**	**41.97 ± 2.57**	53.77 ± 4.91	0.4443
	TPin-SVM	74.48 ± 2.46	41.02 ± 3.15	**52.94 ± 5.38**	0.4697
	GP-SVM	**74.87 ± 2.32**	**41.97 ± 2.57**	53.77 ± 4.91	0.0410
	CSVM	74.74 ± 2.44	41.69 ± 3.00	**54.33 ± 4.70**	0.0278
Diabetes (299 × 12)	ATGP-SVM	73.70 ± 1.82	39.57 ± 1.84	56.87 ± 2.29	0.3372
	TPin-SVM	**80.61 ± 5.61**	54.60 ± 7.98	**66.69 ± 5.86**	0.4825
	GP-SVM	73.70 ± 1.82	39.57 ± 1.84	56.87 ± 2.29	0.0301
	CSVM	72.27 ± 2.96	34.60 ± 2.70	46.09 ± 3.74	0.0163
Bupa (345 × 5)	ATGP-SVM	**61.74 ± 3.73**	**19.82 ± 7.33**	**49.82 ± 4.86**	0.1618
	TPin-SVM	58.84 ± 5.62	6.44 ± 9.14	12.06 ± 14.77	0.1590
	GP-SVM	**61.74 ± 3.73**	**19.82 ± 7.33**	**49.82 ± 4.86**	0.0144
	CSVM	57.97 ± 4.58	4.93 ± 7.39	16.30 ± 18.45	0.0094
Sonar (208 × 60)	ATGP-SVM	**57.22 ± 7.11**	**13.47 ± 15.51**	**51.22 ± 10.05**	0.5362
	TPin-SVM	55.28 ± 7.96	10.75 ± 15.49	47.78 ± 9.84	0.4843
	GP-SVM	**57.22 ± 7.11**	**13.47 ± 15.51**	**51.22± 10.85**	0.0565
	CSVM	53.09 ± 6.19	6.74 ± 12.69	47.62 ± 6.64	0.0241
Heart Failure Clinical Records (299 × 12)	ATGP-SVM	79.93 ± 6.06	53.10 ± 9.88	63.35 ± 8.88	0.4987
	TPin-SVM	80.61 ± 5.61	54.60 ± 7.98	**66.69 ± 5.86**	0.4825
	GP-SVM	79.93 ± 6.06	53.10 ± 9.88	63.35 ± 8.88	0.0457
	CSVM	**80.62± 6.26**	**54.69 ± 9.66**	66.88 ± 6.85	0.0242
Breast (116 × 9)	ATGP-SVM	**57.72 ± 3.64**	**17.35 ± 10.06**	**45.72 ± 10.55**	0.0850
	TPin-SVM	56.85 ± 4.29	12.07 ± 16.96	45.10 ± 16.18	0.0803
	GP-SVM	**57.72± 3.64**	**17.35± 10.06**	**45.72± 10.55**	0.0078
	CSVM	56.01 ± 2.04	9.72 ± 13.40	44.90 ± 14.89	0.0060
Rice (3810 × 7)	ATGP-SVM	**92.76 ± 0.72**	**85.19 ± 1.59**	**93.68 ± 0.54**	3.1768
	TPin-SVM	92.28 ± 0.91	84.42 ± 1.85	93.13 ± 0.76	2.9074
	GP-SVM	**92.76± 0.72**	**85.19± 1.59**	**93.68± 0.54**	0.3055
	CSVM	92.62 ± 0.70	84.97 ± 1.49	93.52 ± 0.63	0.1330
WDBC (569 × 30)	ATGP-SVM	96.13 ± 0.71	91.81 ± 1.27	94.67 ± 0.77	0.5891
	TPin-SVM	96.13 ± 0.71	91.81 ± 1.30	94.64 ± 0.75	0.5753
	GP-SVM	96.13 ± 0.71	91.81 ± 1.27	94.67 ± 0.77	0.0619
	CSVM	**96.48± 0.57**	**92.53 ± 1.01**	**95.20 ± 0.51**	0.0375
NHANES (6287 × 7)	ATGP-SVM	**96.93 ± 0.28**	**88.37 ± 1.38**	**90.11 ± 1.33**	2.7089
	TPin-SVM	88.63 ± 2.79	50.20 ± 10.16	46.88 ± 12.73	2.3376
	GP-SVM	**96.93± 0.28**	**88.38± 1.59**	**90.10± 1.53**	0.2962
	CSVM	94.03 ± 0.62	76.16 ± 3.42	78.94 ± 3.48	0.1736

Table 12 Comparison of ATGP-SVM, TPin-SVM, GP-SVM and CSVM on UCI data with noise using a RBF kernel.

Dataset		Criteria	ACC (%)	MCC (%)	F_1 (%)	Time (Seconds)
Ionosphere	$\sigma^2 = 0.1$	ATGP-SVM	**88.34 ± 4.99**	75.29 ± 10.30	**91.34 ± 3.80**	0.4189
		TPin-SVM	88.33 ± 4.67	**75.32 ± 9.51**	91.33 ± 3.60	0.4314
		GP-SVM	88.05 ± 5.09	74.74 ± 10.43	91.14 ± 3.91	0.0459
		CSVM	87.77 ± 4.59	73.91 ± 9.21	90.91 ± 3.60	0.0214
	$\sigma^2 = 1.0$	ATGP-SVM	**78.66 ± 5.80**	**53.60 ± 11.04**	**84.78 ± 4.35**	0.4146
		TPin-SVM	73.81 ± 11.02	41.00 ± 18.01	79.58 ± 10.12	0.4180
		GP-SVM	75.52 ± 11.57	45.66 ± 17.95	80.92 ± 10.34	0.0461
		CSVM	73.52 ± 10.13	41.19 ± 17.38	78.99 ± 9.87	0.0224
Pima	$\sigma^2 = 0.1$	ATGP-SVM	74.73 ± 2.72	41.50 ± 6.63	52.39 ± 4.87	0.4390
		TPin-SVM	74.08 ± 2.72	39.72 ± 6.79	50.09 ± 4.89	0.4377
		GP-SVM	74.60 ± 2.48	41.12 ± 6.05	52.23± 4.64	0.0393
		CSVM	**74.87 ± 2.17**	**41.74 ± 5.12**	**53.07 ± 4.73**	0.0243
	$\sigma^2 = 1.0$	ATGP-SVM	**70.44 ± 3.84**	**28.60 ± 10.14**	**38.29 ± 8.09**	0.4131
		TPin-SVM	70.05 ± 3.83	27.50 ± 9.49	36.26 ± 8.28	0.4110
		GP-SVM	70.05 ± 3.69	27.44 ± 9.92	37.62 ± 8.08	0.0360
		CSVM	70.05 ± 3.69	27.44 ± 9.92	37.62 ± 8.08	0.0247
Diabetes	$\sigma^2 = 0.1$	ATGP-SVM	**73.57 ± 2.48**	**38.94 ± 8.09**	**55.05 ± 6.08**	0.3309
		TPin-SVM	70.97 ± 2.45	31.31 ± 4.54	40.87 ± 4.07	0.4377
		GP-SVM	**73.57 ± 2.48**	**38.94 ± 8.09**	**55.05 ± 6.08**	0.0297
		CSVM	71.49 ± 1.75	32.59 ± 4.57	42.75 ± 3.34	0.0166
	$\sigma^2 = 1.0$	ATGP-SVM	**69.92 ± 1.74**	**29.42 ± 6.98**	**44.32 ± 3.17**	0.3132
		TPin-SVM	68.49 ± 3.60	23.97 ± 5.09	25.94 ± 5.72	0.2923
		GP-SVM	**69.92 ± 1.74**	**29.42 ± 6.98**	**44.32 ± 3.17**	0.0289
		CSVM	65.50 ± 4.27	5.68 ± 8.73	6.59 ± 8.75	0.0192
Bupa	$\sigma^2 = 0.1$	ATGP-SVM	**57.68 ± 8.52**	**11.41 ± 17.90**	**43.01 ± 10.80**	0.1618
		TPin-SVM	55.65 ± 6.12	4.59 ± 9.41	22.47 ± 16.61	0.1592
		GP-SVM	57.39± 8.07	10.80± 16.98	42.80 ± 10.28	0.0142
		CSVM	56.23 ± 6.70	1.67 ± 9.67	10.15 ± 12.89	0.0096
	$\sigma^2 = 1.0$	ATGP-SVM	**56.52 ± 7.04**	**9.08 ± 15.75**	**39.56 ± 8.44**	0.1571
		TPin-SVM	54.49 ± 10.31	3.29 ± 23.13	27.80 ± 14.06	0.1543
		GP-SVM	**56.52 ± 7.04**	**9.08 ± 15.75**	**39.56 ± 8.44**	0.0141
		CSVM	55.36 ± 7.58	3.60 ± 14.61	24.71 ± 15.44	0.0096
Sonar	$\sigma^2 = 0.1$	ATGP-SVM	**55.30 ± 10.90**	**10.90 ± 8.29**	**48.88 ± 6.79**	0.4998
		TPin-SVM	52.94 ± 5.58	5.45 ± 11.32	43.75 ± 5.47	0.4913
		GP-SVM	52.92 ± 6.35	7.09 ± 10.08	41.61 ± 7.81	0.0491
		CSVM	53.88 ± 4.30	8.14 ± 8.24	48.69 ± 4.04	0.0274
	$\sigma^2 = 1.0$	ATGP-SVM	**53.37± 8.57**	**9.25 ± 18.75**	46.84 ± 10.30	0.4940
		TPin-SVM	52.87 ± 5.70	8.47 ± 13.96	**51.22 ± 9.45**	0.5170
		GP-SVM	50.49± 7.58	3.11± 13.63	36.59 ± 14.94	0.0468
		CSVM	51.46 ± 6.98	5.49 ± 15.97	46.41 ± 10.16	0.0344

Table 13 Comparison of ATGP-SVM, TPin-SVM, GP-SVM and CSVM on UCI data with noise using a RBF kernel (continued).

Dataset	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time (Seconds)
Heart Failure Clinical Records	$\sigma^2 = 0.1$ ATGP-SVM	76.26 ± 4.10	**42.99 ± 8.69**	**59.48 ± 7.91**	0.4511
	TPin-SVM	75.59 ± 4.28	38.66 ± 7.43	47.88 ± 7.10	0.4377
	GP-SVM	72.91 ± 5.61	30.28 ± 5.08	36.60± 6.52	0.0372
	CSVM	**76.59 ± 4.95**	41.44 ± 6.06	52.38 ± 7.26	0.0189
	$\sigma^2 = 1.0$ ATGP-SVM	**71.25 ± 10.04**	26.17 ± 16.01	25.00 ± 18.08	0.4289
	TPin-SVM	70.58 ± 9.02	25.53 ± 9.88	24.41 ± 13.87	0.4165
	GP-SVM	70.58 ± 9.02	25.20 ± 10.69	26.75 ± 13.23	0.0386
	CSVM	70.91 ± 9.38	**26.42 ± 12.72**	**29.65 ± 13.53**	0.0192
Breast	$\sigma^2 = 0.1$ ATGP-SVM	**54.20± 8.92**	**9.10 ± 22.60**	39.66 ± 14.69	0.0857
	TPin-SVM	54.20 ± 13.05	7.62 ± 31.92	**44.47 ± 17.94**	0.0819
	GP-SVM	53.30 ± 13.62	7.19 ± 29.26	38.86± 18.32	0.0075
	CSVM	53.30 ± 14.95	3.25 ± 32.75	43.50 ± 18.55	0.0060
	$\sigma^2 = 1.0$ ATGP-SVM	**52.57 ± 2.86**	**5.37 ± 7.50**	**38.87 ± 7.64**	0.0835
	TPin-SVM	50.83 ± 4.66	4.23 ± 11.29	32.69 ± 7.23	0.0784
	GP-SVM	49.09 ± 6.06	3.53 ± 9.66	15.58 ± 14.52	0.0078
	CSVM	50.00 ± 4.35	2.17 ± 10.93	28.48 ± 10.05	0.0060
Rice	$\sigma^2 = 0.1$ ATGP-SVM	**91.84 ± 1.00**	**83.33 ± 2.10**	**92.89 ± 0.82**	2.8116
	TPin-SVM	91.31 ± 1.02	82.47 ± 2.02	92.26 ± 0.87	2.3788
	GP-SVM	91.78 ± 0.94	83.22 ± 2.00	92.85± 0.77	0.2742
	CSVM	91.31± 0.70	82.33 ± 1.49	92.40± 0.54	0.1198
	$\sigma^2 = 1.0$ ATGP-SVM	**85.14 ± 0.99**	**69.66 ± 2.27**	**87.05 ± 0.74**	2.8194
	TPin-SVM	84.62 ± 1.12	68.81 ± 2.52	86.38 ± 0.79	2.3411
	GP-SVM	84.17 ± 0.71	67.89 ± 1.87	85.99 ± 0.30	0.2119
	CSVM	84.62 ± 0.90	68.62 ± 2.02	86.54 ± 0.70	0.1359
WDBC	$\sigma^2 = 0.1$ ATGP-SVM	**95.96 ± 1.06**	**91.30 ± 2.43**	**94.34 ± 1.69**	0.5773
	TPin-SVM	95.78 ± 1.03	90.94± 2.36	94.09 ± 1.63	0.5669
	GP-SVM	**95.96 ± 1.06**	**91.30 ± 2.43**	**94.34± 1.69**	0.0539
	CSVM	95.43± 1.43	90.18 ± 3.04	93.68± 1.88	0.0488
	$\sigma^2 = 1.0$ ATGP-SVM	89.28 ± 2.78	76.84 ± 5.99	84.45 ± 4.23	0.6149
	TPin-SVM	89.46 ± 2.65	77.23 ± 5.70	84.74 ± 4.06	0.6231
	GP-SVM	89.28 ± 2.55	76.82 ± 5.59	84.50 ± 4.03	0.0594
	CSVM	**90.16 ± 1.94**	**78.81 ± 4.02**	**86.04 ± 3.10**	0.0339
NHANES	$\sigma^2 = 0.1$ ATGP-SVM	**93.15 ± 0.69**	**73.96 ± 1.88**	**77.88 ± 2.06**	2.6348
	TPin-SVM	86.08 ± 2.34	32.68± 8.57	25.41 ± 9.13	2.4375
	GP-SVM	93.06 ± 0.89	73.78 ± 1.82	77.74± 1.78	0.2694
	CSVM	91.09± 0.72	62.82 ± 4.28	65.72± 4.61	0.1726
	$\sigma^2 = 1.0$ ATGP-SVM	85.43 ± 0.76	39.55 ± 4.50	46.94 ± 4.18	2.4095
	TPin-SVM	84.55 ± 1.91	15.26 ± 7.96	8.60 ± 4.97	2.3494
	GP-SVM	**85.51 ± 0.90**	**39.89 ± 3.67**	**47.18 ± 3.36**	0.2119
	CSVM	84.59 ± 1.96	16.00 ± 8.60	9.63 ± 5.69	0.2177

3.1.7 STATISTICAL ANALYSIS

We will utilize the ACC, MCC, and F_1-score for linear kernel case from Tables 20 and 21 and RBF kernel case from Tables 26 and 27. In the case of the linear kernel, assuming the null hypothesis where all methods are considered equivalent, the computation of the Friedman statistic for accuracy (ACC) is as follows:

$$\begin{aligned}\chi_F^2 =& \frac{12 \times 30}{4 \times (4+1)} \\ & \times \left[(1.70^2 + 2.63^2 + 2.46^2 + 3.20^2) - \frac{4 \times 5^2}{4}\right] \\ \approx & 19.77, \\ F_F = & \frac{(30-1) \times 19.77}{30 \times (4-1) - 19.77} \approx 8.16.\end{aligned}$$

For Matthews Correlation Coefficient (MCC), the Friedman statistic is computed as follows:

$$\begin{aligned}\chi_F^2 =& \frac{12 \times 30}{4 \times (4+1)} \\ & \times \left[(1.78^2 + 2.70^2 + 2.45^2 + 3.06^2) - \frac{4 \times 5^2}{4}\right] \\ \approx & 14.85, \\ F_F = & \frac{(30-1) \times 14.85}{30 \times (4-1) - 14.85} \approx 5.73.\end{aligned}$$

Similarly, for the F_1-score, the Friedman statistic is computed as:

$$\begin{aligned}\chi_F^2 =& \frac{12 \times 30}{4 \times (4+1)} \\ & \times \left[(2.08^2 + 2.60^2 + 2.45^2 + 2.86^2) - \frac{4 \times 5^2}{4}\right] \\ \approx & 4.83, \\ F_F = & \frac{(30-1) \times 4.83}{30 \times (4-1) - 4.83} \approx 1.64.\end{aligned}$$

For the RBF kernel, assuming the null hypothesis where all methods are considered equivalent, the computation of the Friedman statistic for Accuracy (ACC) is as follows:

$$\chi_F^2 = \frac{12 \times 30}{4 \times (4+1)}$$

$$\times \left[(1.57^2 + 3.08^2 + 2.43^2 + 2.92^2) - \frac{4 \times 5^2}{4}\right]$$
$$\approx 24.87,$$
$$F_F = \frac{(30-1) \times 24.87}{30 \times (4-1) - 24.87} \approx 11.08.$$

For Matthews Correlation Coefficient (MCC), the Friedman statistic is computed as:

$$\chi_F^2 = \frac{12 \times 30}{4 \times (4+1)} \times \left[(1.63^2 + 3.00^2 + 2.45^2 + 2.92^2) - \frac{4 \times 5^2}{4}\right]$$
$$\approx 21.34,$$
$$F_F = \frac{(30-1) \times 21.34}{30 \times (4-1) - 21.34} \approx 9.02.$$

Similarly, for the F_1-score, the Friedman statistic is computed as:

$$\chi_F^2 = \frac{12 \times 18}{4 \times (4+1)} \times \left[(1.68^2 + 3.07^2 + 2.47^2 + 2.78^2) - \frac{4 \times 5^2}{4}\right]$$
$$\approx 11.87,$$
$$F_F = \frac{(18-1) \times 11.87}{18 \times (4-1) - 11.87} \approx 4.79.$$

At a 0.05 confidence level, the critical value of $F(3, 87)$ is 2.723. Thus, for ACC and MCC with the linear kernel ($8.16 > 2.723$ and $5.73 > 2.723$), and for ACC, MCC, and F_1-score with the RBF kernel ($11.08 > 2.723$, $9.02 > 2.723$, and $4.79 > 2.723$), the null hypothesis is rejected, indicating significant differences among the algorithms. However, for the F_1-score with the linear kernel ($1.64 < 2.723$), the null hypothesis is not rejected, suggesting no significant differences for this metric.

The Nemenyi post hoc test, with a critical difference $CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$ and $CD = 0.856$, shows that ATGP-SVM outperforms some baseline methods but does not significantly differ from others. Nonetheless, ATGP-SVM generally has the lowest average ranks for ACC, MCC, and F_1-score. Average ranks across different noise percentages, based on accuracy, are shown in Figure 13 and 14.

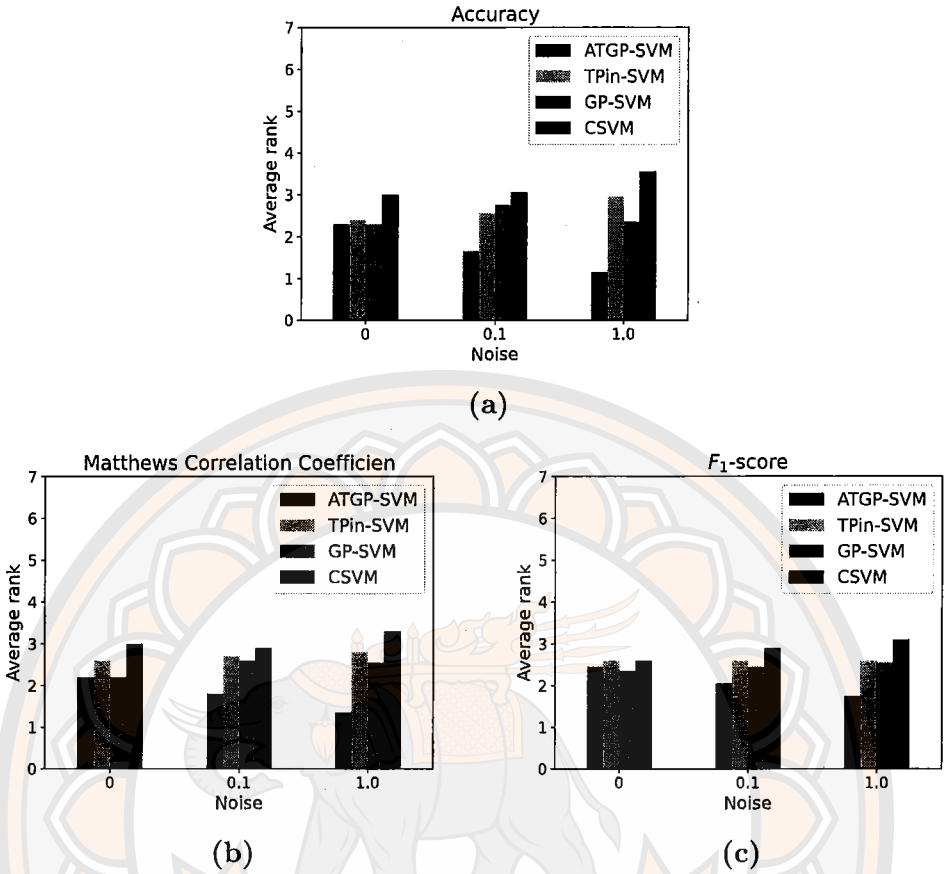


Figure 13 The average rank at different variance of noise for ACC, MCC, and F_1-score of each method using the linear kernel.

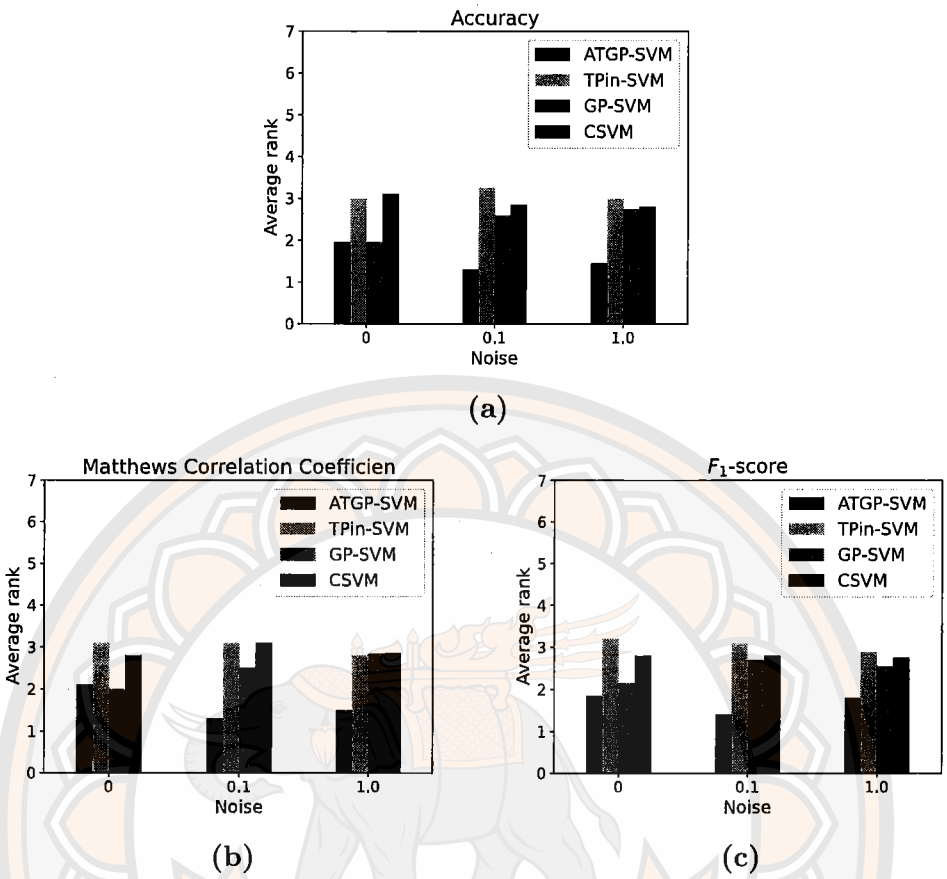


Figure 14 The average rank at different variance of noise for ACC, MCC, and F_1-score of each method using the RBF kernel.

3.2 Smooth support vector machine with rescaled generalized pinball loss

We introduce a smooth rescaled generalized pinball loss function (L_{srgp}), which enhances the smooth generalized pinball loss used in SGP-SVM through a rescaling technique. Based on this, we develop a robust SVM model named SRGP-SVM that incorporates L_{srgp}. This method effectively addresses noise, preserves sparsity, provides boundedness for managing outliers, and allows for greater flexibility in parameter adjustment, all while ensuring the differentiability of the objective functions. The L_{srgp} is given by

$$L_{srgp}(u) = \eta(1 - \exp(-\frac{L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u,\mu)}{\lambda}))$$

$$= \begin{cases} \eta(1-\exp(-\frac{\tau_1\left(u-\frac{\epsilon_1}{\tau_1}\right)-\frac{\tau_1^2}{2}\mu}{\lambda})), & \frac{\epsilon_1}{\tau_1}+\tau_1\mu \leq u, \\ \eta(1-\exp(-\frac{\frac{1}{2\mu}\left(u-\frac{\epsilon_1}{\tau_1}\right)^2}{\lambda})), & \frac{\epsilon_1}{\tau_1} < u < \frac{\epsilon_1}{\tau_1}+\tau_1\mu, \\ 0, & \frac{-\epsilon_2}{\tau_2} \leq u \leq \frac{\epsilon_1}{\tau_1}, \\ \eta(1-\exp(-\frac{\frac{1}{2\mu}\left(u+\frac{\epsilon_2}{\tau_2}\right)^2}{\lambda})), & \frac{-\epsilon_2}{\tau_2}-\tau_2\mu < u < \frac{-\epsilon_2}{\tau_2}, \\ \eta(1-\exp(-\frac{-\tau_2\left(u+\frac{\epsilon_2}{\tau_2}\right)-\frac{\tau_2^2}{2}\mu}{\lambda})), & u \leq \frac{-\epsilon_2}{\tau_2}-\tau_2\mu, \end{cases}$$

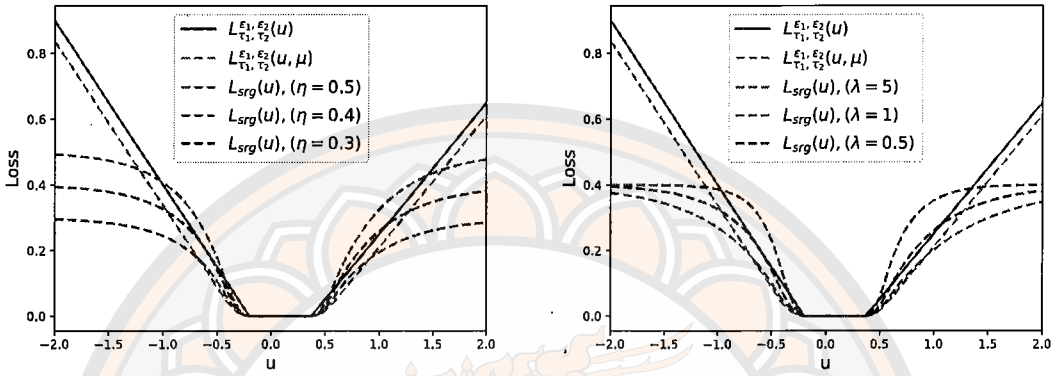
where parameters $0 \leq \tau_1, \tau_2 \leq 1, \eta, \lambda, > 0, \epsilon_1, \epsilon_2 \geq 0, \mu \in \mathbb{R}$ (see Figure 15). The L_{srgp} effectively manages the amount of loss through τ_1, while τ_2 further adjusts the loss and mitigates the impact of noise. Additionally, η serves as a constant that regulates the bounds of the loss function, and $\lambda > 0$ acts as a scale parameter that modulates its magnitude, similar to $L_{rp}(u)$. The L_{srgp} also defines the (ϵ_1, ϵ_2)-insensitive zone, where the loss value is zero, which lies between $-\epsilon_2/\tau_2$ and ϵ_1/τ_1. This reduces the loss along the support line, rendering it insignificant. This characteristic of L_{srgp} can also decrease computation time, as many components become zero, while μ controls the smoothness of the function, further highlighting the sparsity and smoothness of $L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}$. Therefore, L_{srgp} combines the advantages of both L_{rp} and $L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}$.

Furthermore, the smooth rescaled generalized pinball loss within SVM (SRGP-SVM) provides a flexible framework that encompasses variations such as the rescaled hinge loss within SVM (RH-SVM) [58] and the rescaled pinball loss within SVM (RP-SVM) [59]. In particular, when $\tau_1 = 1$ and $\tau_2 = \epsilon_1 = \epsilon_2 = 0$ with $\mu \to 0^+$, SRGP-SVM approximates SH-SVM. Similarly, SRGP-SVM approximates SP-SVM when $\tau_1 = 1$, $\tau_2 = \tau$, and $\epsilon_1 = \epsilon_2 = 0$ while $\mu \to 0^+$.

Without loss of generality, we can reformulate compact form of the loss function $L_{srgp}(u)$ as follows:

$$L_{srgp}(u) = \begin{cases} r(\tau_1(u - \theta_1) - \frac{\tau_1}{2}\delta), & \theta_1 + \delta \leq u, \\ r(\frac{\tau_1}{2\delta}(u-\theta_1)^2), & \theta_1 \leq u \leq \theta_1 + \delta, \\ 0, & -\theta_2 \leq u \leq \theta_1, \\ r(\frac{\tau_2}{2\delta}(u+\theta_2)^2), & -\theta_2 - \delta < u < -\theta_2 \\ r(-\tau_2(u+\theta_2) - \frac{\tau_2}{2}\delta), & u \leq -\theta_2 - \delta, \end{cases}$$

where $r(a) = \eta(1 - (\exp(-a)/\lambda)), a \in \mathbb{R}$. However, we only show a convenient way to write function formulas. Therefore, we still use the formula from the previous equation to prove the properties and to present the experimental results for easy comparison with the previous loss function.



(a) Variation in η (b) Variation in λ

Figure 15 Illustration of the smooth rescaled generalized pinball loss.

Below are additional principal attributes of the $L_{srgp}(u)$ function:

- The proposed L_{srgp} is a non-negative, asymmetric, bounded, non-convex loss function that adheres to the inequality:

$$0 \leq L_{srgp}(u) \leq \eta.$$

and

$$\lim_{u \to +\infty} L_{srgp}(u) = \eta, \lim_{u \to -\infty} L_{srgp}(u) = \eta.$$

This indicates that $L_{srgp}(u)$ is less sensitive to outliers than traditional losses such as hinge loss, pinball loss, ϵ-insensitive pinball loss and generalized pinball loss.

- $L_{srgp}(u)$ is differentiable and its derivative is also bounded. The gradient of

$L_{srgp}(u)$-loss is as follows:

$$\nabla L_{srgp}(u) = \begin{cases} \frac{\tau_1\eta}{\lambda}\exp(-\frac{\tau_1\left(u-\frac{\epsilon_1}{\tau_1}\right)-\frac{\tau_1^2}{2}\mu}{\lambda}), & \frac{\epsilon_1}{\tau_1}+\tau_1\mu \leq u, \\ \frac{\eta}{\lambda\mu}(u-\frac{\epsilon_1}{\tau_1})\exp(-\frac{\frac{1}{2\mu}\left(u-\frac{\epsilon_1}{\tau_1}\right)^2}{\lambda}), & \frac{\epsilon_1}{\tau_1} < u < \frac{\epsilon_1}{\tau_1}+\tau_1\mu, \\ 0, & \frac{-\epsilon_2}{\tau_2} \leq u \leq \frac{\epsilon_1}{\tau_1}, \\ \frac{\eta}{\lambda\mu}(u+\frac{\epsilon_2}{\tau_2})\exp(-\frac{\frac{1}{2\mu}\left(u+\frac{\epsilon_2}{\tau_2}\right)^2}{\lambda}), & \frac{-\epsilon_2}{\tau_2}-\tau_2\mu < u < \frac{-\epsilon_2}{\tau_2}, \\ -\frac{\tau_2\eta}{\lambda}\exp(-\frac{-\tau_2\left(u+\frac{\epsilon_2}{\tau_2}\right)-\frac{\tau_2^2}{2}\mu}{\lambda}), & u \leq \frac{-\epsilon_2}{\tau_2}-\tau_2\mu, \end{cases} \tag{3.2.1}$$

and we have

$$\lim_{u\to+\infty}\nabla L_{srgp}(u) = 0 \text{ and } \lim_{u\to-\infty}\nabla L_{srgp}(u) = 0$$

Thus the L_{srgp}-loss is a robust loss function in presence of outliers and noises. For a training point $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$, it does not contribute to hyperplane when $-\epsilon_2/\tau_2 \leq 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b) \leq \epsilon_1/\tau_1$. Therefore, the SRGP-SVM demonstrates a sparsity to some degree.

To analyze the asymptotic behavior of $L_{srgp}(u)$ under varying parameters, we consider the following proposition.

Proposition 3.2.1. *For any $u \in \mathbb{R}^n$ and $\eta = (1-\exp(-1/\lambda))^{-1}$, it can be proved that $L_{srgp}(u)$ tends to $L_{sgp}(u)$ when $\lambda \to \infty$. Moreover, $L_{srgp}(u)$ tends to $L_{gp}(u)$ when $\lambda \to \infty$ and $\mu \to 0^+$.*

Proof. For an exponential function, it can be Taylor expanded into the following form,

$$\exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots$$

thus we get the following Taylor series of $L_{srgp}(u)$:

$$L_{srgp}(u) = \eta\sum_{k=1}^{+\infty}\frac{(-1)^{k+1}(\lambda)^{-k}L_{sgp}^k(u)}{k!},$$

where $L_{sgp}(u)$ is smooth generalized pinball loss as seen in [28]. When $\eta = (1-\exp(-1/\lambda))^{-1}$ and $\lambda \to +\infty$, by L'Hospital's rule, it can be known that

$$\lim_{\lambda\to+\infty}\eta\lambda^{-k} = \lim_{\lambda\to+\infty}\frac{\left(\frac{1}{\lambda}\right)^k}{1-\exp\left(-\frac{1}{\lambda}\right)} = \begin{cases} 1, & k=1 \\ 0, & k>1 \end{cases}$$

and thus we obtain

$$\lim_{\lambda \to +\infty} L_{srgp} = L_{sgp}.$$

From Lemma 1 in [60], we get $\lim_{\mu \to 0^+} L_{sgp} = L_{gp}$. Hence

$$\lim_{\mu \to 0^+} (\lim_{\lambda \to +\infty} L_{srgp}) = L_{gp}.$$

This means that smooth generalized pinball loss $L_{sgp}(u)$ can be seen as a special case of $L_{srgp}(u)$ with $\lambda \to +\infty$. Moreover, L_{srgp} encompasses and extends existing loss functions. □

Through the incorporation of the proposed L_{srgp} into SVM, we introduce a novel robust SVM formulation that utilizes the L_{srgp} loss. This formulation is presented as follows:

$$\min_{\boldsymbol{\omega}} \ \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m} L_{srgp}(u_i), \tag{3.2.2}$$

where $\boldsymbol{\omega} = [\mathbf{w}^\top, b]^\top \in \mathbb{R}^{n+1}, u_i = 1 - y_i(\boldsymbol{\omega}^\top[\mathbf{x}_i^\top, 1]^\top)$. Since the L_{srgp}-loss is continuous and differentiable, (3.2.2) can employ the BFGS method, a well-established second-order optimization technique known for its effectiveness in solving differentiable optimization problems. In many cases, BFGS outperforms widely used first-order optimization methods. However, the L_{srgp}-loss is also nonconvex, meaning that (3.2.2) involves nonconvex minimization. Therefore, we will use modified versions of the BFGS method for solving (3.2.2), specifically the modified BFGS method (M-BFGS) as described in [67]. In the next section, we will outline the steps to use M-BFGS to solve the problem in (3.2.2) and perform convergence analysis.

3.3 Modified BFGS method for solving SRGP-SVM

This section describes the M-BFGS method we used to solve our problem. Again, we consider the non-convex differentiable problems (3.2.2):

$$\min_{\boldsymbol{\omega}} F(\boldsymbol{\omega}) := \ \frac{1}{2}\|\boldsymbol{\omega}\|^2 + C\sum_{i=1}^{m} L_{srgp}(u_i), \tag{3.3.1}$$

where $\boldsymbol{\omega} = [\mathbf{w}^\top, b]^\top \in \mathbb{R}^{n+1}$, $u_i = 1 - y_i(\boldsymbol{\omega}^\top[\mathbf{x}_i^\top, 1]^\top)$. The M-BFGS method solving SRGP-SVM can be described as follows:

$$\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k - \alpha_k(\mathbf{B}_k^{-1})\nabla F(\boldsymbol{\omega}_k),$$

we compute the $\nabla F(\boldsymbol{\omega}_k)$, that is,

$$\nabla F(\boldsymbol{\omega}_k) = \boldsymbol{\omega}_k - C\sum_{i=1}^{m} \nabla L_{srgp}(u_i^k) y_i [\mathbf{x}_i^\top, 1]^\top, \tag{3.3.2}$$

where $u_i^k = 1 - y_i(\boldsymbol{\omega}_k^\top[\mathbf{x}_i^\top, 1]^\top)$,

$$\nabla L_{srgp}(u_i^k) = \begin{cases} \frac{\tau_1\eta}{\lambda}\exp(-\frac{\tau_1\left(u_i^k - \frac{\epsilon_1}{\tau_1}\right) - \frac{\tau_1^2}{2}\mu}{\lambda}), & \frac{\epsilon_1}{\tau_1} + \tau_1\mu \le u_i^k, \\ \frac{\eta}{\lambda\mu}\exp(-\frac{\frac{1}{2\mu}\left(u_i^k - \frac{\epsilon_1}{\tau_1}\right)^2}{\lambda}), & \frac{\epsilon_1}{\tau_1} < u_i^k < \frac{\epsilon_1}{\tau_1} + \tau_1\mu, \\ \hat{0}, & \frac{-\epsilon_2}{\tau_2} \le u_i^k \le \frac{\epsilon_1}{\tau_1}, \\ \frac{\eta}{\lambda\mu}\exp(-\frac{\frac{1}{2\mu}\left(u_i^k + \frac{\epsilon_2}{\tau_2}\right)^2}{\lambda}), & \frac{-\epsilon_2}{\tau_2} - \tau_2\mu < u_i^k < \frac{-\epsilon_2}{\tau_2}, \\ -\frac{\tau_2\eta}{\lambda}\exp(-\frac{-\tau_2\left(u_i^k + \frac{\epsilon_2}{\tau_2}\right) - \frac{\tau_2^2}{2}\mu}{\lambda}), & u_i^k \le \frac{-\epsilon_2}{\tau_2} - \tau_2\mu, \end{cases}$$

and $\alpha_k > 0$ is step size satisfying the Armijo's-type line search condition. This condition determines α_k as largest value in the set $\{d^i | i = 0, 1, \dots\}$ that satisfies the inequality

$$F(\boldsymbol{\omega}_k - \alpha_k(\mathbf{B}_k^{-1})\nabla F(\boldsymbol{\omega}_k)) \le F(\boldsymbol{\omega}_k) - c\alpha_k \nabla F(\boldsymbol{\omega}_k)^\top(\mathbf{B}_k^{-1})\nabla F(\boldsymbol{\omega}_k), \tag{3.3.3}$$

where c and d are constants in the interval $(0, 1)$, and $\mathbf{B}_k$ is an approximation to the Hessian matrix $\nabla^2 F(\boldsymbol{\omega}_k)$. Then, for the next iteration, the $\mathbf{B}_k$ is defined as the matrix that satisfies the secant condition as follows:

$$\mathbf{B}_{k+1} = \begin{cases} \mathbf{B}_k - \dfrac{\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^\top\mathbf{B}_k}{\mathbf{s}_k^\top\mathbf{B}_k\mathbf{s}_k} + \dfrac{\mathbf{y}_k\mathbf{y}_k^\top}{\mathbf{y}_k^\top\mathbf{s}_k}, & \dfrac{\mathbf{y}_k^\top\mathbf{s}_k}{\|\mathbf{s}_k\|^2} \ge \sigma_1\|\nabla F(\mathbf{w}_k)\|^{\sigma_2}, \\ \mathbf{B}_k, & \text{otherwise}, \end{cases} \tag{3.3.4}$$

where σ_1 and σ_2 are positive constants and

$$\mathbf{s}_k = \boldsymbol{\omega}_{k+1} - \boldsymbol{\omega}_k,$$
$$\mathbf{y}_k = \nabla F(\boldsymbol{\omega}_{k+1}) - \nabla F(\boldsymbol{\omega}_k).$$

Clearly, if $\dfrac{\mathbf{y}_k^\top\mathbf{s}_k}{\|\mathbf{s}_k\|^2} \ge \sigma_1\|\nabla F(\boldsymbol{\omega}_k)\|^{\sigma_2}$ holds for all k, then

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^\top\mathbf{B}_k}{\mathbf{s}_k^\top\mathbf{B}_k\mathbf{s}_k} + \frac{\mathbf{y}_k\mathbf{y}_k^\top}{\mathbf{y}_k^\top\mathbf{s}_k}, \tag{3.3.5}$$

which implies that $\mathbf{B}_{k+1}$ in the M-BFGS method becomes equivalent to $\mathbf{B}_{k+1}$ in the BFGS method. From the update of (3.3.4), we can see that if we start from $\mathbf{B}_0$ as a positive definite matrix, we will find that $\mathbf{B}_k$ is a positive definite matrix for all k. This property is crucial because it ensures that $-\alpha_k(\mathbf{B}_k^{-1})\nabla F(\boldsymbol{\omega}_k)$ serves as a descent direction, leading to a decrease in the objective function values for all k, i.e.,

$$F(\boldsymbol{\omega}_{k+1}) \leq F(\boldsymbol{\omega}_k), \quad \forall k. \tag{3.3.6}$$

The proposed M-BFGS for solving SRGP-SVM is summarized in Algorithm 3.

Algorithm 3 M-BFGS-SRGP-SVM

Require: Choose an initial point $\boldsymbol{\omega}_0$, an initial symmetric and positive definite matrix $\mathbf{B}_0$, constants $\sigma_1, \sigma_2 > 0$, number of iteration $K > 0$ and tolerance $t > 0$.

1: **while** $k < K$ and $\|\nabla F(\boldsymbol{\omega}_k)\| > t$ **do**
2: Determine a stepsize $\alpha_k > 0$ by (3.3.3).
3: Compute gradient $\nabla F(\boldsymbol{\omega}_k)$ by (3.3.2).
4: Update the next iterate be $\boldsymbol{\omega}_{k+1} = \boldsymbol{\omega}_k - \alpha_k(\mathbf{B}_k^{-1})\nabla F(\boldsymbol{\omega}_k)$.
5: Update Hessian approximation matrix $\mathbf{B}_{k+1}$ by (3.3.4)
6: **end while**

3.3.1 Convergence analysis

In this section, we prove the convergence of Algorithm 3 under the following assumption, which we assume throughout this section. To prove the convergence of the proposed model, we follow the approach of [67] and make the following assumption.

Assumption 3.3.1. The level set

$$\Omega = \{\boldsymbol{\omega} \in \mathbb{R}^n \mid F(\boldsymbol{\omega}) \leq F(\boldsymbol{\omega}_0)\}$$

is bounded. From (3.3), it is clear that the sequence $\{\boldsymbol{\omega}_k\}$ generated by Algorithm 3 is contained in Ω, which means it satisfies Assumption 4.1.1.

Assumption 3.3.2. The function F is continuously differentiable on Ω, and there exists a constant $L > 0$ such that

$$\|\nabla F(\mathbf{u}) - \nabla F(\mathbf{v})\| \leq L\|\mathbf{u} - \mathbf{v}\| \quad \forall \mathbf{u}, \mathbf{v} \in \Omega. \tag{3.3.7}$$

In the following proposition, we will demonstrate that the gradient ∇F of the function F is Lipschitz continuous.

Proposition 3.3.3. *The gradient ∇F of the function F is Lipschitz continuous with constant $L_F = 1 + CG\sum_{i=1}^{m} y_i^2(1 + \|x_i\|^2)$, where $G = \max\{\frac{\tau_1^2\eta}{\lambda^2}, \frac{\tau_2^2\eta}{\lambda^2}\}$.*

Proof. From (3.2.1), it is easy to verify that it is Lipschitz continuous, namely

$$|\nabla L_{srgp}(a) - \nabla L_{srgp}(b)| \leq G|a - b|, \text{ for any } a, b,$$

where $G = \max\{\frac{\tau_1^2\eta}{\lambda^2}, \frac{\tau_2^2\eta}{\lambda^2}\}$. Let $p_i = y_i[\mathbf{x}_i^\top, 1]^\top$, we get $\nabla F(\mathbf{u}) = \mathbf{u} - C\sum_{i=1}^{m} \nabla L_{srgp}(1 - (\mathbf{u}^\top p_i))p_i$. For any $\mathbf{u}, \mathbf{v}$, we have

$$\begin{aligned}
&\|\nabla F(\mathbf{u}) - \nabla F(\mathbf{v})\| \\
&= \|\mathbf{u} - C\sum_{i=1}^{m} \nabla L_{srgp}(1 - (\mathbf{u}^\top p_i))p_i - \mathbf{v} - C\sum_{i=1}^{m} \nabla L_{srgp}(1 - (\mathbf{v}^\top p_i))p_i\| \\
&\leq \|\mathbf{u} - \mathbf{v}\| + C\sum_{i=1}^{m} \|\nabla L_{srgp}(1 - (\mathbf{u}^\top p_i))p_i - \nabla L_{srgp}(1 - (\mathbf{v}^\top p_i))p_i\| \\
&\leq \|\mathbf{u} - \mathbf{v}\| + C\sum_{i=1}^{m} |\nabla L_{srgp}(1 - (\mathbf{u}^\top p_i)) - \nabla L_{srgp}(1 - (\mathbf{v}^\top p_i))|\|p_i\| \\
&\leq \|\mathbf{u} - \mathbf{v}\| + CG\sum_{i=1}^{m} |\mathbf{u}^\top p_i - \mathbf{v}^\top p_i|\|p_i\| \\
&\leq \|\mathbf{u} - \mathbf{v}\| + CG\sum_{i=1}^{m} \|\mathbf{u} - \mathbf{v}\|\|p_i\|^2 \\
&\leq (1 + CG\sum_{i=1}^{m} \|p_i\|^2)\|\mathbf{u} - \mathbf{v}\|.
\end{aligned}$$

Therefore, we have shown that $\|\nabla F(\mathbf{u}) - \nabla F(\mathbf{v})\| \leq L_F\|\mathbf{u} - \mathbf{v}\|$, where $L_F = 1 + CG\sum_{i=1}^{m} y_i^2(1 + \|\mathbf{x}_i\|^2)$, which proves that ∇F is Lipschitz continuous with constant L_F. □

From Proposition 3.3.3, it follows that F satisfies Assumption 4.1.2. We will now demonstrate that the convergence of Algorithm 3 for solving SRGP-SVM is theoretically guaranteed, using an approach similar to [67]. To prove the convergence results, we will begin quote the following useful Lemma 3.2 in [67].

Lemma 3.3.4. *[67] Let $\boldsymbol{B}_k$ be updated by the BFGS formula (3.3.5). Suppose $\boldsymbol{B}_0$ is symmetric and positive definite and there are positive constants $m \leq M$ such that for all $k \geq 0$, $\boldsymbol{y}_k$ and $\boldsymbol{s}_k$ satisfy*

$$\frac{\boldsymbol{y}_k^\top \boldsymbol{s}_k}{\|\boldsymbol{s}_k\|^2} \geq m, \quad \frac{\|\boldsymbol{y}_k\|^2}{\boldsymbol{y}_k^T \boldsymbol{s}_k} \leq M.$$

Then there exist constants $\beta_1, \beta_2, \beta_3 > 0$ such that, for any positive integer t, the relations

$$\|\boldsymbol{B}_k \boldsymbol{s}_k\| \leq \beta_1 \|\boldsymbol{s}_k\|, \quad \beta_2 \|\boldsymbol{s}_k\|^2 \leq \boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k \leq \beta_3 \|\boldsymbol{s}_k\|^2 \tag{3.3.8}$$

holds for at least $\lceil t/2 \rceil$ values of $k \in \{1, \ldots, t\}$.

The results of Lemma 3.3.4 provide a crucial guarantee of the existence of positive constants $\beta_1, \beta_2, \beta_3$ such that (3.3.8) holds for infinitely many values of k. This condition is necessary for demonstrating the convergence of Algorithm 3. We can now proceed to establish the convergence of Algorithm 3.

Theorem 3.3.5. *Let $\{\boldsymbol{\omega}_k\}$ be generated by Algorithm 3. If there are positive constants $\beta_1, \beta_2, \beta_3 > 0$ such that the relations* (3.3.8) *hold for infinitely many k, then we have*

$$\liminf_{k \to \infty} \|\nabla F(\boldsymbol{\omega}_k)\| = 0. \tag{3.3.9}$$

Proof. Since Assumptions 4.1.1 and 4.1.2 are satisfied, and given the update rule in Algorithm 3, we can conclude (3.3.9) by directly applying the proof technique outlined in Theorem 3.1 of [67]. The next step is to establish that there exist infinitely many indices k that satisfy (3.3.8). For simplicity, we define the set of indices as follows:

$$K = \left\{ i \;\middle|\; \frac{\mathbf{y}_i^\top \mathbf{s}_i}{\|\mathbf{s}_i\|^2} \geq \sigma_1 \|\nabla F(\boldsymbol{\omega}_i)\|^{\sigma_2} \right\}. \tag{3.3.10}$$

First, consider the case where the set K is finite. In this scenario, the sequence $\mathbf{B}_k$ will stabilize to a constant after a finite number of iterations. Since each update $\mathbf{B}_k$ in Algorithm 3 is symmetric and positive definite, there exist constants $\beta_1, \beta_2, \beta_3 > 0$ such that (3.3.8) is satisfied for sufficiently large k. Next, consider the scenario where K is infinite. To reach a contradiction, assume that (3.3.9) does not hold,

which implies there exists a constant $\epsilon > 0$ such that $\|\nabla F(\boldsymbol{\omega}_k)\| \geq \epsilon$ for all k. By the definition of K in (3.3.10), we have

$$\frac{\mathbf{y}_k^\top \mathbf{s}_k}{\|\mathbf{s}_k\|^2} \geq \sigma_1 \epsilon^{\sigma_2},$$

for all $k \in K$. Together with (4.1.2), this implies that for any $k \in K$,

$$\frac{\|\mathbf{y}_k\|^2}{\mathbf{y}_k^T \mathbf{s}_k} \leq \frac{L^2}{\sigma_1 \epsilon^{\sigma_2}}.$$

By utilizing Lemma 3.3.4 on the subsequence $\{\mathbf{B}_k\}_{k \in K}$, we can deduce the existence of constants $\beta_1, \beta_2, \beta_3 > 0$ such that (3.3.8) is satisfied for infinitely many values of k. This completes the proof. □

According to Theorem 3.3.5, there exists a subsequence of $\{\boldsymbol{\omega}_k\}$ that converges to a stationary point ω^* of (3.3.1). Therefore, we can now establish the convergence theorem for Algorithm 3 regarding the solution of the SRGP-SVM.

3.3.2 Numerical experiments

Numerical simulations were conducted on various datasets to evaluate the effectiveness of the proposed SRGP-SVM method. As baseline methods, we selected SGP-SVM, SP-SVM, and SH-SVM. The M-BFGS method, as described in [67], was used to solve all methods. The experiments were divided into two parts. First, experiments were performed on synthetic datasets, followed by the application of the proposed method to benchmark UCI datasets. The simulations were executed on a MacBook Air with an Apple M1 chip, featuring an 8-core CPU (3.2GHz) and 8 GB of memory. Five-fold cross-validation was employed for each dataset, involving random partitioning of the data into five subsets, with one subset reserved as the testing set. It is important to note that the performance of the proposed method is closely linked to the selection of its parameters. To assess the effectiveness of the method, the study uses ACC, MCC, and F1-score. Additionally, we assessed the average computation time for each method, enabling a comprehensive comparison of their computational efficiency. This analysis provides insights into the trade-offs between execution speed and performance.

The Python code used in this work requires setting up the necessary modules and is based on the BaseEstimator class, which includes self in function inputs. If you intend to use the code, we recommend reviewing it. The implementation was carried out using the following environment: pandas 1.3.2, numpy 1.20.3, matplotlib 3.5.3, scikit-learn 0.24.2, scipy 1.6.2, and cvxopt 1.3.0. The following Python code implements the modified BFGS method for solving SRGP-SVM in this experiment:

```
class M_BFGS_SRGPIN_SVM(BaseEstimator):
    def __init__(self, tau1=0.9, tau2 =0.9, epsilon1 = 0.1,
        ↪ epsilon2 =0.1, lambd = 0.1, eta = 100, mu = 0.1,C=1,
        ↪ max_iteration = 100):
        self.C = C
        self.tau1 = tau1
        self.tau2 = tau2
        self.epsilon1 = epsilon1
        self.epsilon2 = epsilon2
        self.lambd = lambd
        self.eta =eta
        self.mu = mu
        self.max_iteration = max_iteration

    def srgp_loss(self, u):
        if u >= (self.epsilon1 / self.tau1) + self.tau1 * self.mu:
            return self.eta * (1 - np.exp(- (self.tau1 * (u -
                ↪ self.epsilon1 / self.tau1) - ((self.tau1 ** 2)*
                ↪ self.mu) / (2 )) / self.lambd))
        elif self.epsilon1 / self.tau1 <= u <= (self.epsilon1 /
            ↪ self.tau1) + self.tau1 * self.mu:
            return self.eta * (1 - np.exp(- (1 / (2 * self.mu) *
                ↪ (u - self.epsilon1 / self.tau1) ** 2) /
                ↪ self.lambd))
        elif -self.epsilon2 / self.tau2 <= u <= self.epsilon1 /
            ↪ self.tau1:
            return 0
        elif -(self.epsilon2 / self.tau2) - self.tau2 * self.mu <=
            ↪ u <= -self.epsilon2 / self.tau2:
            return self.eta * (1 - np.exp(- (1 / (2 * self.mu) *
                ↪ (u + self.epsilon2 / self.tau2) ** 2) /
                ↪ self.lambd))
```

```
22          elif u <= -(self.epsilon2 / self.tau2) - self.tau2 *
                ↪ self.mu:
23              return self.eta * (1 - np.exp(- (-self.tau2 * (u +
                    ↪ self.epsilon2 / self.tau2) - ((self.tau2 ** 2) *
                    ↪ self.mu) / (2)) / self.lambd))
24          else:
25              return 0
26
27      def F(self, omega, X, y):
28          regularization_term = 0.5 * (np.linalg.norm(omega) ** 2)
29          srgp_loss_term = 0
30          for i in range(len(y)):
31              u = 1 - y[i] * (np.dot(omega, X[i]))
32              srgp_loss_term += self.C * self.srgp_loss(u)
33          return regularization_term + srgp_loss_term
34      def nabla_L_srgp(self, u):
35          if  self.epsilon1 / self.tau1 + self.tau1 * self.mu <= u:
36              return (self.tau1 * self.eta / self.lambd) *
                    ↪ np.exp(-((self.tau1 * (u - self.epsilon1 /
                    ↪ self.tau1) - self.tau1**2 / (2 * self.mu)) /
                    ↪ self.lambd))
37          elif self.epsilon1 / self.tau1 <= u <= self.epsilon1 /
                ↪ self.tau1 + self.tau1 * self.mu:
38              return (self.eta / (self.lambd * self.mu)) *
                    ↪ np.exp(-(1 / (2 * self.mu) * (u - self.epsilon1
                    ↪ / self.tau1)**2) / self.lambd)
39          elif -self.epsilon2 / self.tau2 <= u <= self.epsilon1 /
                ↪ self.tau1:
40              return 0
41          elif (-self.epsilon2 / self.tau2 )- self.tau2 * self.mu <=
                ↪ u <= -self.epsilon2 / self.tau2:
42              return (self.eta / (self.lambd * self.mu)) *
                    ↪ np.exp(-(1 / (2 * self.mu) * (u + self.epsilon2
                    ↪ / self.tau2)**2) / self.lambd)
43          elif u <= -self.epsilon2 / self.tau2 - self.tau2 * self.mu:
44              return -(self.tau2 * self.eta / self.lambd) *
                    ↪ np.exp(-(-self.tau2 * (u + self.epsilon2 /
                    ↪ self.tau2) - self.tau2**2 / (2 * self.mu)) /
                    ↪ self.lambd)
45          else:
```

```
46              return 0
47      def gradient(self, omega, X, y):
48          m = X.shape[0]
49          n = X.shape[1]
50          g_F = omega.copy()
51          for i in range(m):
52              u = 1 - y[i] * (np.dot(omega, X[i]) )
53              g_L_srgp = self.nabla_L_srgp(u)
54              g_F += -self.C * g_L_srgp * y[i] * X[i]
55          return m, n, g_F
56
57      def Update_BFGS(self, B, dw, dg):
58          dg_t =  dg[:, np.newaxis]
59          Bdw = np.dot(B, dw)
60          dw_t_B = np.dot(dw, B)
61          dwBdw = np.dot(np.dot(dw, B), dw
62          p = dg_t * dg
63          u = Bdw[:, np.newaxis] * dw_t_B
64          B_new = B + (p / np.dot(dg, dw)) - (u / dwBdw)
65          return p, u, B_new
66
67      def armijo_rule(self, omega, H, g, X, y):
68          lambda_k = 100
69          c=0.1
70          while True:
71              lhs = self.F(omega -  lambda_k * np.dot(H, g), X, y)
72              rhs = self.F(omega, X, y) - c * lambda_k * np.dot(g,
                    ↪ np.dot(H, g))
73              if lhs <= rhs:
74                  break
75              lambda_k *= 0.5
76          return lambda_k
77
78      def fit(self, X, y):
79          k = 0
80          j = 1
81          ones_vector = np.ones((X.shape[0], 1))
82          X = np.hstack((X, ones_vector))
83          omega = np.zeros(len(X[0]))
84          obj_M_BFGS_SRGPIN_SVM = []
```

```
        iter_M_BFGS_SRGPIN_SVM = []
        B = np.identity(len(X[0]))
        pbar = tqdm(total=self.max_iteration, desc="Training
            ↪ M_BFGS_SRGPIN_SVM ", leave=False)
        for iteration in range(self.max_iteration):
            iter_M_BFGS_SRGPIN_SVM.append(k)
            cost = self.F(omega, X, y)
            obj_M_BFGS_SRGPIN_SVM.append(cost)
            _, _, g = self.gradient(omega, X, y)
            H = np.linalg.inv(B)
            lambda_k = self.armijo_rule(omega,H, g,X, y)
            omega_new = omega - (lambda_k * (np.matmul(H, g)))
            _, _, g_new = self.gradient(omega_new, X, y)
            dw = omega_new - omega
            if np.all(omega_new == omega) or np.linalg.norm(g_new)
                ↪ < 0.0001 or (np.linalg.norm(dw)**2)==0:
                print('Stop')
                break
            dg = g_new - g
            if  (np.matmul(dg, dw) / (np.linalg.norm(dw)**2)) >=
                ↪ (np.linalg.norm(g)):
                _, _, B_new = self.Update_BFGS(B, dw, dg)
            else:
                B_new = B
            B = B_new
            omega = omega_new
            pbar.update(1)
        self.omega = omega
        self.w = omega[:-1]
        self.b = omega[-1]
        return self.w, self.b

    def predict(self, X):
        decision = np.dot(X, self.w) + self.b
        predictions = np.sign(decision)
        predictions = np.where(predictions == 0, -1, predictions)
        return predictions

    def score(self, X, y):
        y_pred = np.dot(X, self.w) + self.b
```

```
122         y_pred = np.sign(y_pred)
123         y_pred = np.where(y_pred == 0, -1, y_pred)
124         accuracy = (y_pred == y).mean()
125         return accuracy
```

3.3.3 Experiments on synthetic datasets

We evaluated our approach using a two-dimensional scenario, drawing equal samples from two Gaussian distributions: $x_i, i \in i : y_i = 1 \sim \mathcal{N}(\mu_1, \Sigma_1)$ and $x_i, i \in i : y_i = -1 \sim \mathcal{N}(\mu_2, \Sigma_2)$, where $\mu_1 = [0.5, -3]^\top$, $\mu_2 = [-0.5, 3]^\top$, and $\Sigma_1 = \Sigma_2 = [0.2\ 0; 0\ 3]$. The training data consisted of 400 samples (DATA-I) and 12,000 samples (DATA-II), evenly divided between the two classes. To introduce complexity, we added outliers, with labels chosen from $-1, 1$ and positions following a Gaussian distribution $\mathcal{N}(\mu_o, \Sigma_o)$ where $\mu_o = [0, -10]^\top$ and $\Sigma_o = [1\ \ -0.8; -0.8\ 1]$. The outlier ratio r, set to values 0.00, 0.05, 0.10, 0.15, and 0.20, controls the proportion of outliers relative to the total sample size, where $r = 0.00$ indicates no outliers.

In the M-BFGS method, we set σ_1 and σ_2 to 1, with stopping criteria $t = 10^{-4}$ and $K = 20$. For all methods, we used $C = 1$ and $\mu = 0.1$. The optimal τ for SP-SVM was selected from $\{0.5, 0.7, 0.9\}$, while for SRGP-SVM and SGP-SVM, both τ_1 and τ_2 were chosen from this same set. The best ϵ_1 and ϵ_2 values for SRGP-SVM and SGP-SVM were selected from $\{0.01, 0.05\}$. Additionally, we set $\lambda = 1$ and chose λ and η for SRGP-SVM from $\{0.5, 1, 2, 5\}$. Optimal parameters were determined via five-fold cross-validation to maximize ACC. Tables 14 and 15 display the optimal parameters for varying r values in DATA-I and DATA-II, respectively.

Table 14 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on DATA-I.

r	SH-SVM C, μ	SP-SVM τ, C, μ	SGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ C, μ	SRGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ λ, η, C, μ
0.00	1, 0.1	0.5, 1, 0.1	0.5, 0.9, 0.01, 0.01 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 2, 1, 0.1
0.05	1, 0.1	0.9, 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 2, 1, 0.1
0.10	1, 0.1	0.9, 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 0.1	0.7, 0.9, 0.05, 0.01 1, 1, 1, 0.1
0.15	1, 0.1	0.9, 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.7, 0.01, 0.01 1, 1, 1, 0.1
0.20	1, 0.1	0.7, 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.7, 0.01, 0.05 1, 0.5, 1, 0.1

Table 15 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on DATA-II.

r	SH-SVM C, μ	SP-SVM τ, C, μ	SGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ C, μ	SRGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ λ, η, C, μ
0.00	1, 0.1	0.7, 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.9, 0.01, 0.05 1, 5, 1, 0.1
0.05	1, 0.1	0.7, 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.9, 0.01, 0.05 1, 5, 1, 0.1
0.10	1, 0.1	0.7, 1, 0.1	0.7, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.9, 0.01, 0.05 1, 2, 1, 0.1
0.15	1, 0.1	0.9, 1, 0.1	0.7, 0.9, 0.05, 0.05 1, 0.1	0.5, 0.9, 0.01, 0.05 1, 2, 1, 0.1
0.20	1, 0.1	0.9, 1, 0.1	0.7, 0.9, 0.05, 0.05 1, 0.1	0.5, 0.9, 0.01, 0.05 1, 2, 1, 0.1

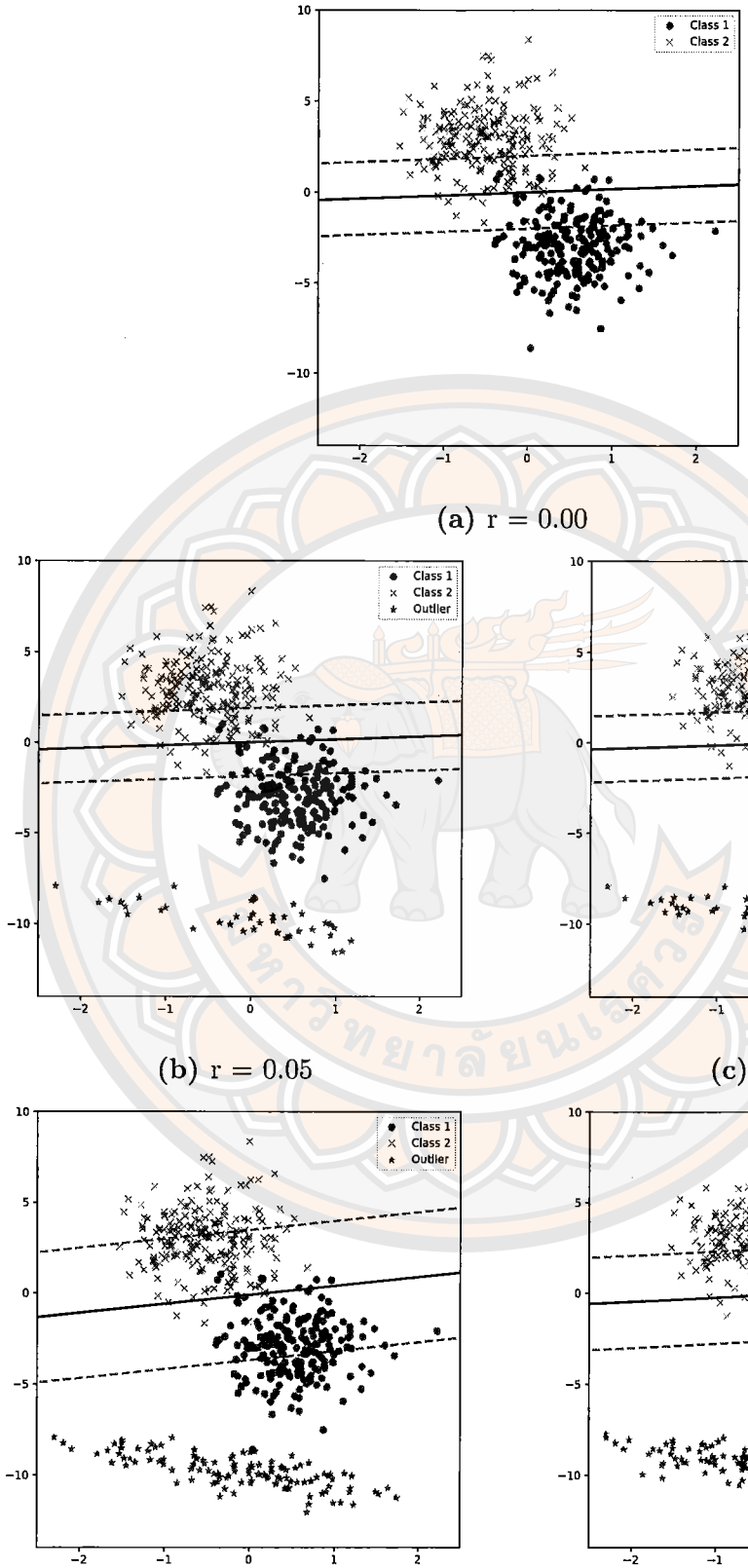
Table 16 compares SRGP-SVM, SGP-SVM, SP-SVM, and SH-SVM on two synthetic datasets, evaluating their ACC, MCC, F_1, and computation time under varying outlier levels (represented by different r values). In DATA-I, SRGP-SVM

consistently achieves the highest ACC, MCC, and F_1 across all outlier levels, demonstrating its robustness and reliability, although it requires more computation time compared to the other models. SH-SVM is the fastest, particularly with minimal outliers, but its performance metrics are slightly lower than those of SRGP-SVM and SGP-SVM. Both SGP-SVM and SP-SVM deliver comparable performance, with SP-SVM showing slightly better F1-scores under higher outlier conditions. A similar trend is observed in DATA-II, where SRGP-SVM again leads in metrics, though its computation time increases significantly as the outlier level rises. SH-SVM maintains the shortest computation times across most outlier levels, though its performance drops more noticeably as the outlier level increases. SGP-SVM and SP-SVM balance accuracy and efficiency, with SP-SVM performing particularly well under high-outlier conditions. Overall, SRGP-SVM is optimal for tasks requiring high accuracy, while SH-SVM is a practical choice when computation time is a priority.

Table 16 Comparisons of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on synthetic datasets.

Dataset	r	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time
DATA-I	0.00	SRGP-SVM	**97.68 ± 1.26**	**95.40 ± 2.47**	**97.61 ± 1.38**	0.6501
		SGP-SVM	97.42 ± 1.41	94.88 ± 2.78	97.34 ± 1.53	0.2362
		SP-SVM	97.42 ± 0.81	94.87 ± 1.53	97.30 ± 0.89	0.3095
		SH-SVM	97.42 ± 1.41	94.88 ± 2.78	97.34 ± 1.53	**0.1969**
	0.05	SRGP-SVM	**92.06 ± 3.36**	**84.48 ± 4.43**	**92.06 ± 2.09**	0.5252
		SGP-SVM	91.83 ± 2.64	84.03 ± 5.08	91.90 ± 2.25	**0.1860**
		SP-SVM	91.83 ± 2.64	84.03 ± 5.08	91.90 ± 2.25	0.2714
		SH-SVM	91.13 ± 3.32	82.46 ± 6.75	91.17 ± 2.71	0.3157
	0.10	SRGP-SVM	**89.32 ± 1.75**	**79.10 ± 3.33**	**89.75 ± 1.12**	0.8588
		SGP-SVM	89.11 ± 2.71	78.69 ± 5.60	89.56 ± 2.04	**0.2011**
		SP-SVM	88.89 ± 2.72	78.21 ± 5.67	89.36 ± 1.86	0.3277
		SH-SVM	88.47 ± 2.53	77.20 ± 5.17	88.85 ± 1.60	0.2201
	0.15	SRGP-SVM	**83.86 ± 2.12**	68.72 ± 4.53	84.27 ± 1.32	0.8494
		SGP-SVM	83.47 ± 2.87	67.99 ± 5.92	84.08 ± 1.66	**0.2036**
		SP-SVM	83.86 ± 2.59	**68.73 ± 5.34**	**84.45 ± 1.37**	0.3436
		SH-SVM	82.68 ± 2.81	66.05 ± 5.81	82.94 ± 1.96	0.3024
	0.20	SRGP-SVM	**84.13 ± 1.85**	**69.49 ± 3.11**	**84.71 ± 2.35**	1.2958
		SGP-SVM	83.94 ± 2.38	69.23 ± 5.19	84.47 ± 2.49	**0.2102**
		SP-SVM	83.39 ± 3.06	68.21 ± 6.49	83.89 ± 3.22	0.3240
		SH-SVM	82.48 ± 3.31	66.10 ± 6.77	82.87 ± 3.51	0.2964
DATA-II	0.00	SRGP-SVM	**97.71 ± 0.38**	**95.43 ± 0.76**	**97.71 ± 0.38**	34.2374
		SGP-SVM	97.41 ± 0.41	94.83 ± 0.82	97.41 ± 0.39	9.6841
		SP-SVM	97.66 ± 0.33	95.33 ± 0.65	97.66 ± 0.32	8.5258
		SH-SVM	97.66 ± 0.37	95.31 ± 0.75	97.65 ± 0.37	**6.0556**
	0.05	SRGP-SVM	**93.07 ± 0.71**	**86.48 ± 1.34**	**93.33 ± 0.63**	6.6168
		SGP-SVM	92.68 ± 0.72	85.62 ± 1.39	92.91 ± 0.65	8.1003
		SP-SVM	92.84 ± 0.69	85.97 ± 1.31	93.09 ± 0.63	8.3629
		SH-SVM	92.32 ± 0.50	84.88 ± 0.93	92.55 ± 0.45	**6.2276**
	0.10	SRGP-SVM	**88.75 ± 0.67**	**78.15 ± 1.35**	**89.38 ± 0.61**	21.3095
		SGP-SVM	87.66 ± 0.63	75.56 ± 1.21	88.08 ± 0.57	11.5787
		SP-SVM	88.69 ± 0.60	77.98 ± 1.12	89.31 ± 0.50	8.6152
		SH-SVM	88.36 ± 0.59	77.21 ± 1.07	88.93 ± 0.51	**8.6090**
	0.15	SRGP-SVM	**85.45 ± 0.80**	**71.90 ± 1.62**	**86.55 ± 0.68**	21.0877
		SGP-SVM	84.24 ± 0.79	68.81 ± 1.66	84.96 ± 0.63	12.4207
		SP-SVM	85.11 ± 0.59	70.90 ± 1.17	86.06 ± 0.46	**8.2706**
		SH-SVM	78.44 ± 1.98	58.67 ± 3.88	80.81 ± 1.45	8.4608
	0.20	SRGP-SVM	**83.14 ± 0.41**	**67.99 ± 0.94**	**84.75 ± 0.30**	12.3295
		SGP-SVM	81.24 ± 0.89	62.90 ± 1.94	82.15 ± 0.79	13.8231
		SP-SVM	82.20 ± 0.77	65.27 ± 1.69	83.45 ± 0.63	10.0628
		SH-SVM	82.21 ± 0.71	65.26 ± 1.61	83.43 ± 0.59	**8.7313**

Moreover, Figures 16 and 17 illustrate the outcomes of SRGP-SVM solved using M-BFGS with the following fixed parameters: for both DATA-I and DATA-II, $C = 1$, $\tau_1 = 0.5$, $\tau_2 = 0.9$, and $\mu = 0.1$. For DATA-I, $\epsilon_1 = 0.05$ and $\epsilon_2 = 0.01$, while for DATA-II, $\epsilon_1 = 0.01$ and $\epsilon_2 = 0.05$. The black line represents the outcomes achieved by SRGP-SVM, demonstrating that SRGP-SVM is insensitive to outliers and effectively captures the true data structure. These results suggest that SRGP-SVM is robust to outliers. Additionally, we consider η and λ from the set $\{0.1, 0.5, 1, 2, 5\}$, which demonstrates the impact of η and λ on the accuracy of SRGP-SVM for the DATA-I and DATA-II with different outliers levels, as depicted in Figure 18 and 19. We focus on the impact of two parameters, η and λ, while keeping the others fixed. Appropriately low values of both η and λ can reduce the impact of outliers and control the loss, thereby improving the accuracy of SRGP-SVM, as illustrated in Figures 18a, 18b, 18c and 18d for DATA-I and 19a, 19b, 19c, 19d and 19e for DATA-II. However, selecting excessively low values for η and λ may be problematic, as this can lead to an undue reduction in the overall loss function, as illustrated in Figure 18e. Our analysis indicates that η should not be too small in the presence of outliers, and both η and λ should be appropriately adjusted to ensure a proper fit to the data.



(a) r = 0.00

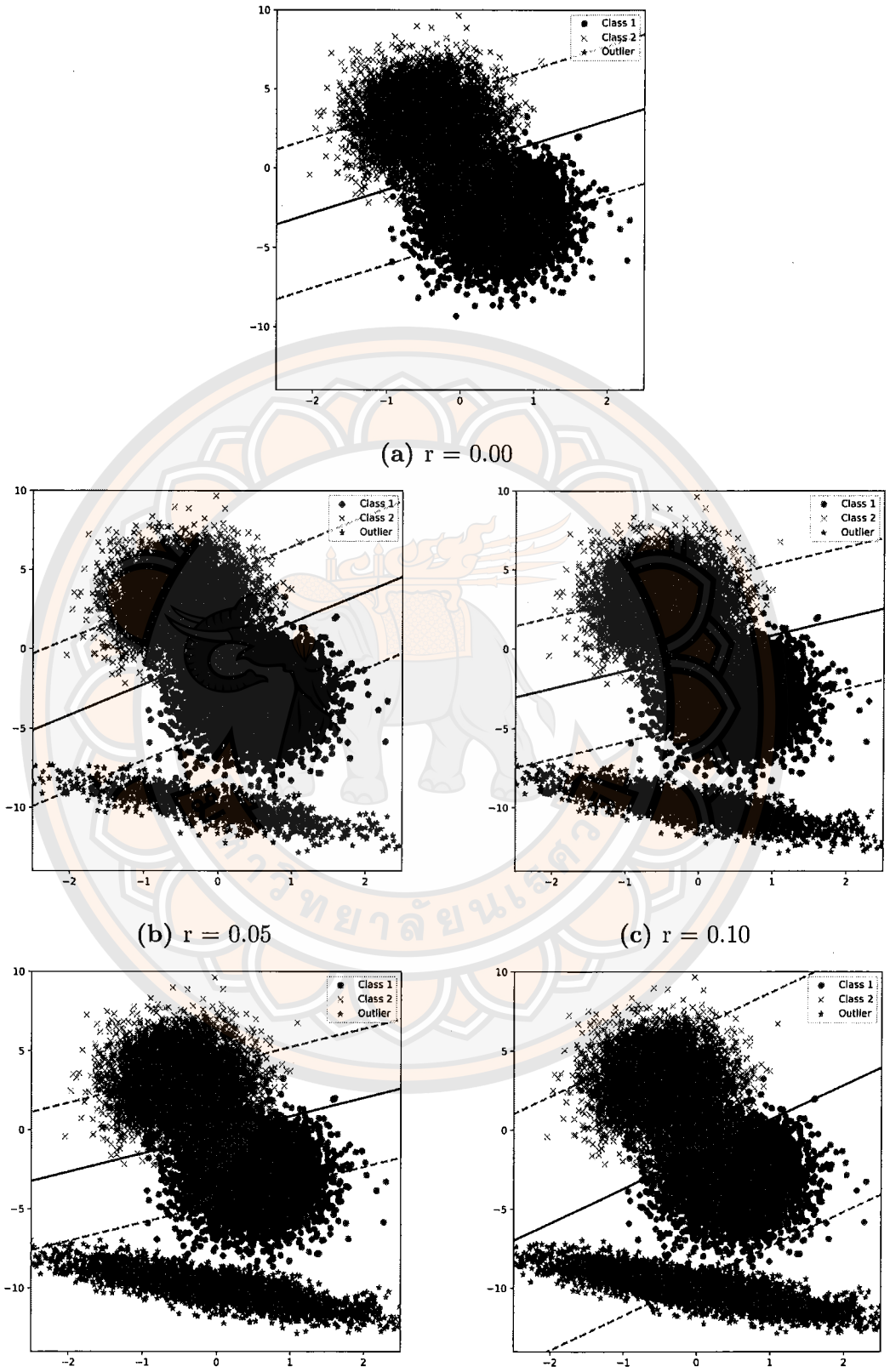
(b) r = 0.05

(c) r = 0.10

(d) r = 0.15

(e) r = 0.20

Figure 16 The classifier obtained by SRGP-SVM on DATA-I.



(a) r = 0.00

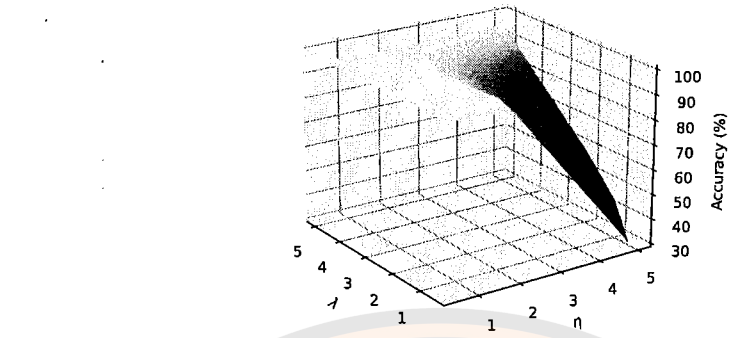
(b) r = 0.05

(c) r = 0.10

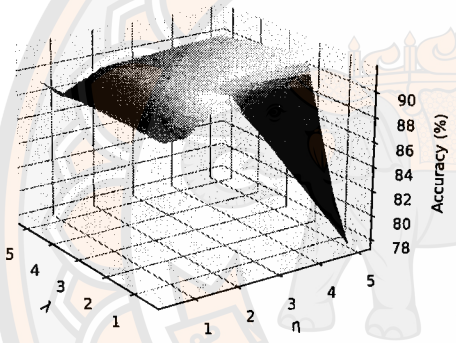
(d) r = 0.15

(e) r = 0.20

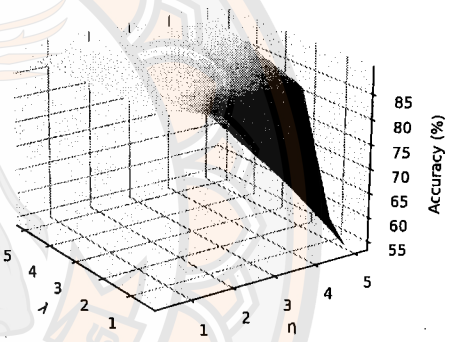
Figure 17 The classifier obtained by SRGP-SVM on DATA-II.



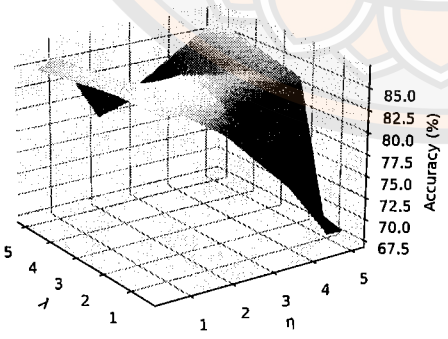
(a) r = 0.00



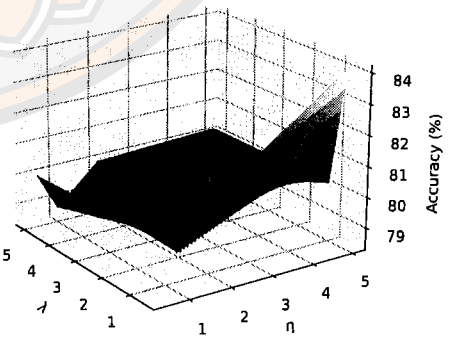
(b) r = 0.05



(c) r = 0.10

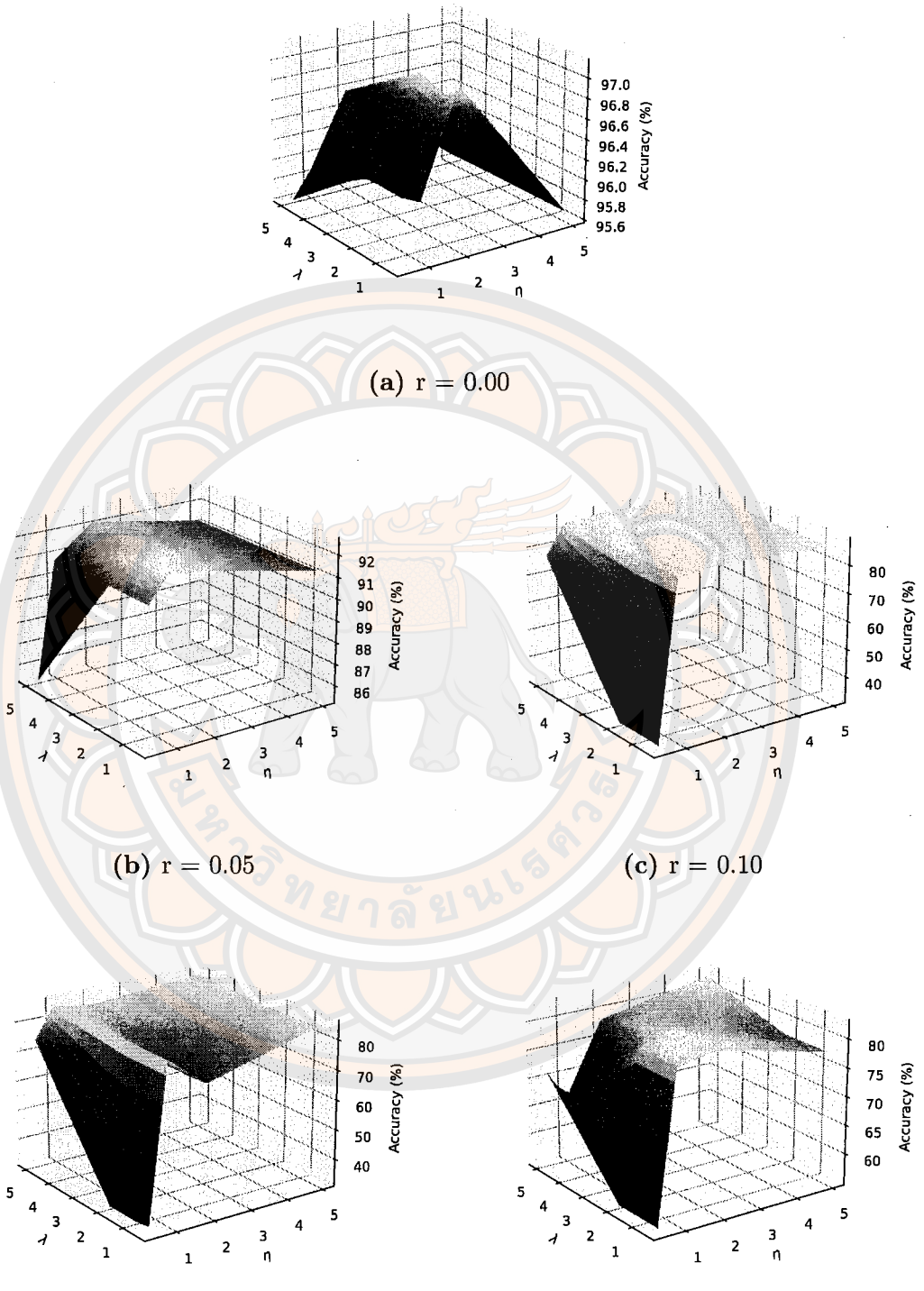


(d) r = 0.15



(e) r = 0.20

Figure 18 Variation in test accuracy of SRGP-SVM attributed to η and λ on DATA-I.



(a) r = 0.00

(b) r = 0.05

(c) r = 0.10

(d) r = 0.15

(e) r = 0.20

Figure 19 Variation in test accuracy of SRGP-SVM attributed to η and λ on DATA-II.

3.3.4 Experiments on UCI benchmark datasets

We conducted experiments on UCI benchmark datasets (Ionosphere [37], Australian [68], Diabetes [39], Appendicitis [69], Heart [70], WDBC [71], Pima [38], Rice [44], Heart Failure [42] and Auction [72]) under both noisy and noiseless settings. In the noisy setting, the training datasets are affected by zero-mean Gaussian noise with variances of 0.1 and 1.0. Additionally, we explored kernel methods by utilizing an approximate feature map. We assessed both a linear kernel and a nonlinear kernel, such as the RBF kernel.

Table 17 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data without noise using a linear kernel.

Datasets	SH-SVM C, μ	SP-SVM τ, C, μ	SGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ C, μ	SRGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ λ, η, C, μ
Ionosphere	1, 0.1	0.9, 1, 0.1	0.9, 0.1, 0.01, 0.05 1, 0.1	0.9, 0.1, 0.01, 0.01 1, 5, 1, 0.1
Australian	1, 0.1	0.1, 1, 0.1	0.5, 0.1, 0.05, 0.01 1, 0.1	0.2, 0.1, 0.01, 0.05 0.5, 5, 1, 0.1
Diabetes	1, 0.1	0.1, 1, 0.1	0.9, 0.2, 0.01, 0.05 1, 0.1	0.9, 0.1, 0.01, 0.01 1, 5, 1, 0.1
Appendicitis	1, 0.1	0.2, 1, 0.1	0.1, 0.5, 0.05, 0.01 1, 0.1	0.1, 0.5, 0.01, 0.01 0.5, 5, 1, 0.1
Heart	1, 0.1	0.5, 1, 0.1	0.2, 0.1, 0.05, 0.01 1, 0.1	0.2, 0.5, 0.05, 0.01 0.5, 5, 1, 0.1
WDBC	1, 0.1	0.1, 1, 0.1	0.5, 0.1, 0.01, 0.01 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 5, 1, 0.1
Pima	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.9, 0.2, 0.05, 0.01 1, 5, 1, 0.1
Rice	1, 0.1	0.9, 1, 0.1	0.1, 0.1, 0.01, 0.01 1, 0.1	0.2, 0.2, 0.05, 0.05 1, 5, 1, 0.1
Heart Failure	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.01 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 0.5, 1, 0.1
Auction	1, 0.1	0.9, 1, 0.1	0.9, 0.9, 0.05, 0.05 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 0.5, 1, 0.1

In this investigation, we conducted parameter optimization aiming to maximize accuracy through five-fold cross-validation on each dataset. Upon parameter

Table 18 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise ($\sigma^2 = 0.1$) using a linear kernel.

Datasets	SH-SVM C, μ	SP-SVM τ, C, μ	SGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ C, μ	SRGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ λ, η, C, μ
Ionosphere	1, 0.1	0.1, 1, 0.1	0.9, 0.2, 0.01, 0.05 1, 0.1	0.9, 0.2, 0.05, 0.05 1, 0.5, 1, 0.1
Australian	1, 0.1	0.1, 1, 0.1	0.5, 0.1, 0.05, 0.01 1, 0.1	0.1, 0.1, 0.01, 0.05 0.5, 5, 1, 0.1
Diabetes	1, 0.1	0.2, 1, 0.1	0.5, 0.5, 0.01, 0.05 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 5, 1, 0.1
Appendicitis	1, 0.1	0.5, 1, 0.1	0.1, 0.1, 0.05, 0.01 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 5, 1, 0.1
Heart	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.1, 0.05, 0.05 1, 5, 1, 0.1
WDBC	1, 0.1	0.1, 1, 0.1	0.5, 0.1, 0.05, 0.01 1, 0.1	0.5, 0.2, 0.01, 0.01 1, 0.5, 1, 0.1
Pima	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.2, 0.05, 0.05 1, 1, 1, 0.1
Rice	1, 0.1	0.9, 1, 0.1	0.1, 0.1, 0.01, 0.01 1, 0.1	0.1, 0.9, 0.01, 0.01 1, 1, 1, 0.1
Heart Failure	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.01 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 0.5, 1, 0.1
Auction	1, 0.1	0.9, 1, 0.1	0.9, 0.9, 0.05, 0.05 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 0.5, 1, 0.1

selection, the optimal values were employed to scrutinize the experimental results. The precise optimal parameter values are outlined in the corresponding experimental sections. We set $\sigma_1 = \sigma_2 = 1$ and selected $t = 10^{-4}$ and $K = 20$ as the iteration parameters in the M-BFGS method. Since the parameters C and μ, which control the total loss of all data and the smoothness of the function, respectively, are common to all methods and do not directly affect the reduction of the outlier effect, we will fix $C = 1$ and $\mu = 0.1$ to facilitate the analysis of the other parameters. The optimal τ in SP-SVM and τ_1 and τ_2 in SRGP-SVM and SGP-SVM are chosen from the set $\{0.1, 0.2, 0.5, 0.9\}$. The optimal ϵ_1 and ϵ_2 in SRGP-SVM and SGP-SVM are chosen from the set $\{0.01, 0.05\}$. Additionally, we select parameters λ and η in

Table 19 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise ($\sigma^2 = 1.0$) using a linear kernel.

Datasets	SH-SVM C, μ	SP-SVM τ, C, μ	SGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ C, μ	SRGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ λ, η, C, μ
Ionosphere	1, 0.1	0.5, 1, 0.1	0.2, 0.2, 0.05, 0.01 1, 0.1	0.5, 0.2, 0.05, 0.01 1, 0.5, 1, 0.1
Australian	1, 0.1	0.1, 1, 0.1	0.5, 0.1, 0.05, 0.01 1, 0.1	0.5, 0.1, 0.05, 0.01 1, 1, 1, 0.1
Diabetes	1, 0.1	0.1, 1, 0.1	0.2, 0.1, 0.01, 0.01 1, 0.1	0.5, 0.9, 0.01, 0.01 1, 2, 1, 0.1
Appendicitis	1, 0.1	0.2, 1, 0.1	0.1, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 2, 1, 0.1
Heart	1, 0.1	0.2, 1, 0.1	0.5, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.1, 0.05, 0.05 1, 5, 1, 0.1
WDBC	1, 0.1	0.9, 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 0.1	0.5, 0.9, 0.01, 0.01 0.5, 0.5, 1, 0.1
Pima	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.9, 0.2, 0.01, 0.05 0.5, 0.5, 1, 0.1
Rice	1, 0.1	0.9, 1, 0.1	0.1, 0.1, 0.01, 0.01 1, 0.1	0.1, 0.9, 0.01, 0.01 1, 1, 1, 0.1
Heart Failure	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.01 1, 0.1	0.2, 0.9, 0.01, 0.01 1, 0.5, 1, 0.1
Auction	1, 0.1	0.9, 1, 0.1	0.5, 0.9, 0.01, 0.05 1, 0.1	0.9, 0.9, 0.01, 0.01 1, 0.5, 1, 0.1

SRGP-SVM from the set $\{0.5, 1\}$ and $\{0.1, 0.5, 1, 2, 5\}$, respectively. The optimal parameters are determined through five-fold cross-validation to maximize the ACC. The optimal parameters for the linear kernel in both noiseless and noisy cases are presented in Tables 17, 18, and 19. The optimal parameters for the RBF kernel in noiseless and noisy cases are shown in Tables 23, 24, and 25. Initially, we consider the noiseless case for the linear kernel in Table 20. It can be observed that:

- SRGP-SVM generally provides the highest accuracy across multiple datasets, particularly excelling in the Ionosphere, Australian, Heart, and Auction datasets.
- SGP-SVM stands out with strong MCC performance in several datasets, in-

but generally lags behind SH-SVM in speed.

- SH-SVM consistently has the fastest computation times across all datasets and also achieves the best performance in terms of accuracy and other metrics on certain datasets, such as Appendicitis, WDBC, Rice, and Heart Failure.

Overall, in the noiseless case using a linear kernel, SH-SVM consistently shows the fastest computation times, while SRGP-SVM generally provides strong results in terms of accuracy. However, SGP-SVM and SP-SVM also deliver competitive performance across different criteria depending on the dataset.

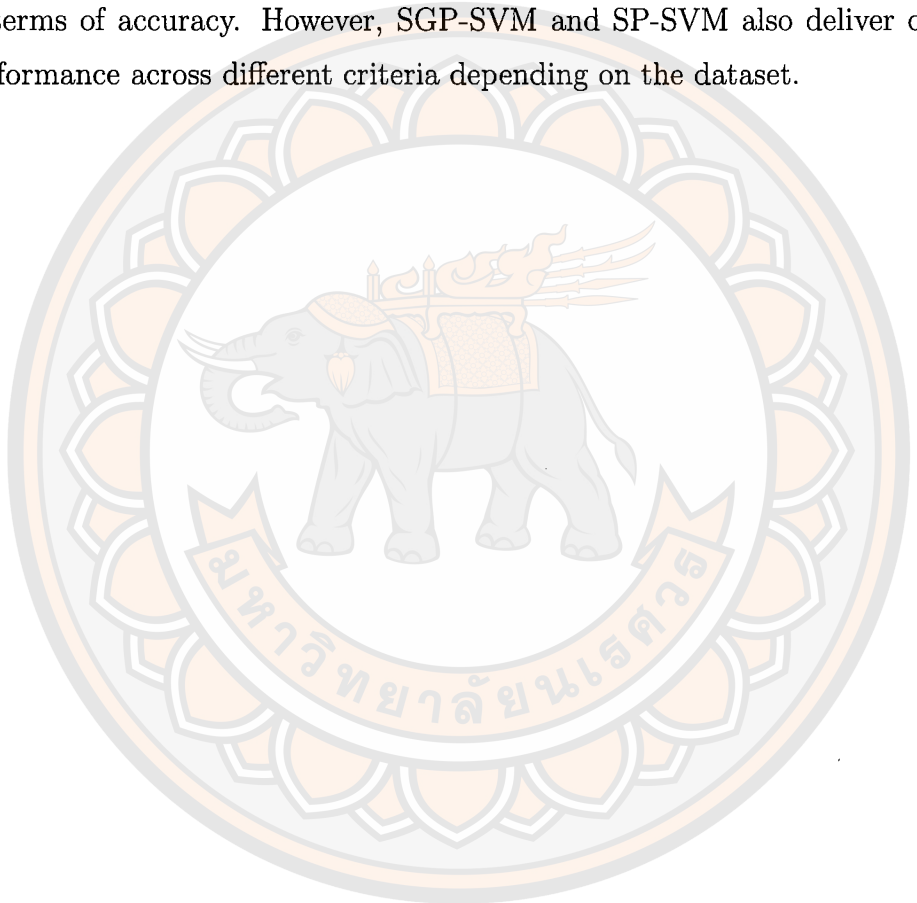


Table 20 Comparisons of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data without noise using a linear kernel.

Dataset	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time
Ionosphere	SRGP-SVM	**86.09 ± 1.80**	71.75 ± 3.20	84.25 ± 1.80	2.4663
(350 × 34)	SGP-SVM	85.65 ± 1.16	**72.18 ± 2.62**	**85.06 ± 2.02**	0.4454
	SP-SVM	85.51 ± 1.30	71.82 ± 3.03	84.85 ± 2.41	0.5944
	SH-SVM	85.51 ± 0.92	71.78 ± 2.01	84.86 ± 1.79	**0.4427**
Australian	SRGP-SVM	**86.23 ± 1.89**	72.05 ± 3.42	84.38 ± 1.96	3.4977
(690 × 14)	SGP-SVM	85.51 ± 1.30	71.82 ± 3.03	84.85 ± 2.41	0.7914
	SP-SVM	85.65 ± 1.25	**72.08 ± 2.94**	84.97 ± 2.43	0.7354
	SH-SVM	85.65 ± 0.85	72.04 ± 1.88	**84.98 ± 1.82**	**0.5671**
Diabetes	SRGP-SVM	75.78 ± 2.89	46.44 ± 5.13	**64.58 ± 2.44**	1.9816
(1151 × 19)	SGP-SVM	**77.09 ± 2.25**	**47.82 ± 2.86**	62.93 ± 2.65	0.9876
	SP-SVM	76.83 ± 2.32	47.31 ± 2.73	62.53 ± 2.34	0.8387
	SH-SVM	76.96 ± 2.63	47.77 ± 3.99	63.32 ± 2.99	**0.7367**
Appendicitis	SRGP-SVM	85.93 ± 4.82	48.06 ± 25.40	56.03 ± 28.56	0.2454
(106 × 7)	SGP-SVM	85.93 ± 6.44	45.31 ± 27.70	49.46 ± 28.13	0.0860
	SP-SVM	84.03 ± 10.97	33.76 ± 31.84	37.46 ± 33.38	**0.0844**
	SH-SVM	**88.79 ± 6.75**	**52.35 ± 29.85**	**57.60 ± 30.49**	0.1021
Heart	SRGP-SVM	**84.44 ± 5.69**	**68.35 ± 11.52**	**86.14 ± 4.99**	0.8933
(303 × 14)	SGP-SVM	84.07 ± 4.48	67.75 ± 8.97	85.73 ± 4.39	0.2815
	SP-SVM	82.59 ± 4.77	64.61 ± 9.73	84.54 ± 4.07	0.2523
	SH-SVM	83.33 ± 5.62	66.07 ± 11.47	85.19 ± 4.77	**0.2064**
WDBC	SRGP-SVM	96.66 ± 1.03	92.92 ± 2.12	95.55 ± 1.24	3.3468
(569 × 30)	SGP-SVM	96.30 ± 1.31	92.13 ± 2.67	94.95 ± 1.57	0.5918
	SP-SVM	96.66 ± 0.66	92.93 ± 1.34	95.42 ± 0.78	0.4920
	SH-SVM	**97.01 ± 1.19**	**93.66 ± 2.52**	**96.00 ± 1.55**	**0.3586**
Pima	SRGP-SVM	75.26 ± 2.55	45.19 ± 6.76	**64.15 ± 6.12**	5.2275
(768 × 8)	SGP-SVM	76.56 ± 2.64	46.44 ± 4.29	62.24 ± 2.67	0.6091
	SP-SVM	**77.60 ± 2.96**	**48.83 ± 5.31**	63.63 ± 3.44	0.6982
	SH-SVM	76.95 ± 2.57	47.29 ± 4.50	62.75 ± 2.88	**0.5229**
Rice	SRGP-SVM	91.47 ± 0.99	82.66 ± 2.13	92.46 ± 0.75	4.4473
(3810 × 7)	SGP-SVM	92.49 ± 0.94	84.72 ± 2.02	93.39 ± 0.76	4.4368
	SP-SVM	92.18 ± 1.07	84.10 ± 2.19	93.37 ± 0.88	**3.2084**
	SH-SVM	**92.97 ± 0.68**	**85.65 ± 1.46**	**93.85 ± 0.59**	3.3586
Heart Failure	SRGP-SVM	**82.93 ± 3.12**	**61.35 ± 6.00**	**72.34 ± 5.66**	2.0357
(299 × 12)	SGP-SVM	53.19 ± 6.82	6.53 ± 15.98	40.93 ± 12.16	0.3496
	SP-SVM	54.53 ± 11.95	10.30 ± 21.76	44.28 ± 13.37	0.2673
	SH-SVM	60.53 ± 12.39	17.36 ± 28.34	47.27 ± 20.13	**0.2380**
Auction	SRGP-SVM	**87.03 ± 0.63**	10.23 ± 1.99	5.67 ± 2.70	9.7506
(2043 × 7)	SGP-SVM	70.09 ± 2.82	11.05 ± 4.87	25.40 ± 4.37	9.7506
	SP-SVM	67.01 ± 6.58	10.33 ± 6.81	**55.74 ± 12.44**	2.2623
	SH-SVM	55.74 ± 12.44	**15.69 ± 8.11**	28.02 ± 4.37	**1.4443**

Table 21 Comparisons of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise using a linear kernel.

Dataset	σ^2	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time
Ionosphere	0.1	SRGP-SVM	**85.36 ± 1.80**	70.28 ± 3.54	83.39 ± 2.67	2.2977
(350 × 34)		SGP-SVM	84.78 ± 1.52	70.16 ± 3.36	83.97 ± 2.64	0.4149
		SP-SVM	84.93 ± 1.48	**70.48 ± 3.30**	**84.13 ± 2.71**	0.3691
		SH-SVM	85.07 ± 1.42	70.44 ± 3.10	84.04 ± 2.64	**0.3122**
	1.0	SRGP-SVM	76.96 ± 3.65	53.66 ± 7.19	74.68 ± 4.24	0.7505
		SGP-SVM	**78.99** ± 3.27	**57.45 ± 6.89**	**76.23 ± 4.50**	0.3234
		SP-SVM	78.70 ± 2.23	56.90 ± 4.77	76.01 ± 3.38	0.2861
		SH-SVM	77.54 ± 4.27	54.56 ± 9.08	74.60 ± 6.04	**0.2308**
Australian	0.1	SRGP-SVM	**85.36 ± 1.91**	70.17 ± 3.87	83.32 ± 2.48	4.6384
(690 × 14)		SGP-SVM	84.93 ± 1.48	**70.48 ± 3.27**	**84.14 ± 2.68**	1.0206
		SP-SVM	84.93 ± 1.48	70.48 ± 3.30	84.13 ± 2.71	0.7063
		SH-SVM	85.07 ± 1.42	70.44 ± 3.10	84.04 ± 2.64	**0.4910**
	1.0	SRGP-SVM	**79.13 ± 3.56**	**58.14 ± 6.73**	**76.97 ± 3.12**	4.5096
		SGP-SVM	78.55 ± 4.50	56.68 ± 8.32	75.70 ± 4.45	0.8750
		SP-SVM	78.12 ± 4.36	55.98 ± 7.81	75.25 ± 4.35	0.6473
		SH-SVM	78.55 ± 4.34	56.84 ± 7.78	75.61 ± 4.40	**0.4769**
Diabetes	0.1	SRGP-SVM	**75.91 ± 1.11**	**45.14 ± 2.34**	**61.28 ± 2.59**	1.6550
(1151 × 19)		SGP-SVM	74.61 ± 0.83	41.03 ± 2.88	55.92 ± 4.18	0.8354
		SP-SVM	75.26 ± 1.69	43.60 ± 1.70	59.95 ± 2.14	0.7588
		SH-SVM	75.39 ± 1.32	43.90 ± 1.73	60.52 ± 2.32	**0.5273**
	1.0	SRGP-SVM	**72.27 ± 1.84**	**37.58 ± 4.65**	**57.26 ± 4.98**	3.6874
		SGP-SVM	71.35 ± 4.81	32.00 ± 7.76	44.04 ± 6.77	0.8475
		SP-SVM	71.35 ± 4.13	32.32 ± 5.63	46.04 ± 5.11	0.6851
		SH-SVM	71.74 ± 2.77	33.77 ± 3.93	50.25 ± 5.08	**0.5119**
Appendicitis	0.1	SRGP-SVM	85.02 ± 7.78	**47.01 ± 30.73**	**52.14 ± 30.26**	0.3341
(106 × 7)		SGP-SVM	84.03 ± 10.97	33.76 ± 31.84	37.46 ± 33.38	0.0877
		SP-SVM	80.26 ± 9.96	9.22 ± 18.44	8.00 ± 16.00	0.1242
		SH-SVM	**86.02 ± 9.88**	43.55 ± 31.61	50.39 ± 30.09	**0.0776**
	1.0	SRGP-SVM	83.07 ± 4.57	**41.82 ± 23.58**	**48.65 ± 26.36**	0.2277
		SGP-SVM	82.08 ± 8.18	29.45 ± 24.11	29.43 ± 24.64	0.0834
		SP-SVM	82.08 ± 8.18	33.61 ± 28.09	34.76 ± 28.87	0.0775
		SH-SVM	**83.98 ± 2.14**	40.52 ± 20.48	42.76 ± 23.09	**0.0672**
Heart	0.1	SRGP-SVM	**84.07 ± 5.05**	**67.85 ± 10.62**	85.48 ± 4.66	1.3154
(303 × 14)		SGP-SVM	82.22 ± 4.32	63.68 ± 8.82	84.23 ± 3.81	0.2712
		SP-SVM	83.33 ± 4.83	66.26 ± 10.17	85.34 ± 4.03	0.2860
		SH-SVM	83.70 ± 5.67	66.98 ± 11.53	**85.59 ± 5.03**	**0.2165**
	1.0	SRGP-SVM	**79.63 ± 6.09**	58.68 ± 11.83	80.89 ± 6.44	1.3355
		SGP-SVM	78.52 ± 5.32	56.18 ± 10.52	81.36 ± 5.39	0.2833
		SP-SVM	77.04 ± 6.48	53.19 ± 13.01	79.87 ± 6.59	0.2496
		SH-SVM	79.63 ± 7.50	**58.92 ± 14.84**	**81.53 ± 8.03**	**0.2369**

Table 22 Comparisons of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise using a linear kernel (continued).

Dataset	σ^2	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time
WDBC	0.1	SRGP-SVM	**96.30 ± 2.47**	**92.21 ± 5.09**	**95.16 ± 3.07**	3.1254
(569 × 30)		SGP-SVM	96.13 ± 1.54	91.74 ± 3.18	94.77 ± 1.89	0.6164
		SP-SVM	95.60 ± 1.57	90.58 ± 3.29	94.02 ± 2.04	0.4839
		SH-SVM	95.60 ± 1.93	90.63 ± 4.04	94.05 ± 2.51	**0.3724**
	1.0	SRGP-SVM	**91.56 ± 2.15**	**82.08 ± 4.53**	**88.85 ± 2.83**	2.1204
		SGP-SVM	85.23 ± 3.70	69.31 ± 7.04	75.68 ± 6.75	0.7091
		SP-SVM	90.51 ± 2.69	79.94 ± 6.24	85.62 ± 4.76	0.5668
		SH-SVM	89.80 ± 3.05	78.20 ± 6.18	86.34 ± 3.69	**0.4707**
Pima	0.1	SRGP-SVM	74.48 ± 3.10	43.65 ± 8.15	**63.01 ± 6.25**	1.4284
(768 × 8)		SGP-SVM	75.78 ± 2.35	**44.68 ± 6.42**	60.87 ± 5.13	0.5971
		SP-SVM	**75.78 ± 2.19**	44.64 ± 5.71	60.61 ± 4.44	0.7341
		SH-SVM	75.65 ± 2.40	44.41 ± 6.43	60.88 ± 5.31	**0.5867**
	1.0	SRGP-SVM	71.36 ± 1.44	**37.96 ± 2.25**	**60.20 ± 2.78**	5.4437
		SGP-SVM	73.04 ± 4.57	37.18 ± 9.50	53.16 ± 7.92	0.6137
		SP-SVM	73.17 ± 4.77	37.52 ± 10.03	53.49 ± 8.35	0.6827
		SH-SVM	**73.30 ± 4.30**	37.63 ± 8.64	53.54 ± 7.74	**0.5757**
Rice	0.1	SRGP-SVM	**90.60 ± 1.18**	**80.95 ± 2.52**	**91.67 ± 0.87**	16.1857
(3810 × 7)		SGP-SVM	80.79 ± 7.32	61.41 ± 14.83	82.65 ± 6.44	4.6910
		SP-SVM	81.21 ± 5.03	62.23 ± 10.17	82.97 ± 4.65	4.4916
		SH-SVM	86.25 ± 2.13	73.62 ± 3.62	87.00 ± 2.29	**2.8010**
	1.0	SRGP-SVM	**85.04 ± 1.19**	**70.01 ± 2.39**	**86.48 ± 1.05**	15.7048
		SGP-SVM	78.79 ± 1.02	57.12 ± 2.28	80.95 ± 0.73	4.8973
		SP-SVM	81.92 ± 1.05	63.38 ± 2.37	83.85 ± 0.59	4.5454
		SH-SVM	52.13 ± 16.39	4.19 ± 32.33	55.05 ± 16.49	**2.9618**
Heart Failure	0.1	SRGP-SVM	**81.60 ± 3.37**	**59.26 ± 4.49**	**71.63 ± 3.64**	1.7145
(299 × 12)		SGP-SVM	50.82 ± 7.79	0.91 ± 16.05	38.45 ± 13.31	0.2901
		SP-SVM	50.16 ± 7.16	1.94 ± 14.27	40.48 ± 12.41	0.2483
		SH-SVM	50.49 ± 8.45	0.88 ± 20.59	39.18 ± 16.09	**0.2072**
	1.0	SRGP-SVM	**72.90 ± 3.94**	**42.15 ± 7.47**	**60.61 ± 5.70**	1.7605
		SGP-SVM	59.50 ± 8.97	15.64 ± 17.44	46.61 ± 10.26	0.3142
		SP-SVM	58.16 ± 7.35	18.24 ± 15.16	48.92 ± 7.38	0.2663
		SH-SVM	52.86 ± 10.01	6.95 ± 19.76	42.47 ± 10.29	**0.2119**
Auction	0.1	SRGP-SVM	**86.93 ± 0.73**	7.70 ± 3.96	5.58 ± 2.58	10.1965
(2043 × 7)		SGP-SVM	66.23 ± 2.60	9.60 ± 6.07	25.05 ± 4.75	3.0673
		SP-SVM	65.45 ± 4.28	**9.81 ± 6.06**	**25.23 ± 4.77**	2.3299
		SH-SVM	41.42 ± 16.89	5.32 ± 6.91	22.72 ± 2.97	**1.4356**
	1.0	SRGP-SVM	**86.29 ± 0.85**	1.90 ± 6.31	3.47 ± 3.91	11.6335
		SGP-SVM	65.78 ± 2.36	0.70 ± 6.68	18.12 ± 4.58	3.4709
		SP-SVM	63.87 ± 11.68	3.20 ± 4.28	20.12 ± 1.00	3.0230
		SH-SVM	62.56 ± 3.39	**5.61 ± 6.99**	**22.55 ± 5.25**	**1.5762**

Next, We will consider the noise case using linear kernel, which can be seen in Table 21 and 22:

- SRGP-SVM achieves the highest ACC, MCC, and F_1 score across various datasets, particularly excelling in noisy environments with low noise levels ($\sigma^2 = 0.1$), although its training time is significantly longer, especially for larger datasets.
- SGP-SVM demonstrates strong performance, particularly in MCC and F_1 scores, but slightly behind SRGP-SVM. It is faster in training, especially on smaller datasets, and handles higher noise levels ($\sigma^2 = 1.0$) effectively, although it struggles a bit with lower noise.
- SP-SVM balances predictive accuracy and speed, offering shorter training times than both SRGP-SVM and SGP-SVM. However, it shows some variation in MCC performance under noisy conditions and is ideal for fast processing where moderate noise resilience is acceptable.
- SH-SVM is the quickest method in terms of training time, making it ideal for time-sensitive applications. It provides reasonable accuracy but performs less effectively in highly noisy environments, with lower MCC and F_1 scores compared to SRGP-SVM and SGP-SVM.

Table 23 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data without noise using a RBF kernel.

Datasets	SH-SVM	SP-SVM	SGP-SVM	SRGP-SVM
	C, μ	τ, C, μ	$\tau_1, \tau_2, \epsilon_1, \epsilon_2$	$\tau_1, \tau_2, \epsilon_1, \epsilon_2$
			C, μ	λ, η, C, μ
Ionosphere	1, 0.1	0.9, 1, 0.1	0.1, 0.1, 0.01, 0.01	0.1, 0.1, 0.01, 0.01
			1, 0.1	1, 5, 1, 0.1
Australian	1, 0.1	0.1, 1, 0.1	0.5, 0.5, 0.01, 0.01	0.5, 0.5, 0.05, 0.01
			1, 0.1	0.5, 5, 1, 0.1
Diabetes	1, 0.1	0.1, 1, 0.1	0.1, 0.1, 0.05, 0.05	0.5, 0.5, 0.05, 0.05
			1, 0.1	0.5, 5, 1, 0.1
Appendicitis	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05	0.1, 0.9, 0.05, 0.05
			1, 0.1	1, 5, 1, 0.1
Heart	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.05	0.1, 0.1, 0.05, 0.05
			1, 0.1	1, 5, 1, 0.1
WDBC	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.05	0.1, 0.9, 0.05, 0.01
			1, 0.1	1, 5, 1, 0.1
Pima	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05	0.1, 0.1, 0.05, 0.05
			1, 0.1	1, 5, 1, 0.1
Rice	1, 0.1	0.9, 1, 0.1	0.9, 0.9, 0.01, 0.05	0.1, 0.9, 0.05, 0.05
			1, 0.1	1, 5, 1, 0.1
Heart Failure	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05	0.1, 0.1, 0.05, 0.05
			1, 0.1	0.5, 5, 1, 0.1
Auction	1, 0.1	0.9, 1, 0.1	0.5, 0.5, 0.01, 0.05	0.5, 0.5, 0.05, 0.05
			1, 0.1	1, 5, 1, 0.1

Table 24 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise ($\sigma^2 = 0.1$) using a RBF kernel.

Datasets	SH-SVM C, μ	SP-SVM τ, C, μ	SGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ C, μ	SRGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ λ, η, C, μ
Ionosphere	1, 0.1	0.1, 1, 0.1	0.9, 0.2, 0.01, 0.05 1, 0.1	0.9, 0.2, 0.05, 0.05 1, 0.5, 1, 0.1
Australian	1, 0.1	0.9, 1, 0.1	0.1, 0.1, 0.01, 0.01 1, 0.1	0.1, 0.1, 0.05, 0.05 0.5, 5, 1, 0.1
Diabetes	1, 0.1	0.1, 1, 0.1	0.1, 0.1, 0.05, 0.05 1, 0.1	0.5, 0.5, 0.05, 0.05 0.5, 5, 1, 0.1
Appendicitis	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.9, 0.05, 0.05 1, 5, 1, 0.1
Heart	1, 0.1	0.5, 1, 0.1	0.5, 0.1, 0.05, 0.01 1, 0.1	0.2, 0.5, 0.01, 0.05 1, 0.5, 1, 0.1
WDBC	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.9, 0.05, 0.05 1, 5, 1, 0.1
Pima	1, 0.1	0.9, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.1, 0.05, 0.05 1, 5, 1, 0.1
Rice	1, 0.1	0.9, 1, 0.1	0.1, 0.9, 0.01, 0.05 1, 0.1	0.1, 0.9, 0.05, 0.05 1, 5, 1, 0.1
Heart Failure	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.1, 0.05, 0.05 0.5, 5, 1, 0.1
Auction	1, 0.1	0.9, 1, 0.1	0.9, 0.9, 0.01, 0.01 1, 0.1	0.5, 0.5, 0.05, 0.05 1, 2, 1, 0.1

Table 25 The optimal parameters of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise ($\sigma^2 = 1.0$) using a RBF kernel.

Datasets	SH-SVM C, μ	SP-SVM τ, C, μ	SGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ C, μ	SRGP-SVM $\tau_1, \tau_2, \epsilon_1, \epsilon_2$ λ, η, C, μ
Ionosphere	1, 0.1	0.9, 1, 0.1	0.1, 0.1, 0.01, 0.01 1, 0.1	0.1, 0.1, 0.05, 0.05 1, 5, 1, 0.1
Australian	1, 0.1	0.1, 1, 0.1	0.5, 0.5, 0.01, 0.01 1, 0.1	0.5, 0.5, 0.05, 0.01 0.5, 5, 1, 0.1
Diabetes	1, 0.1	0.1, 1, 0.1	0.5, 0.5, 0.05, 0.05 1, 0.1	0.5, 0.5, 0.05, 0.05 0.5, 2, 1, 0.1
Appendicitis	1, 0.1	0.2, 1, 0.1	0.1, 0.9, 0.05, 0.01 1, 0.1	0.5, 0.9, 0.05, 0.01 1, 2, 1, 0.1
Heart	1, 0.1	0.2, 1, 0.1	0.5, 0.5, 0.05, 0.01 1, 0.1	0.9, 0.2, 0.01, 0.01 0.5, 0.1
WDBC	1, 0.1	0.2, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.2, 0.9, 0.01, 0.05 1, 5, 1, 0.1
Pima	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.1, 0.05, 0.05 1, 2, 1, 0.1
Rice	1, 0.1	0.9, 1, 0.1	0.1, 0.9, 0.01, 0.05 1, 0.1	0.1, 0.9, 0.01, 0.05 1, 5, 1, 0.1
Heart Failure	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.1, 0.1, 0.05, 0.05 0.5, 5, 1, 0.1
Auction	1, 0.1	0.1, 1, 0.1	0.9, 0.1, 0.05, 0.05 1, 0.1	0.2, 0.2, 0.01, 0.05 1, 0.1, 1, 0.1

Overall, SRGP-SVM stands out as the most reliable and robust method, excelling in both predictive accuracy and handling noise, though it requires more computation time. For scenarios where time is a critical factor, SH-SVM provides a more efficient option while maintaining reasonable accuracy levels.

We then consider the noiseless case for the RBF kernel, as shown in Table 26. It can be observed that:

- SRGP-SVM demonstrated the highest accuracy across datasets like Ionosphere, Australian, and WDBC, indicating its effectiveness in various classification tasks. It excelled in metrics such as Matthews Correlation Coefficient

ever, it generally required longer training times, especially on larger datasets like Rice.

- SGP-SVM delivered competitive accuracy, notably on datasets like Australian and WDBC, where it ranked closely behind SRGP-SVM. It showed strengths in MCC and F1 scores, making it a solid choice for performance-oriented tasks. Additionally, it benefited from significantly shorter training times, especially in smaller datasets, allowing for quicker model deployment.

- SP-SVM achieved accuracy levels similar to SGP-SVM on datasets such as Heart and Diabetes, while maintaining reasonable F1 scores. This method consistently required shorter training times across all datasets, making it suitable for applications that demand fast processing without compromising predictive performance.

- SH-SVM excelled in training speed, achieving the fastest times on datasets like Ionosphere and Australian. While it maintained high accuracy, it sometimes lagged in key metrics like MCC and F1 scores in noisy environments. This indicates a potential trade-off between computational efficiency and predictive accuracy, making it best suited for time-sensitive applications.

Overall, SRGP-SVM stands out as the most reliable and accurate method, particularly in noise-free environments, while SGP-SVM and SP-SVM offer a favorable balance of accuracy and speed. In contrast, SH-SVM provides a suitable option for time-sensitive applications, albeit with some sacrifice in predictive performance.

Table 26 Comparisons of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data without noise using a RBF kernel.

Dataset	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time
Ionosphere	SRGP-SVM	**71.22 ± 6.00**	43.38 ± 10.41	75.29 ± 5.80	0.9070
(350 × 34)	SGP-SVM	70.36 ± 5.10	**47.36 ± 6.70**	**72.38 ± 4.35**	0.3462
	SP-SVM	70.65 ± 4.05	45.01 ± 6.71	73.52 ± 3.36	0.3171
	SH-SVM	64.08 ± 18.67	39.60 ± 20.28	60.45 ± 30.33	**0.2496**
Australian	SRGP-SVM	80.43 ± 2.43	61.23 ± 4.48	77.47 ± 4.14	4.1597
(690 × 14)	SGP-SVM	75.65 ± 4.11	56.23 ± 6.77	77.05 ± 4.96	0.6939
	SP-SVM	72.17 ± 4.84	51.62 ± 7.66	74.95 ± 5.04	0.5257
	SH-SVM	**85.65 ± 0.85**	**72.04 ± 1.88**	**84.98 ± 1.82**	**0.5067**
Diabetes	SRGP-SVM	**71.88 ± 2.66**	32.44 ± 8.95	37.80 ± 11.67	4.3275
(1151 × 19)	SGP-SVM	67.97 ± **3.14**	39.00 ± 4.86	63.03 ± 3.50	0.6549
	SP-SVM	67.06 ± 2.18	38.78 ± 3.51	63.00 ± 2.77	0.5549
	SH-SVM	67.58 ± 2.41	**39.09 ± 3.53**	**63.14 ± 3.19**	**0.4656**
Appendicitis	SRGP-SVM	**88.79 ± 8.53**	**49.23 ± 29.36**	**53.03 ± 29.95**	0.1291
(106 × 7)	SGP-SVM	83.07 ± 8.68	31.85 ± 21.41	33.89 ± 21.97	**0.0801**
	SP-SVM	82.12 ± 10.06	25.52 ± 24.93	28.89 ± 25.92	0.1110
	SH-SVM	83.07 ± 10.98	28.98 ± 30.90	32.70 ± 31.90	0.1041
Heart	SRGP-SVM	73.70 ± 14.83	42.48 ± 34.75	81.22 ± 8.02	1.0153
(303 × 14)	SGP-SVM	84.07 ± 4.77	67.48 ± 9.68	85.98 ± 4.01	**0.2005**
	SP-SVM	**84.44 ± 3.43**	**68.42 ± 6.87**	**86.07 ± 3.22**	0.2380
	SH-SVM	84.07 ± 4.77	67.50 ± 9.64	85.85 ± 4.24	0.2336
WDBC	SRGP-SVM	95.78 ± 1.41	90.93 ± 3.10	94.09 ± 2.14	2.4985
(569 × 30)	SGP-SVM	96.48 ± 0.79	92.56 ± 1.52	95.15 ± 0.89	0.5771
	SP-SVM	96.66 ± 0.66	92.93 ± 1.34	95.42 ± 0.78	0.5029
	SH-SVM	**97.01 ± 1.19**	**93.66 ± 2.52**	**96.00 ± 1.55**	**0.3417**
Pima	SRGP-SVM	67.58 ± 2.81	11.19 ± 14.49	16.67 ± 20.76	2.7284
(768 × 8)	SGP-SVM	70.96 ± 2.27	30.94 ± 8.11	40.09 ± 3.96	0.6330
	SP-SVM	71.22 ± 1.95	31.89 ± 7.43	40.04 ± 2.74	0.6830
	SH-SVM	**71.22 ± 2.58**	**31.54 ± 9.14**	**40.92 ± 4.97**	**0.5880**
Rice	SRGP-SVM	**88.19 ± 1.83**	**76.39 ± 3.44**	**90.12 ± 0.99**	15.7806
(3810 × 7)	SGP-SVM	76.12 ± 7.26	59.10 ± 9.47	73.80 ± 9.97	4.2263
	SP-SVM	86.27 ± 7.06	74.19 ± 10.75	86.08 ± 9.68	3.6930
	SH-SVM	71.71 ± 2.23	54.16 ± 3.12	67.55 ± 2.86	**2.3791**
Heart Failure	SRGP-SVM	**77.56 ± 6.92**	43.64 ± 17.23	55.76 ± 15.67	1.1790
(299 × 12)	SGP-SVM	76.58 ± 8.12	43.96 ± 12.84	54.52 ± 9.34	**0.2106**
	SP-SVM	76.57 ± 8.02	42.65 ± 15.68	53.18 ± 11.96	0.2887
	SH-SVM	77.25 ± 8.37	**45.78 ± 14.45**	**56.32 ± 10.92**	0.2805
Auction	SRGP-SVM	**80.61 ± 13.61**	1.99 ± 3.99	5.38 ± 10.77	5.1431
(2043 × 7)	SGP-SVM	54.04 ± 2.81	1.46 ± 5.09	20.85 ± 3.15	1.8624
	SP-SVM	56.88 ± 2.92	2.80 ± 4.03	**21.36 ± 2.56**	1.6964
	SH-SVM	57.38 ± 23.01	**3.05 ± 5.69**	20.34 ± 4.38	**1.2496**

Table 27 Comparisons of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise using a RBF kernel.

Dataset	σ^2	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time
Ionosphere	0.1	SRGP-SVM	72.09 ± 1.54	36.03 ± 18.34	76.70 ± 4.35	1.3068
(350 × 34)		SGP-SVM	69.81 ± 1.53	44.29 ± 4.88	72.46 ± 2.08	0.2861
		SP-SVM	**72.37 ± 1.84**	**46.24 ± 3.68**	**75.90 ± 1.71**	0.2902
		SH-SVM	62.09 ± 17.56	34.34 ± 17.54	59.10 ± 29.60	**0.2403**
	1.0	SRGP-SVM	**65.82 ± 7.92**	10.46 ± 20.92	**78.65 ± 4.49**	2.0817
		SGP-SVM	62.96 ± 3.00	32.97 ± 3.29	64.72 ± 3.86	0.3048
		SP-SVM	65.25 ± 2.81	**33.47 ± 5.86**	68.97 ± 2.94	0.3168
		SH-SVM	44.18 ± 12.28	8.35 ± 16.70	14.88 ± 29.77	**0.2634**
Australian	0.1	SRGP-SVM	**80.87 ± 4.26**	**61.61 ± 7.86**	**77.37 ± 4.51**	4.4095
(690 × 14)		SGP-SVM	72.90 ± 2.73	51.55 ± 5.01	75.04 ± 3.74	0.8232
		SP-SVM	70.72 ± 3.68	48.84 ± 6.30	73.80 ± 4.25	0.7175
		SH-SVM	74.93 ± 5.40	55.25 ± 7.53	76.87 ± 3.78	**0.5443**
	1.0	SRGP-SVM	**70.14 ± 6.62**	37.77 ± 19.29	52.73 ± 26.63	4.1535
		SGP-SVM	67.39 ± 3.92	**39.39 ± 7.86**	**69.76 ± 4.43**	0.6910
		SP-SVM	66.38 ± 3.45	38.27 ± 6.86	69.46 ± 3.94	0.5225
		SH-SVM	66.67 ± 3.31	38.92 ± 6.27	69.69 ± 3.94	**0.4465**
Diabetes	0.1	SRGP-SVM	**68.62 ± 0.89**	17.35 ± 14.87	19.03 ± 16.23	4.5629
(1151 × 19)		SGP-SVM	67.45 ± 2.73	38.58 ± 4.25	62.84 ± 4.09	0.6801
		SP-SVM	66.93 ± 2.67	**39.28 ± 4.00**	**63.33 ± 3.13**	0.5643
		SH-SVM	66.54 ± 2.69	37.37 ± 4.06	62.24 ± 3.85	**0.4702**
	1.0	SRGP-SVM	**66.80 ± 5.73**	12.21 ± 15.21	16.75 ± 21.05	4.4119
		SGP-SVM	60.29 ± 3.99	**25.37 ± 8.14**	56.23 ± 5.35	0.7256
		SP-SVM	58.86 ± 3.98	25.17 ± 7.39	**56.40 ± 5.29**	0.5506
		SH-SVM	58.99 ± 4.57	24.81 ± 8.62	56.14 ± 5.75	**0.4389**
Appendicitis	0.1	SRGP-SVM	**87.84 ± 8.56**	**45.31 ± 27.70**	**49.45 ± 28.13**	0.2143
(106 × 7)		SGP-SVM	84.94 ± 8.12	39.93 ± 22.89	39.76 ± 24.21	**0.0743**
		SP-SVM	83.03 ± 9.30	29.45 ± 24.11	29.43 ± 24.64	0.1386
		SH-SVM	83.98 ± 9.76	33.61 ± 28.09	34.76 ± 28.87	0.1318
	1.0	SRGP-SVM	**82.16 ± 9.44**	9.18 ± 18.35	8.89 ± 17.78	0.5258
		SGP-SVM	82.12 ± 10.92	**21.46 ± 18.69**	**20.38 ± 17.03**	**0.0706**
		SP-SVM	81.21 ± 11.46	14.33 ± 18.72	14.67 ± 18.09	0.1081
		SH-SVM	82.16 ± 10.36	20.66 ± 18.14	19.67 ± 16.75	0.0949
Heart	0.1	SRGP-SVM	67.04 ± 12.96	25.69 ± 31.96	76.27 ± 7.05	1.2488
(303 × 14)		SGP-SVM	**82.22 ± 4.16**	**63.98 ± 8.16**	**84.32 ± 4.04**	0.2178
		SP-SVM	80.37 ± 4.48	60.41 ± 8.95	82.78 ± 4.67	0.2633
		SH-SVM	81.48 ± 4.22	62.43 ± 8.39	83.81 ± 3.95	**0.2039**
	1.0	SRGP-SVM	59.63 ± 11.07	11.59 ± 23.19	73.30 ± 6.92	0.9577
		SGP-SVM	**75.56 ± 4.12**	**49.54 ± 8.27**	78.95 ± 4.27	0.2305
		SP-SVM	74.07 ± 4.22	46.58 ± 8.96	77.65 ± 4.48	**0.2071**
		SH-SVM	75.56 ± 4.29	49.43 ± 8.91	**78.90 ± 4.49**	0.2248

Table 28 Comparisons of SRGP-SVM, SGP-SVM, SP-SVM and SH-SVM on UCI data with noise using a RBF kernel (continued).

Dataset	σ^2	Criteria	ACC (%)	MCC (%)	F_1 (%)	Time
WDBC	0.1	SRGP-SVM	95.25 ± 1.06	89.86 ± 2.44	93.46 ± 1.73	2.2572
(569 × 30)		SGP-SVM	**95.78 ± 0.87**	**91.08 ± 1.62**	**94.28 ± 0.94**	0.6164
		SP-SVM	95.08 ± 2.52	89.62 ± 5.33	93.17 ± 3.67	0.6398
		SH-SVM	95.08 ± 1.32	89.56 ± 2.73	93.38 ± 1.63	**0.3551**
	1.0	SRGP-SVM	**92.44 ± 0.41**	84.08 ± 0.99	**89.95 ± 0.89**	1.9893
		SGP-SVM	92.27 ± 1.51	83.61 ± 3.18	89.58 ± 2.01	0.6392
		SP-SVM	92.44 ± 1.19	**84.09 ± 2.51**	89.74 ± 1.77	0.5309
		SH-SVM	90.68 ± 1.20	80.35 ± 2.49	87.49 ± 1.36	**0.4811**
Pima	0.1	SRGP-SVM	65.63 ± 3.84	7.01 ± 9.39	11.28 ± 14.63	2.7359
(768 × 8)		SGP-SVM	70.18 ± 3.27	28.64 ± 6.63	38.05 ± 5.07	0.5804
		SP-SVM	70.31 ± 3.02	29.46 ± 6.61	36.99 ± 6.21	0.5702
		SH-SVM	**70.70 ± 2.54**	**30.22 ± 5.15**	**39.76 ± 2.65**	**0.5197**
	1.0	SRGP-SVM	65.36 ± 3.48	3.00 ± 6.01	3.45 ± 6.90	2.2736
		SGP-SVM	65.89 ± 2.66	**12.94 ± 5.02**	10.83 ± 10.01	0.5833
		SP-SVM	**66.02 ± 3.44**	11.58 ± 7.15	9.04 ± 11.53	0.5883
		SH-SVM	65.76 ± 2.46	12.28 ± 4.99	**11.41 ± 9.43**	**0.4688**
Rice	0.1	SRGP-SVM	**85.67 ± 1.61**	**72.06 ± 2.60**	**88.49 ± 0.95**	16.5398
(3810 × 7)		SGP-SVM	70.37 ± 2.18	51.87 ± 3.00	65.66 ± 3.08	3.5564
		SP-SVM	84.91 ± 6.93	71.53 ± 10.33	84.71 ± 9.87	3.6760
		SH-SVM	71.21 ± 1.90	53.02 ± 2.64	66.94 ± 2.89	**2.3754**
	1.0	SRGP-SVM	**77.66 ± 2.89**	**56.45 ± 4.30**	**83.15 ± 2.17**	18.0738
		SGP-SVM	66.06 ± 2.32	41.20 ± 7.80	61.72 ± 0.90	4.0470
		SP-SVM	69.24 ± 5.78	45.54 ± 7.88	66.64 ± 7.81	3.8511
		SH-SVM	69.48 ± 1.15	48.43 ± 1.41	65.46 ± 0.74	**2.4823**
Heart Failure	0.1	SRGP-SVM	**76.90 ± 9.05**	**45.45 ± 17.63**	**53.79 ± 13.85**	1.1309
(299 × 12)		SGP-SVM	75.89 ± 7.75	40.87 ± 16.92	51.18 ± 12.68	**0.1993**
		SP-SVM	76.56 ± 7.93	43.01 ± 17.87	52.03 ± 13.31	0.3105
		SH-SVM	76.56 ± 7.93	42.76 ± 18.03	53.10 ± 13.79	0.2979
	1.0	SRGP-SVM	**72.89 ± 4.26**	**30.42 ± 17.33**	**40.87 ± 16.77**	0.9750
		SGP-SVM	69.55 ± 8.03	22.64 ± 10.39	24.86 ± 9.79	**0.2340**
		SP-SVM	69.22 ± 8.38	21.32 ± 10.80	23.43 ± 11.52	0.3750
		SH-SVM	69.89 ± 7.73	22.93 ± 10.39	27.23 ± 8.41	0.3753
Auction	0.1	SRGP-SVM	**74.05 ± 16.01**	1.37 ± 1.70	8.74 ± 10.72	5.4243
(2043 × 7)		SGP-SVM	64.31 ± 11.25	**4.66 ± 7.20**	**21.70 ± 3.16**	2.3643
		SP-SVM	56.97 ± 6.09	1.93 ± 3.21	20.74 ± 1.80	1.8383
		SH-SVM	57.58 ± 23.13	1.87 ± 4.16	19.31 ± 3.24	**1.3417**
	1.0	SRGP-SVM	67.06 ± 5.64	**4.31 ± 9.42**	19.60 ± 6.96	2.4708
		SGP-SVM	**68.28 ± 4.10**	3.30 ± 2.52	19.80 ± 1.75	2.0067
		SP-SVM	67.21 ± 3.20	3.64 ± 2.92	**20.21 ± 1.91**	1.6795
		SH-SVM	64.81 ± 1.54	0.52 ± 5.98	18.19 ± 4.32	**1.2385**

Finally, we review the noisy case for the RBF kernel in Table 27 and 28. It can be observed that:

- SRGP-SVM generally exhibits competitive accuracy and F_1 scores, particularly in the Ionosphere and Australian datasets. It shows the highest F_1 score across multiple datasets, indicating robust performance in balancing precision and recall. However, it often has longer computation times, especially with larger datasets, such as Rice.
- SGP-SVM Offers consistent performance across datasets, especially with a lower noise level ($\sigma^2 = 0.1$). It frequently achieves commendable MCC scores and provides a good balance between accuracy and time efficiency, especially evident in smaller datasets like Appendicitis.
- SP-SVM competes closely with both SRGP-SVM and SGP-SVM in terms of accuracy and F_1 scores. Notably, it performs well in various metrics across most datasets, making it a strong contender, though it does not outperform SRGP-SVM in most scenarios.
- SH-SVM generally lags behind the other methods in accuracy and F_1 scores, particularly in the presence of higher noise levels ($\sigma^2 = 1.0$). While it shows reduced computation times in many cases, its lower performance in classification metrics indicates it may not be suitable for applications requiring high accuracy.

Overall, SRGP-SVM emerges as the most reliable method for various datasets, especially when noise levels are low. SGP-SVM and SP-SVM follow closely, offering robust alternatives depending on the specific dataset and required computational resources. SH-SVM may be reconsidered for applications where speed is prioritized over accuracy.

Based on the experimental outcomes, we observe that as the noise level increases, η in SRGP-SVM decreases to mitigate its impact while still maintaining strong performance. Additionally, the values of τ_1, τ_2, ϵ_1, ϵ_2 in other methods, along

with λ in SRGP-SVM, should be optimized for each data type. Overall, SRGP-SVM demonstrates notable performance even in datasets without noise. Moreover, it excels particularly when the data exhibits heightened levels of noise, outperforming other baseline methods in such scenarios. Specifically, SRGP-SVM possesses the flexibility to adjust its parameters and the boundedness of its loss function.

3.3.5 STATISTICAL ANALYSIS

The Friedman Test, accompanied by post hoc analysis [47], serves to evaluate the statistical significance of the proposed SRGP-SVM against SGP-SVM, SP-SVM, and SH-SVM. The method with the lowest rank value in the Friedman test is considered the best performing. We will utilize the ACC, MCC, and F_1-score for all case.

For accuracy (ACC), under the assumption of the null hypothesis where all methods are considered equivalent, the computation of the Friedman statistic unfolds as:

$$\chi_F^2 = \frac{12 \times 60}{4 \times (4+1)} \left[(1.84^2 + 2.73^2 + 2.92^2 + 2.52^2) - \frac{4 \times 5^2}{4} \right] \approx 25.35,$$
$$F_F = \frac{(60-1) \times 25.35}{60 \times (4-1) - 25.35} \approx 9.67.$$

For Matthews Correlation Coefficient (MCC), the Friedman statistic computation unfolds as:

$$\chi_F^2 = \frac{12 \times 60}{4 \times (4+1)} \left[(2.45^2 + 2.60^2 + 2.53^2 + 2.42^2) - \frac{4 \times 5^2}{4} \right] \approx 0.71,$$
$$F_F = \frac{(60-1) \times 0.71}{60 \times (4-1) - 0.71} \approx 0.23.$$

Similarly, for the F_1-score, the Friedman statistic is computed as:

$$\chi_F^2 = \frac{12 \times 60}{4 \times (4+1)} \left[(2.30^2 + 2.65^2 + 2.73^2 + 2.32^2) - \frac{4 \times 5^2}{4} \right] \approx 5.32,$$
$$F_F = \frac{(60-1) \times 5.32}{60 \times (4-1) - 5.32} \approx 1.80.$$

We observe that at a confidence level of 0.05, the critical value of $F(3, 177)$ is 2.75. Thus, for ACC, where $9.67 > 2.75$, the null hypothesis is rejected. This indicates significant differences among the algorithms at a 0.05 confidence level. However, with regards to the MCC and the F_1-score, where $0.23 < 2.75$ and $1.80 < 2.75$,

respectively, the null hypothesis is not rejected. This suggests no significant differences among the algorithms at the 0.05 confidence level for this particular metric. From the Nemenyi post hoc test, with a critical difference of $CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$ yielding $CD = 0.80$, it is observed that the proposed SRGP-SVM outperforms both SGP-SVM and SP-SVM but does not exhibit significant differences compared to SH-SVM in ACC. Nevertheless, the average rank of ACC for SRGP-SVM is the lowest compared to the other methods, while maintaining strong MCC and F_1-score illustrated in Figure 20.

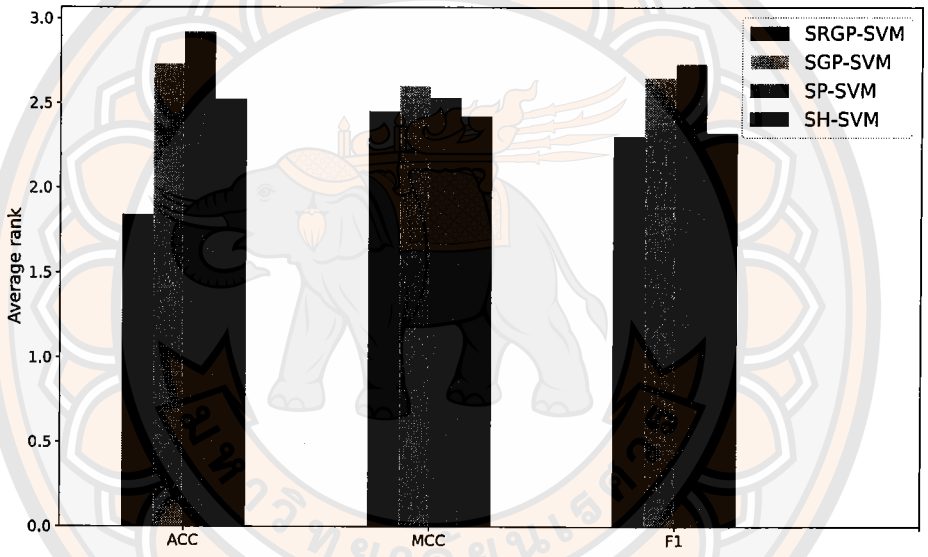


Figure 20 The average rank for ACC, MCC, F_1-score of each methods

CHAPTER IV

REGULARIZED NESTEROV'S ACCELERATED DAMPED BFGS METHOD FOR STOCHASTIC OPTIMIZATION WITH APPLICATIONS

In this section, we introduce a novel adaptive momentum coefficient in regularized stochastic Nesterov's accelerated quasi-Newton methods for stochastic optimization problems, including nonconvex cases. For any iteration k, we can see that compute μ_k in (2.5.8) adjusting between μ and 0. Although using the momentum of μ_k defined by the constant μ value to switch to zero will reduce the negative impact of the momentum and improve the overall performance of the model. However, the constant μ may lead to overshooting problem. Since the momentum μ should decrease as the method approaches the stationary point to avoid the overshoot problem. Therefore, we should define a μ value that can be adjusted to decrease. Moreover, μ_k in (2.5.8) by adjusting by observing the value of the objective function ($F(\mathbf{w}_k) := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w}_k, \xi_{k,i})$ and $F(\mathbf{w}_k + \mu\mathbf{z}_k) := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w}_k + \mu\mathbf{z}_k, \xi_{k,i})$) with a large dataset, it incurs a high computational cost and setting fixed values μ is not flexible. Therefore, we proposed flexible momentum coefficient by observe from the objective function of some samples of the dataset ($\hat{f}(\mathbf{w}_k, \hat{\xi}_k) := \frac{1}{L_k}\sum_{i=1}^{L_k} f(\mathbf{w}_k, \xi_{k,i})$ and $\hat{f}(\mathbf{w}_k + \beta_k\mathbf{z}_k, \hat{\xi}_k) := \frac{1}{L_k}\sum_{i=1}^{L_k} f(\mathbf{w}_k + \beta_k\mathbf{z}_k, \xi_{k,i})$) instead. Since $L_k \leq N$, then obviously it reduces the constant computation greatly if $L_k \ll N$. The novel adaptive momentum coefficient is defined as follows

$$\mu_k = \begin{cases} \beta_k, & \text{if } \hat{f}(\mathbf{w}_k + \beta_k\mathbf{z}_k, \hat{\xi}_k) < \hat{f}(\mathbf{w}_k, \hat{\xi}_k), \\ 0, & \text{otherwise}, \end{cases} \tag{4.0.11}$$

where $\beta_k \in (0, 1)$ and $\beta_k \leq \beta_{k-1}$ for all k (Note that this can be satisfied if β_k is non-increasing). Therefore, it can be seen that the novel adaptive momentum coefficient offers greater flexibility in adjusting the momentum coefficient and reducing computational cost. We now present our DRES-NAQ method as Algorithm 4.

Algorithm 4 DRES-NAQ

Require: A training set represented as a collection $\Theta = [\theta_1, \ldots, \theta_N]$. Choose an initial point $\mathbf{w}_1$, an initial symmetric and positive definite matrix $\mathbf{B}_1$, $\mathbf{z}_1 = \mathbf{0}$, positive constants γ and δ such that $0.8\gamma < \delta$, batch sizes $\{L_k\}_{k\geq 1}$ and $\{m_k\}_{k\geq 1}$, where $L_k < N$ and $m_k < N$ for all k, momentum coefficient $\{\beta_k\}_{k\geq 1}$ and step sizes $\{\alpha_k\}_{k\geq 1}$.

1: **for** $k = 1, 2, \ldots$ **do**

2: Select L_k sample $\hat{\xi}_k = [\xi_{k,1}, \ldots, \xi_{k,L_k}]$ from $\Theta = [\theta_1, \ldots, \theta_N]$.

3: Calculate $\hat{f}(\mathbf{w}_k, \hat{\xi}_k) := \frac{1}{L_k}\sum_{i=1}^{L_k} f(\mathbf{w}_k, \xi_{k,i})$ and $\hat{f}(\mathbf{w}_k + \beta_k \mathbf{z}_k, \hat{\xi}_k) := \frac{1}{L_k}\sum_{i=1}^{L_k} f(\mathbf{w}_k + \beta_k \mathbf{z}_k, \xi_{k,i})$

4: Calculate momentum coefficient

$$\mu_k = \begin{cases} \beta_k, & \text{if } \hat{f}(\mathbf{w}_k + \beta_k \mathbf{z}_k, \hat{\xi}_k) < \hat{f}(\mathbf{w}_k, \hat{\xi}_k), \\ 0, & \text{otherwise.} \end{cases}$$

5: Select m_k sample $\xi_k = [\xi_{k,1}, \ldots, \xi_{k,m_k}]$ from $\Theta = [\theta_1, \ldots, \theta_N]$.

6: Calculate stochastic gradient $\mathbf{g}(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_k) = \frac{1}{m_k}\sum_{i=1}^{m_k} \nabla f(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_{k,i})$.

7: Generate a positive definite Hessian inverse approximation $\mathbf{H}_k = \mathbf{B}_k^{-1}$.

8: Update

$$\mathbf{z}_{k+1} = \mu_k \mathbf{z}_k - \alpha_k(\mathbf{H}_k + \Gamma \mathbf{I})\mathbf{g}(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_k),$$
$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{z}_{k+1}.$$

9: Calculate stochastic gradient $\mathbf{g}(\mathbf{w}_{k+1}, \xi_k) = \frac{1}{m_k}\sum_{i=1}^{m_k} \nabla f(\mathbf{w}_{k+1}, \xi_{k,i})$.

10: Calculate $\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{z}_k)$ and $\mathbf{y}_k = \mathbf{g}(\mathbf{w}_{k+1}, \xi_k) - \mathbf{g}(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_k)$.

11: Calculate damped parameter

$$\phi_k = \begin{cases} \dfrac{0.8\mathbf{s}_k^T(\mathbf{B}_k + \gamma \mathbf{I})\mathbf{s}_k - \delta \mathbf{s}_k^T \mathbf{s}_k}{\mathbf{s}_k^T(\mathbf{B}_k + \gamma \mathbf{I})\mathbf{s}_k - \mathbf{s}_k^T \mathbf{y}_k}, & \text{if } \mathbf{s}_k^T \mathbf{y}_k \leq 0.2\mathbf{s}_k^T(\mathbf{B}_k + \gamma \mathbf{I})\mathbf{s}_k + \delta \mathbf{s}_k^T \mathbf{s}_k, \\ 1, & \text{otherwise.} \end{cases}$$

12: Calculate $\hat{\mathbf{y}}_k = \phi_k \mathbf{y}_k + (1 - \phi_k)(\mathbf{B}_k + \gamma \mathbf{I})\mathbf{s}_k - \delta \mathbf{s}_k$.

13: Update Hessian approximation matrix

$$\mathbf{B}_{k+1} = \mathbf{B}_k - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^\top \mathbf{B}_k}{\mathbf{s}_k^\top \mathbf{B}_k \mathbf{s}_k} + \frac{\hat{\mathbf{y}}_k \hat{\mathbf{y}}_k^\top}{\hat{\mathbf{y}}_k^\top \mathbf{s}_k} + \delta \mathbf{I}.$$

14: **end for**

By setting $\beta_k = \mu$, $L_k = N$, and $\theta_k = 1$ for all k, DRES-NAQ simplifies to RES-NAQ [94]. Additionally, when $\mu_k = 0$ for all k, DRES-NAQ corresponds to the damped RES method presented in [96]. Therefore, DRES-NAQ serves as a unified framework encompassing both RES-NAQ and damped RES, combining the advantages of these approaches.

4.1 Convergence and complexity analysis of DRES-NAQ with diminishing step size

We assume that an noisy gradients of F outputs a stochastic gradient $\nabla f(\mathbf{w}, \xi)$ of F for a given $\mathbf{w}$. The following assumptions are required throughout this paper. The assumptions in this section are based on those outlined in [95].

Assumption 4.1.1. $F : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. $F(\mathbf{w})$ is lower bounded by a real number F^{low} for any $\mathbf{w} \in \mathbb{R}^n$. ∇F is globally Lipschitz continuous with Lipschitz constant L; namely for any $\mathbf{v}, \mathbf{u} \in \mathbb{R}^n$,

$$\|\nabla F(\mathbf{v}) - \nabla F(\mathbf{u})\| \leq L\|\mathbf{v} - \mathbf{u}\|.$$

Assumption 4.1.2. For any iteration k, we have

$$\mathbb{E}_{\xi_k}\left[\mathbf{g}(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_k)\right] = \nabla F\left(\mathbf{w}_k + \mu_k \mathbf{z}_k\right),$$

$$\mathbb{E}_{\xi_k}\left[\|\mathbf{g}(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_k) - \nabla F\left(\mathbf{w}_k + \mu_k \mathbf{z}_k\right)\|^2\right] \leq \sigma^2,$$

where $\sigma > 0$ is the noise level of the gradient estimation, and $\xi_k, k = 1, 2, \ldots$, are independent samples, and for a given k the random variable ξ_k is independent of $\{\mathbf{w}_j + \mu_j \mathbf{z}_j\}_{j=1}^k$.

From Assumption 4.1.2 we can see that $\mathbf{g}(\mathbf{w}_k, \xi_k)$ has the following properties:

$$\mathbb{E}\left[\mathbf{g}(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_k) \mid \mathbf{w}_k + \mu_k \mathbf{z}_k,\right] = \nabla F\left(\mathbf{w}_k + \mu_k \mathbf{z}_k\right), \tag{4.1.1}$$

$$\mathbb{E}\left[\|\mathbf{g}(\mathbf{w}_k + \mu_k \mathbf{z}_k, \xi_k) - \nabla F\left(\mathbf{w}_k + \mu_k \mathbf{z}_k\right)\|^2 \mid \mathbf{w}_k + \mu_k \mathbf{z}_k,\right] \leq \frac{\sigma^2}{m_k}. \tag{4.1.2}$$

Assumption 4.1.3. There exist two positive constants $\underline{\kappa}, \bar{\kappa}$ such that

$$\underline{\kappa} I \preceq \mathbf{H}_k \preceq \bar{\kappa} I \quad , \forall k.$$

We denote $\xi_k = (\xi_{k,1}, \ldots, \xi_{k,m_k})$ as the random samples in the k-th iteration, and $\xi_{[k]} = (\xi_1, \ldots, \xi_k)$ as the random samples from the first k iterations. Since $\mathbf{H}_k$ is generated iteratively from historical gradients via a random process, we assume the following for $\mathbf{H}_k$ ($k \geq 2$) to control the randomness (note that $\mathbf{H}_1$ is initialized).

Assumption 4.1.4. For any $k \geq 2$, the random variable $\mathbf{H}_k$ depends only on $\xi_{[k-1]}$.

It then follows directly from Assumption 4.1.4 and (4.1.1) that

$$\mathbb{E}\left[(\mathbf{H}_k + \Gamma\mathbf{I})\mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k) \mid \xi_{[k-1]}\right] = (\mathbf{H}_k + \Gamma\mathbf{I})\nabla F\left(\mathbf{w}_k + \mu_k\mathbf{z}_k\right), \quad (4.1.3)$$

where the expectation is taken with respect to ξ_k generated in the computation of $\mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k)$.

Assumption 4.1.5. For any iteration k, we have

$$\mathbb{E}_{\hat{\xi}_k}\left[\hat{f}(\mathbf{w}_k, \hat{\xi}_k)\right] = F\left(\mathbf{w}_k\right),$$

$$\mathbb{E}_{\hat{\xi}_k}\left[\hat{f}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \hat{\xi}_k)\right] = F\left(\mathbf{w}_k + \mu_k\mathbf{z}_k\right).$$

From (4.0.11), for any iteration k, we have

$$\hat{f}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \hat{\xi}_k) \leq \hat{f}(\mathbf{w}_k, \hat{\xi}_k). \quad (4.1.4)$$

$$F\left(\mathbf{w}_k + \mu_k\mathbf{z}_k\right) = \mathbb{E}\left[\hat{f}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \hat{\xi}_k)\right] \leq \mathbb{E}\left[\hat{f}(\mathbf{w}_k, \hat{\xi}_k)\right] = F\left(\mathbf{w}_k\right). \quad (4.1.5)$$

We will not specify the computation of $\mathbf{H}_k$ and β_k. Instead, we will propose an updating scheme for $\mathbf{H}_k$ and β_k that satisfies both Assumption 4.1.3 and Assumption 4.1.4, with $\beta_k \in (0, 1)$ and $\beta_k \leq \beta_{k-1}$ for all k. The following section examines the convergence and complexity of DRES-NAQ, considering the assumptions outlined earlier.

In this subsection, we analyze the convergence and complexity of DRES-NAQ under the condition that the step size α_k diminishes. Our analysis is informed by [95], which explores the convergence and complexity of a general framework for stochastic quasi-Newton methods in nonconvex optimization. We assume α_k satisfies the following condition:

$$\sum_{k=1}^{+\infty} \alpha_k = +\infty, \quad \sum_{k=1}^{+\infty} \alpha_k^2 < +\infty, \quad (4.1.6)$$

which is a standard assumption in [91, 95, 98]. A simple choice for α_k that satisfies (4.1.6) is $\alpha_k = O(1/k)$. The following lemma establishes that DRES-NAQ exhibits a descent property regarding the expected objective value.

Lemma 4.1.6. *Suppose that $\{\boldsymbol{w}_k\}$ is generated by DRES-NAQ and Assumptions 4.1.1-4.1.5 hold. Further assume that (4.1.6) holds, and $\alpha_k \leq \frac{\underline{\kappa}}{L(\bar{\kappa}+\Gamma)^2}$ for all k. (Note that this can be satisfied if α_k is nonincreasing and the initial step size $\alpha_1 \leq \frac{\underline{\kappa}}{L(\bar{\kappa}+\Gamma)^2}$.) Then the following inequality holds:*

$$\mathbb{E}\left[F\left(\boldsymbol{w}_{k+1}+\mu_{k+1}\boldsymbol{z}_{k+1}\right) \mid \boldsymbol{w}_k+\mu_k\boldsymbol{z}_k\right] \leq F\left(\boldsymbol{w}_k+\mu_k\boldsymbol{z}_k\right) - \frac{1}{2}\alpha_k\underline{\kappa}\left\|\nabla F\left(\boldsymbol{w}_k+\mu_k\boldsymbol{z}_k\right)\right\|^2 + \frac{L\sigma^2(\bar{\kappa}+\Gamma)^2}{2m_k}\alpha_k^2, \quad \forall k \geq 1, \tag{4.1.7}$$

where the conditional expectation is taken with respect to ξ_k.

Proof. Define $\delta_k = \mathbf{g}(\mathbf{w}_k+\mu_k\mathbf{z}_k, \xi_k) - \nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right)$. Based on Algorithm 4, along with Assumptions 4.1.1 and 4.1.3, we obtain the following results.

$$\begin{aligned}
F\left(\mathbf{w}_{k+1}\right) &\leq F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right) + \left\langle \nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right), \mathbf{w}_{k+1}-\mathbf{w}_k+\mu_k\mathbf{z}_k\right\rangle \\
&\quad + \frac{L}{2}\left\|\mathbf{w}_{k+1}-\mathbf{w}_k+\mu_k\mathbf{z}_k\right\|^2 \\
&= F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right) - \alpha_k\left\langle \nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right), (\mathbf{H}_k+\Gamma\mathbf{I})\mathbf{g}(\mathbf{w}_k+\mu_k\mathbf{z}_k, \xi_k)\right\rangle \\
&\quad + \frac{L}{2}\alpha_k^2\left\|(\mathbf{H}_k+\Gamma\mathbf{I})\mathbf{g}(\mathbf{w}_k+\mu_k\mathbf{z}_k, \xi_k)\right\|^2 \\
&\leq F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right) - \alpha_k\left\langle \nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right), (\mathbf{H}_k+\Gamma\mathbf{I})\nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right)\right\rangle \\
&\quad - \alpha_k\left\langle \nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right), (\mathbf{H}_k+\Gamma\mathbf{I})\delta_k\right\rangle \\
&\quad + \frac{L}{2}\alpha_k^2(\bar{\kappa}+\Gamma)^2\left\|\mathbf{g}(\mathbf{w}_k+\mu_k\mathbf{z}_k, \xi_k)\right\|^2.
\end{aligned} \tag{4.1.8}$$

Taking expectation with respect to ξ_k on both sides of (4.1.8), conditioned on $\mathbf{w}_k+\mu_k\mathbf{z}_k$, we obtain

$$\begin{aligned}
\mathbb{E}\left[F\left(\mathbf{w}_{k+1}\right) \mid \mathbf{w}_k+\mu_k\mathbf{z}_k\right] &\leq F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right) && (4.1.9)\\
&\quad - \alpha_k\left\langle \nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right), (\mathbf{H}_k+\Gamma\mathbf{I})\nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right)\right\rangle \\
&\quad + \frac{L}{2}\alpha_k^2(\bar{\kappa}+\Gamma)^2\mathbb{E}\left[\left\|\mathbf{g}(\mathbf{w}_k+\mu_k\mathbf{z}_k, \xi_k)\right\|^2 \mid \mathbf{w}_k+\mu_k\mathbf{z}_k\right], && (4.1.10)
\end{aligned}$$

where we used (4.1.3) and the fact that $\mathbb{E}[\delta_k \mid \mathbf{w}_k] = 0$. From (4.1.2) and the condition $\mathbb{E}[\delta_k \mid \mathbf{w}_k] = 0$, it follows that

$$\begin{aligned}
&\mathbb{E}\left[\|\mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k)\|^2 \mid \mathbf{w}_k + \mu_k\mathbf{z}_k\right] \\
&\quad = \mathbb{E}\left[\|\mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k) - \nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k) + \nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\|^2 \mid \mathbf{w}_k + \mu_k\mathbf{z}_k\right] \\
&\quad = \mathbb{E}\left[\|\nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\|^2 \mid \mathbf{w}_k + \mu_k\mathbf{z}_k\right] + \mathbb{E}\left[\|\mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k) - \nabla F(\mathbf{w}_k)\|^2 \mid \mathbf{w}_k + \mu_k\mathbf{z}_k\right] \\
&\qquad + 2\mathbb{E}\left[\langle \delta_k, \nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\rangle \mid \mathbf{w}_k + \mu_k\mathbf{z}_k\right] \\
&\quad = \|\nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\|^2 + \mathbb{E}\left[\|\mathbf{g}(\mathbf{w}_k + \mu_k\mathbf{z}_k, \xi_k) - \nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\|^2 \mid \mathbf{w}_k + \mu_k\mathbf{z}_k\right] \\
&\quad \leq \|\nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\|^2 + \sigma^2/m_k,
\end{aligned}$$

which together with (4.1.9) and Assumption 4.1.3 yields that

$$\mathbb{E}[F(\mathbf{w}_{k+1}) \mid \mathbf{w}_k + \mu_k\mathbf{z}_k] \leq F(\mathbf{w}_k + \mu_k\mathbf{z}_k) \tag{4.1.11}$$

$$- \left(\alpha_k\underline{\kappa} - \frac{L}{2}\alpha_k^2(\bar{\kappa} + \Gamma)^2\right)\|\nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\|^2 + \frac{L\sigma^2(\bar{\kappa} + \Gamma)^2}{2m_k}\alpha_k^2. \tag{4.1.12}$$

From Assumption 4.1.5 and (4.1.4), we have that

$$\mathbb{E}[F(\mathbf{w}_{k+1} + \mu_{k+1}\mathbf{z}_{k+1}) \mid \mathbf{w}_k + \mu_k\mathbf{z}_k] \leq \mathbb{E}[F(\mathbf{w}_{k+1}) \mid \mathbf{w}_k + \mu_k\mathbf{z}_k]. \tag{4.1.13}$$

Finally, using (4.1.13) and (4.1.11), we obtain that

$$\begin{aligned}
&\mathbb{E}[F(\mathbf{w}_{k+1} + \mu_{k+1}\mathbf{z}_{k+1}) \mid \mathbf{w}_k + \mu_k\mathbf{z}_k] \\
&\leq F(\mathbf{w}_k + \mu_k\mathbf{z}_k) - \left(\alpha_k\underline{\kappa} - \frac{L}{2}\alpha_k^2(\bar{\kappa} + \Gamma)^2\right)\|\nabla F(\mathbf{w}_k + \mu_k\mathbf{z}_k)\|^2 + \frac{L\sigma^2(\bar{\kappa} + \Gamma)^2}{2m_k}\alpha_k^2.
\end{aligned} \tag{4.1.14}$$

Thus, by combining (4.1.14) with the assumption $\alpha_k \leq \frac{\underline{\kappa}}{L(\bar{\kappa}+\Gamma)^2}$, we obtain that inequality (4.1.7) holds. □

We are now ready to give convergence results for DRES-NAQ (Algorithm 4).

Theorem 4.1.7. *Suppose that Assumptions 4.1.1-4.1.5 hold for $\{\boldsymbol{w}_k\}$ generated by DRES-NAQ with batch size $m_k = m$ for all k. If the step size α_k satisfies (4.1.6) and $\alpha_k \leq \frac{\underline{\kappa}}{L(\bar{\kappa}+\Gamma)^2}$ for all k, then*

$$\liminf_{k\to\infty} \|\nabla F(\boldsymbol{w}_k + \mu_k\boldsymbol{z}_k)\| = 0 \quad \textit{with probability } 1. \tag{2.14}$$

Moreover, there exists a positive constant M_F such that

$$\mathbb{E}\left[F\left(\boldsymbol{w}_k+\mu_k \boldsymbol{z}_k\right)\right] \leq M_F, \quad \forall k. \tag{2.15}$$

Proof. Define $\beta_k := \frac{\alpha_k \underline{\kappa}}{2}\left\|\nabla F\left(\mathbf{w}_k+\mu_k \mathbf{z}_k\right)\right\|^2$ and $\gamma_k := F\left(\mathbf{w}_k+\mu_k \mathbf{z}_k\right)+\frac{L\sigma^2(\bar{\kappa}+\Gamma)^2}{2m}\sum_{i=k}^{\infty}\alpha_i^2$. Let $\mathcal{F}_k$ be the σ-algebra measuring β_k, γ_k, and $\mathbf{w}_k+\mu_k \mathbf{z}_k$. From (4.1.7) we know that for any k,

$$\begin{aligned}
\mathbb{E}\left[\gamma_{k+1} \mid \mathcal{F}_k\right] &= \mathbb{E}\left[F\left(\mathbf{w}_{k+1}+\mu_{k+1} \mathbf{z}_{k+1}\right) \mid \mathcal{F}_k\right]+\frac{L\sigma^2(\bar{\kappa}+\Gamma)^2}{2m}\sum_{i=k+1}^{\infty}\alpha_i^2 \\
&\leq F\left(\mathbf{w}_k+\mu_k \mathbf{z}_k\right)-\frac{\alpha_k \underline{\kappa}}{2}\left\|\nabla F\left(\mathbf{w}_k+\mu_k \mathbf{z}_k\right)\right\|^2+\frac{L\sigma^2(\bar{\kappa}+\Gamma)^2}{2m}\sum_{i=k}^{\infty}\alpha_i^2 \\
&= \gamma_k-\beta_k,
\end{aligned} \tag{4.1.15}$$

which implies that $\mathbb{E}\left[\gamma_{k+1}-f^{\text{low}} \mid \mathcal{F}_k\right] \leq \gamma_k-f^{\text{low}}-\beta_k$. Since $\beta_k \geq 0$, it follows that $0 \leq \mathbb{E}\left[\gamma_k-f^{\text{low}}\right] \leq \gamma_1-f^{\text{low}}<+\infty$, which implies the bound in (2.15). According to Definition 2.2.10, $\left\{\gamma_k-f^{low}\right\}$ is a supermartingale. Consequently, Proposition 2.2.11 establishes the existence of a limit γ such that $\lim_{k\to\infty}\gamma_k=\gamma$ with probability 1, and ensures that $\mathbb{E}[\gamma] \leq \mathbb{E}[\gamma_1]$. Note that from (4.1.15), it follows that $\mathbb{E}[\beta_k] \leq \mathbb{E}[\gamma_k]-\mathbb{E}[\gamma_{k+1}]$. Therefore,

$$\mathbb{E}\left[\sum_{k=1}^{\infty}\beta_k\right] \leq \sum_{k=1}^{\infty}\left(\mathbb{E}\left[\gamma_k\right]-\mathbb{E}\left[\gamma_{k+1}\right]\right)<+\infty,$$

which further yields that

$$\sum_{k=1}^{\infty}\beta_k=\frac{\underline{\kappa}}{2}\sum_{k=1}^{\infty}\alpha_k\left\|\nabla F\left(\mathbf{w}_k+\mu_k \mathbf{z}_k\right)\right\|^2<+\infty \quad \text{with probability 1.} \tag{2.17}$$

Since $\sum_{k=1}^{\infty}\alpha_k=+\infty$, it follows that (2.14) holds. □

Subsequently, we are able to analyze the iteration complexity of DRES-NAQ, for a specifically selected step size α_k.

Theorem 4.1.8. *Suppose that Assumptions 4.1.1-4.1.5 hold for $\{\boldsymbol{w}_k\}$ generated by DRES-NAQ with batch size $m_k=m$ for all k. We also assume that the step size α_k is specifically chosen as*

$$\alpha_k=\frac{\underline{\kappa}}{L(\bar{\kappa}+\Gamma)^2}k^{-\beta}, \tag{2.22}$$

where $\beta \in (0.5, 1)$. *Note that this choice satisfies (4.1.6) and* $\alpha_k \leq \frac{\underline{\kappa}}{L(\bar{\kappa}+\Gamma)^2}$ *for all* k. *Then*

$$\frac{1}{N}\sum_{k=1}^{N}\mathbb{E}\left[\|\nabla F\left(\boldsymbol{w}_k+\mu_k\boldsymbol{z}_k\right)\|^2\right] \leq \frac{2L\left(M_F-F^{low}\right)\left(\bar{\kappa}+\Gamma\right)^2}{\underline{\kappa}^2}N^{\beta-1}+\frac{\sigma^2}{(1-\beta)m}\left(N^{-\beta}-N^{-1}\right), \tag{4.1.16}$$

where N *denotes the iteration number. Moreover, for a given* $\epsilon \in (0,1)$, *to guarantee that* $\frac{1}{N}\sum_{k=1}^{N}\mathbb{E}\left[\|\nabla F\left(\boldsymbol{w}_k+\mu_k\boldsymbol{z}_k\right)\|^2\right] < \epsilon$, *the number of iterations* N *needed is at most* $O(\epsilon^{-\frac{1}{1-\beta}})$.

Proof. Taking expectation on both sides of (4.1.7) and summing over $k = 1, \ldots, N$ yields

$$\begin{aligned}
&\frac{1}{2}\underline{\kappa}\sum_{k=1}^{N}\mathbb{E}\left[\|\nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right)\|^2\right] \\
&\leq \sum_{k=1}^{N}\frac{1}{\alpha_k}\left(\mathbb{E}\left[F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right)\right]-\mathbb{E}\left[F\left(\mathbf{w}_{k+1}+\mu_{k+1}\mathbf{z}_{k+1}\right)\right]\right)+\frac{L\sigma^2\bar{\kappa}^2}{2m}\sum_{k=1}^{N}\alpha_k \\
&= \frac{1}{\alpha_1}F\left(\mathbf{w}_1+\mu_1\mathbf{z}_1\right)+\sum_{k=2}^{N}\left(\frac{1}{\alpha_k}-\frac{1}{\alpha_{k-1}}\right)\mathbb{E}\left[F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right)\right] \\
&\quad -\frac{\mathbb{E}\left[F\left(\mathbf{w}_{N+1}+\mu_{N+1}\mathbf{z}_{N+1}\right)\right]}{\alpha_N}+\frac{L\sigma^2(\bar{\kappa}+\Gamma)^2}{2m}\sum_{k=1}^{N}\alpha_k \\
&\leq \frac{M_F}{\alpha_1}+M_F\sum_{k=2}^{N}\left(\frac{1}{\alpha_k}-\frac{1}{\alpha_{k-1}}\right)-\frac{F^{\text{low}}}{\alpha_N}+\frac{L\sigma^2(\bar{\kappa}+\Gamma)^2}{2m}\sum_{k=1}^{N}\alpha_k \\
&= \frac{M_F-F^{\text{low}}}{\alpha_N}+\frac{L\sigma^2(\bar{\kappa}+\Gamma)^2}{2m}\sum_{k=1}^{N}\alpha_k \\
&\leq \frac{L\left(M_F-F^{\text{low}}\right)\left(\bar{\kappa}+\Gamma\right)^2}{\underline{\kappa}}N^{\beta}+\frac{\sigma^2\underline{\kappa}}{2(1-\beta)m}\left(N^{1-\beta}-1\right),
\end{aligned}$$

which results in (4.1.16), where the second inequality is due to (2.15) and the last inequality is due to (2.22). Then for a given $\epsilon > 0$, to guarantee that $\frac{1}{N}\sum_{k=1}^{N}\mathbb{E}\left[\|\nabla F\left(\mathbf{w}_k+\mu_k\mathbf{z}_k\right)\|^2\right] \leq \epsilon$, it suffices to require that

$$\frac{2L\left(M_F-F^{\text{low}}\right)\left(\bar{\kappa}+\Gamma\right)^2}{\underline{\kappa}^2}N^{\beta-1}+\frac{\sigma^2}{(1-\beta)m}\left(N^{-\beta}-N^{-1}\right) < \epsilon.$$

Since $\beta \in (0.5, 1)$, it follows that the number of iterations N needed is at most $O\left(\epsilon^{-\frac{1}{1-\beta}}\right)$. □

Remark 4.1.9. Note that Theorem 4.1.8 also provides iteration complexity analysis for the classic SGD method, which can be regarded as a special case of DRESNAQ with $H_k = I, \mu_k = 0$, for all k and $\Gamma = \delta = 0$.

In the next section, we will show experimental results to compare the performance of DRESNAQ with other methods.

4.2 Numerical experiments

Numerical simulations were performed on various datasets to assess the effectiveness of the proposed DRES-NAQ methods. The simulations were conducted on a MacBook Air with an Apple M1 chip, featuring an 8-core CPU (3.2 GHz) and 8 GB of memory. This section provides an empirical analysis of the performance of DRES-NAQ in comparison to SGD and RES-NAQ for both convex and non-convex problems, using the support vector machine (SVM) framework. First, we compare the DRES-NAQ method with SGD and RES-NAQ in solving the following convex SVM problem with a squared hinge loss function (see [100]):

$$\min_{\mathbf{w}\in\mathbb{R}^n} F(\mathbf{w}) := \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \max(0, 1 - y_i(\mathbf{w}^\top x_i))^2, \qquad \textbf{(convex)}$$

where $C > 0$ is a constant parameter, x_i represents the training data, and $y_i \in \{1, -1\}$ denotes the corresponding label. Additionally, we also compare DRES-NAQ with SGD and RES-NAQ on a non-convex SVM problem involving a smooth truncated squared hinge loss function (see [101]):

$$\min_{\mathbf{w}\in\mathbb{R}^n} F(\mathbf{w}) := \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} a(1-\exp(-\frac{1}{a}\max(0, 1-y_i(\mathbf{w}^\top x_i))^2)), \quad \textbf{(nonconvex)}$$

where $a > 0$ is a constant parameter.

The Python code used in this work requires setting up the necessary modules and is based on the BaseEstimator class, which includes self in function inputs. If you plan to use the code, we recommend reviewing it. The implementation was done using the following environment: pandas 1.3.2, numpy 1.20.3, matplotlib 3.5.3, scikit-learn 0.24.2, scipy 1.6.2. The following code defines functions for the gradient

and cost of the squared hinge loss function and the smooth truncated squared hinge loss function, respectively.

```
    '''gradient of squareHingeloss '''
    def gradient(self, w, x, t):
        u = np.maximum(0, 1-((t[:, np.newaxis]*x).dot(w)))
        p = -2/len(t)*u.dot(t[:,np.newaxis]*x)
        grad = self.C * w + p
        return u,p,grad

    ''' cost function of squareHingeloss '''
    def cost_function(self,w,x,t):
        loss = 1/len(t) *  (np.sum((np.maximum(0, 1-((t[:,
            ↪ np.newaxis]*x).dot(w)))**2)))
        cost = self.C/2 * np.linalg.norm(w)**2  + loss
        return loss, cost

    '''gradient of smooth truncated squared hinge loss'''
    def gradient(self, w, x, t):
        a = self.a
        u = -a*np.exp(-(1-((t[:, np.newaxis]*x).dot(w)))**2) *
            ↪ (-(1-((t[:, np.newaxis]*x).dot(w))))
        p = -2*(1/a)/len(t)*u.dot(t[:,np.newaxis]*x)
        grad = self.C * w + p
        return u,p,grad

    ''' cost function of smooth truncated squared hinge loss'''
    def cost_function(self,w,x,t):
        a = self.a
        loss = 1/len(t) *  np.sum(a*(1-np.exp(-(1/a)*(1-((t[:,
            ↪ np.newaxis]*x).dot(w)))**2)))
        cost = self.C/2 * np.linalg.norm(w)**2  + loss
        return loss, cost
```

The following Python code implements the DRES-NAQ method in this experiment.

```

class DResNaq():
    def __init__(self, C=1, delta=0.0001, gamma=0.0001,
```

```
                epsilon_0=0.0001, tau_0=0.0001,
                mu=0.99, max_epochs=20000, n_batches=1000,
                    ↪ time=100, batches2=100, a=1):
        assert C > 0
        self.C = C
        assert epsilon_0 > 0
        self.epsilon_0 = epsilon_0
        assert tau_0 > 0
        self.tau_0 = tau_0
        assert delta > 0
        self.delta = delta
        assert gamma > 0
        self.gamma = gamma
        assert 0 <= mu < 1
        self.mu = mu
        assert max_epochs > 0 and isinstance(max_epochs, int)
        self.max_epochs = max_epochs
        assert n_batches > 0 and isinstance(n_batches, int)
        self.n_batches = n_batches
        assert time > 0
        self.time = time
        self.batches2 = batches2
        self.a =a

    def Update_DRES_BFGS(self,B, dw, dg, I):
        dg_t =  dg[:, np.newaxis]          Bdw = np.dot(B, dw)
        Bdw = np.dot(B, dw)
        dw_t_B = np.dot(dw, B)
        dwBdw = np.dot(np.dot(dw, B), dw)
        p = dg_t*dg
        u = Bdw[:, np.newaxis] * dw_t_B
        B_new = B + p / np.dot(dg, dw) - u / dwBdw + self.delta * I
        return p, u, B_new

    def fit(self, x, t):
        k = 0
        j = 0
        w = np.zeros(len(x[0]))
        z = np.zeros(len(x[0]))
        a = 0
```

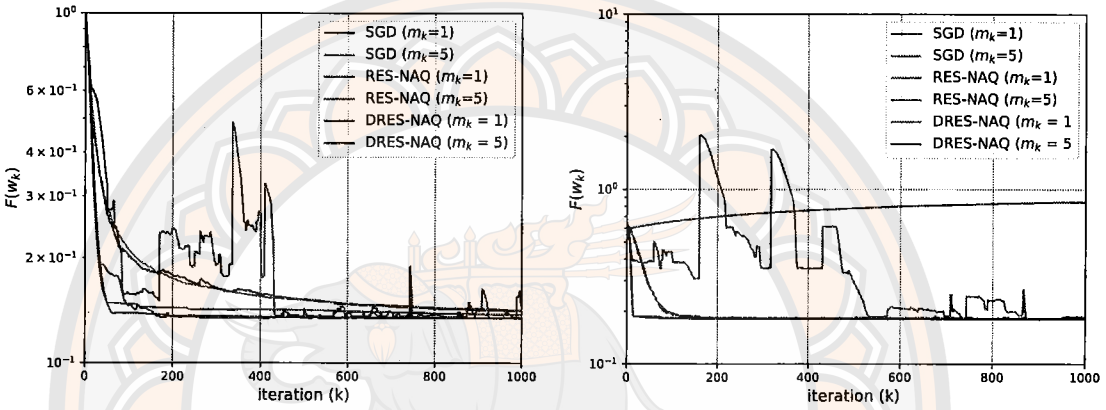
```
43          L=0
44          obj_dresnaq = []
45          objnat_dresnaq = []
46          iter_dresnaq = []
47          mu_ = []
48          B = np.identity(len(x[0]))  #inverse hession
49          I = np.identity(len(x[0]))  #identity
50
51          for epoch in range(self.max_epochs):
52              np.random.seed(42)
53              idx = np.random.permutation(len(t))
54              for i in range(len(t)):
55
56                  r = idx[i*self.n_batches:(i+1)*self.n_batches]
57                  if r.size==0: break
58                  r1 = idx[a * self.batches2:(a + 1) * self.batches2]
59                  if r1.size == 0:
60                      a = 0
61                      r1 = idx[a * self.batches2:(a + 1) *
                            ↪ self.batches2]
62                  a = a + 1
63                  k = k + 1
64                  iter_dresnaq.append(k)
65                  mu_t = 0.99 * np.exp(-0.0001 * k)
66                  _, cost = self.cost_function(w, x[r1, :], t[r1])
67                  _, cost2 = self.cost_function(w, x, t)
68                  obj_dresnaq.append(cost2)
69                  _, cost_naq = self.cost_function(w + (mu_t * z),
                        ↪ x[r1, :], t[r1])
70                  _, cost_naq2 = self.cost_function(w + (mu_t * z),
                        ↪ x, t)
71                  objnat_dresnaq.append(cost_naq2)
72                  if cost < cost_naq or cost == cost_naq:
73                      mu_t = 0
74                      mu_.append(mu_t)
75                      _, _, g_naq = self.gradient(w + (mu_t * z),
                            ↪ x[r, :], t[r])
76                      H = np.linalg.inv(B)
77                      H_RES = H + self.gamma * I
78                      eta = (self.epsilon_0 * self.tau_0) /
```

```
                ↪ (self.tau_0 + k)
79              z_new = mu_t * z - eta * np.matmul(H_RES,
                ↪ g_naq)
80              w_new = w + z_new
81              _, _, g_new = self.gradient(w_new, x[r, :],
                ↪ t[r])
82              dw = w_new - (w + mu_t * z)
83              dg_old = g_new - g_naq
84              A = B+self.delta * I
85              if np.dot(dg_old, dw) < 0.2 *
                ↪ np.dot(np.dot(dw, A), dw)+ (self.delta*
                ↪ np.dot(dw, dw)):
86                  theta = (0.8 * np.dot(np.dot(dw, A), dw)+
                    ↪ (self.delta*np.dot(dw, dw))) //
                    ↪ np.dot(np.dot(dw, A), dw)+
                    ↪ (np.dot(dw,dw))
87              else:
88                  theta = 1
89              dg = theta*dg_old + (1-theta)*np.dot(A,dw) -
                ↪ (self.delta* dw)
90              _, _, B_new = self.Update_DRES_BFGS(B, dw, dg,
                ↪ I)
91              B = B_new
92              w = w_new
93              z = z_new
94          else:
95              j = j+1
96              mu_.append(mu_t)
97              _, _, g_naq = self.gradient(w + (mu_t * z),
                ↪ x[r, :], t[r])
98              H = np.linalg.inv(B)
99              H_RES = H + self.gamma * I
100             eta = (self.epsilon_0 * self.tau_0) /
                ↪ (self.tau_0 + k)
101             z_new = mu_t * z - eta * np.matmul(H_RES,
                ↪ g_naq)
102             w_new = w + z_new
103             _, _, g_new = self.gradient(w_new, x[r, :],
                ↪ t[r])
104             dw = w_new - (w + mu_t * z)
```

```
105                     dg_old = g_new - g_naq
106                     A = B+self.delta * I
107                     if np.dot(dg_old, dw) < 0.2 *
                            ↪ np.dot(np.dot(dw, A), dw)+ (self.delta*
                            ↪ np.dot(dw, dw)):
108                         theta = (0.8 * np.dot(np.dot(dw, A), dw)+
                                ↪ (self.delta* np.dot(dw, dw))) //
                                ↪ np.dot(np.dot(dw, A), dw)+
                                ↪ (np.dot(dw, dw))
109                     else:
110                         theta = 1
111                     dg = theta*dg_old + (1-theta)*np.dot(A,dw) -
                            ↪ (self.delta* dw)
112                     _, _, B_new = self.Update_DRES_BFGS(B, dw, dg,
                            ↪ I)
113                     B = B_new
114                     w = w_new
115                     z = z_new
116
117         self.final_iter = k
118         self._coef = w
119         self.obj_dresnaq = obj_dresnaq
120         self.iter_dresnaq = iter_dresnaq
121         self.objnat_dresnaq = objnat_dresnaq
122         self.mu_ = mu_
123         return self
124
125     def predict(self, x):
126         p = np.sign(np.matmul(x, self._coef))
127         p[p == 0] = 1
128         return p.astype(int)
```

The methods was evaluated on synthetic datasets. We evaluated this approach using a two-dimensional scenario where samples are drawn equally from two Gaussian distributions: x_i, for $i \in \{i : y_i = 1\}$, is sampled from $\mathcal{N}(\mu_{(1)}, \Sigma_{(1)})$, and x_i, for $i \in \{i : y_i = -1\}$, is sampled from $\mathcal{N}(\mu_{(-1)}, \Sigma_{(-1)})$. Specifically, $\mu_{(1)} = [1.5, -3]^\top$, $\mu_{(-1)} = [-1.5, 3]^\top$, and $\Sigma_{(1)} = \Sigma_{(-1)} = [0.2\ 3; 0\ 3]$. We generated training datasets, each containing 500 samples, resulting in a total of $m = 1000$

samples. The SVM parameter settings were $C = 10^{-2}$ and $a = 1$. For each iteration k, the parameter settings for SGD, RES-NAQ, and DRES-NAQ were as follows: the step size $\alpha_k = 10^5/(10^8 + k)$, batch sizes $m_k = 1$ and $m_k = 5$, and for RES-NAQ and DRES-NAQ, $\delta = 10^{-3}$ and $\Gamma = 10^{-3}$. RES-NAQ was configured with $\mu = 0.99$, while DRES-NAQ was configured with the momentum coefficient $\beta_k = 0.99\exp(-(10^{-4})k)$, and $\gamma = 10^{-3}$ and a batch size $L_k = 15$. The stopping criterion was $\|\nabla F(\mathbf{w}_k)\| < \epsilon$, where $\epsilon = 0.25$.



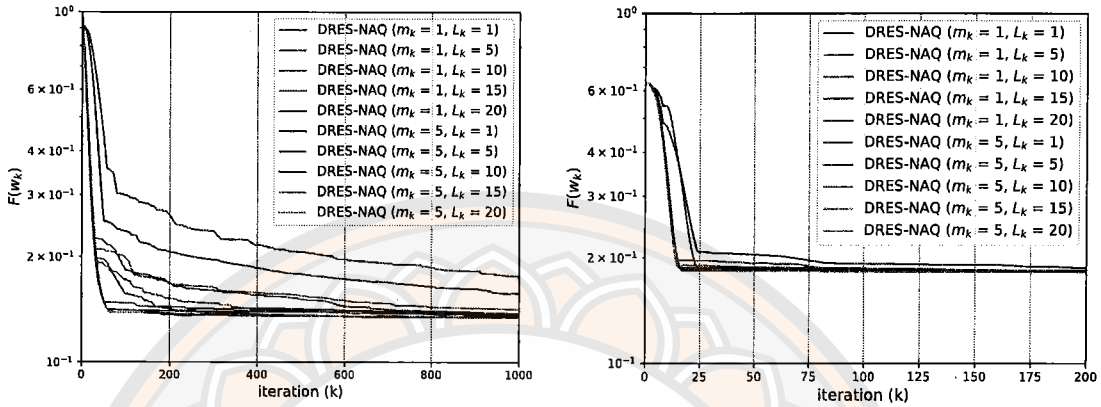
(a) Convex problem **(b)** Nonconvex problem

Figure 21 Comparison of SGD, RES-NAQ, and DRES-NAQ for varying batch sizes.

In Figure 21, we observe that DRES-NAQ consistently outperforms both SGD and RES-NAQ in terms of convergence speed and robustness for convex and nonconvex problems. First, considering the convex problems, as illustrated in Figure 21a, it is evident that while SGD remains stable, its convergence rate is slower compared to RES-NAQ. Although RES-NAQ initially converges more rapidly, it eventually encounters oscillations. In contrast, DRES-NAQ improves both the convergence speed and stability.

For nonconvex problems, as shown in Figure 21b, DRES-NAQ continues to exhibit both stability and accelerated convergence. In this case, although SGD is stable, it converges slowly. RES-NAQ, with a batch size of $m_k = 1$, also oscillates but ultimately converges and reduces the objective function. However, with $m_k = 5$, RES-NAQ fails to converge, as it is not suited for nonconvex optimization, and

the choice of $m_k = 5$ adversely affects performance. Therefore, while increasing the batch size m_k can often enhance convergence speed, careful selection of m_k is essential to avoid negative outcomes.



(a) Convex problem **(b)** Nonconvex problem

Figure 22 Performance of DRES-NAQ with varying batch sizes and values of L_k.

In Figure 22, we observe that increasing the parameter L_k, which controls the number of data samples used to compute the objective function in μ_k, can enhance the performance of DRES-NAQ. However, this also increases the computational overhead. Nevertheless, DRES-NAQ still requires fewer computational resources for μ_k calculations compared to RES-NAQ.

Table 29 Number of iterations and time utilized to achieve the stopping criterion (convex problem).

Batch size	Methods	Iterations	Time
$m_k = 1$	SGD	432	0.05306
	RES-NAQ	102	0.01107
	DRES-NAQ	**49**	**0.00537**
$m_k = 5$	SGD	362	0.02558
	RES-NAQ	90	0.00919
	DRES-NAQ	**45**	**0.00638**

Table 30 Number of iterations and time utilized to achieve the stopping criterion (nonconvex problem).

Batch size	Methods	Iterations	Time
$m_k = 1$	SGD	125	0.02752
	RES-NAQ	160	0.02815
	DRES-NAQ	**75**	**0.01174**
$m_k = 5$	SGD	163	0.02457
	RES-NAQ	-	-
	DRES-NAQ	**37**	**0.00642**

In Table 29, DRES-NAQ consistently surpasses both SGD and RES-NAQ in terms of the number of iterations and time to convergence. Increasing the batch size significantly decreases the iterations and time required for all methods. With a batch size of 5, DRES-NAQ performs optimally, needing only 45 iterations and 0.00638 seconds to converge. Similarly, in Table 30, DRES-NAQ again outperforms SGD and RES-NAQ. RES-NAQ does not converge when the batch size is set to 5. However, DRES-NAQ achieves the best results with a batch size of 5, requiring only 37 iterations and 0.00642 seconds to converge.

On the basis of experiments with synthetic datasets, DRES-NAQ proved to be a more effective optimization method than both SGD and RES-NAQ. By carefully tuning the batch size and the parameters μ_k and L_k. However, there is a trade-off between computational efficiency and convergence speed, as larger batch sizes demand more computational resources, but can lead to faster convergence.

CHAPTER V

CONCLUSION

This thesis introduces three advanced techniques aimed at enhancing classification performance and optimization efficiency in support vector machines (SVMs) and stochastic optimization.

First, the asymmetric truncated generalized pinball loss ($L_{tgp}^{\alpha_1,\alpha_2}$) was developed to improve robustness and flexibility in parameter tuning. Integrated into the ATGP-SVM framework, this loss function ensures insensitivity to noise and outliers while maintaining sparsity. Due to its non-convexity, a difference of convex (DC) programming approach was employed to efficiently optimize ATGP-SVM, ensuring finite convergence. Experimental evaluations on synthetic and UCI benchmark datasets demonstrated that ATGP-SVM consistently outperforms existing methods, particularly in noisy environments.

Second, the smooth rescaled generalized pinball loss (SRGP-SVM) was proposed to further address challenges related to noise sensitivity, parameter flexibility, and outlier robustness. By combining the advantages of rescaling with the differentiability of smooth generalized pinball loss, SRGP-SVM enhances classification performance while maintaining adaptability across various datasets. Experimental results confirm its strong accuracy, MCC, and F_1-score compared to alternative approaches, albeit with a trade-off in computational efficiency. Future work will explore refined parameter selection strategies and the application of more efficient optimization techniques to extend its scalability.

Finally, a regularized stochastic BFGS method was introduced for nonconvex optimization, incorporating Nesterov acceleration and an adaptive momentum coefficient. By dynamically adjusting momentum based on the objective function's value from selected dataset samples, this method mitigates overshooting issues while preserving computational efficiency. We theoretically analyze its convergence and complexity, and numerical results demonstrated its superiority over existing techniques, such as SGD and RES-NAQ, in convex and nonconvex SVM-based classification problems. Further research will explore extensions to more complex machine

learning models and large-scale optimization challenges.

In summary, this thesis presents novel loss functions and optimization strategies that significantly enhance the robustness, adaptability, and computational efficiency of SVM classification and nonconvex stochastic optimization. Future directions include refining parameter selection methods, expanding applicability to broader machine learning models, and integrating more efficient optimization algorithms to address large-scale problems.

REFERENCES

REFERENCES

1. Mohammed N, Zhou W, Bahrani B, Hill DJ. Support vector machines for predicting the impedance model of inverter-based resources. IEEE Transactions on Power Systems. 2024.

2. Damminsed V, Panup W, Wangkeeree R. Laplacian twin support vector machine with pinball loss for semi-supervised classification. IEEE Access. 2023;11:31399-416.

3. Ratiphaphongthon W, Panup W, Wangkeeree R. An improved technique for pneumonia infected patients image recognition based on combination algorithm of smooth generalized pinball SVM and variational autoencoders. IEEE Access. 2022;10:107431-45.

4. Ganapathiraju A, Hamaker JE, Picone J. Applications of support vector machines to speech recognition. IEEE transactions on signal processing. 2004;52(8):2348-55.

5. Pattanayak RM, Behera HS, Panigrahi S. A novel high order hesitant fuzzy time series forecasting by using mean aggregated membership value with support vector machine. Information Sciences. 2023;626:494-523.

6. Sheykhmousa M, Mahdianpari M, Ghanbari H, Mohammadimanesh F, Ghamisi P, Homayouni S. Support vector machine versus random forest for remote sensing image classification: A metaanalysis and systematic review. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing. 2020;13:6308-25.

7. Huang FJ, LeCun Y. Large-scale learning with svm and convolutional for generic object categorization. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 1. IEEE; 2006. p. 284-91.

8. Benítez-Peña S, Blanquero R, Carrizosa E, Ramírez-Cobo P. Costsensitive probabilistic predictions for support vector machines. European Journal of Operational Research. 2024;314(1):268-79.

9. DangW, Kim S, Park S, XuW. The impact of economic and IoT technologies on air pollution: an AI-based simulation equation model using support vector machines. Soft Computing. 2024;28(4):3591- 611.

10. Rehan I, Khan S, Gondal M, Abbas Q, Ullah R. Non-invasive diabetes mellitus diagnostics using laser-induced breakdown spectroscopy and support vector machine algorithm. Arabian Journal for Science and Engineering. 2024;49(1):1257-65.

11. Wang H, Shao Y. Fast generalized ramp loss support vector machine for pattern classification. Pattern Recognition. 2024;146:109987.

12. Wang Y, Baek J, Tang Y, Du J, Hillman M, Chen JS. Support vector machine guided reproducing kernel particle method for imagebased modeling of microstructures. Computational Mechanics. 2024;73(4):907-42.

13. Silva NF, de Andrade LP, da Silva WS, de Melo MK, Tonelli AO. Portfolio optimization based on the pre-selection of stocks by the Support Vector Machine model. Finance Research Letters. 2024;61:105014.

14. Lou C, Xie X. Multi-view universum support vector machines with insensitive pinball loss. Expert Systems with Applications. 2024;248:123480. Available from: https://www.sciencedirect.com/science/article/pii/S0957417424003452.

15. Amma NB, Rajput V. Towards improving the performance of traffic sign recognition using support vector machine-based deep learning model. Multimed Tools Appl. 2024;83(3):6579–600.

16. Ahmed N, Rabbi S, Rahman T, Mia R, Rahman M. Traffic sign detection and recognition model using support vector machine and histogram of oriented gradient. Int J Inf Technol Comput Sci. 2021;13(3):61–73.

17. Adige S, Kurban R, Durmuş A, Karaköse E. Classification of apple images using support vector machines and deep residual networks. Neural Comput Appl. 2023;35(16):12073–87.

18. Jiang F, Lu Y, Chen Y, Cai D, Li G. Image recognition of four rice leaf diseases based on deep learning and support vector machine. Comput Electron Agric. 2020;179:105824.

19. Zhu H, Yang L, Fei J, Zhao L, Han Z. Recognition of carrot appearance quality based on deep feature and support vector machine. Comput Electron Agric. 2021;186:106185.

20. Azarmdel H, Jahanbakhshi A, Mohtasebi SS, Muñoz AR. Evaluation of image processing technique as an expert system in mulberry fruit grading based on

ripeness level using artificial neural networks and support vector machine. Postharvest Biol Technol. 2020;166:111201.

21. Sheykhmousa M, Mahdianpari M, Ghanbari H, Mohammadimanesh F, Ghamisi P, Homayouni S. Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review. IEEE J Sel Top Appl Earth Obs Remote Sens. 2020;13:6308–25.

22. Sumathi K, Pandiaraja P. E-health care patient information retrieval and monitoring system using SVM. Springer; 2023.

23. Singh V, Poonia RC, Kumar S, Dass P, Agarwal P, Bhatnagar V, Raja L. Prediction of COVID-19 coronavirus pandemic based on time series data using support vector machine. J Discrete Math Sci Cryptogr. 2020;23(8):1583–97.

24. Chhajer P, Shah M, Kshirsagar A. The applications of artificial neural networks, support vector machines, and long–short term memory for stock market prediction. Decis Anal J. 2022;2:100015.

25. Cao L, Tay FE. Financial forecasting using support vector machines. Neural Comput Appl. 2001;10:184–92.

26. Vapnik V. The nature of statistical learning theory. Springer science & business media; 1999.

27. Huang X, Shi L, Suykens JAK. Support vector machine classifier with pinball loss. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2014;36(5):984-97.

28. Khemchandani R, Pal A, Chandra S. generalized pinball loss SVMs. Neurocomputing. 2018 10;322.

29. Wu Y, Liu Y. Robust truncated hinge loss support vector machines. Journal of the American Statistical Association. 2007;102(479):974-83. Available from: http://www.jstor.org/stable/27639939.

30. Shen X, Niu L, Qi Z, Tian Y. Support vector machine classifier with truncated pinball loss. Pattern Recognition. 2017 03;68.

31. Yang L, Dong H. Support vector machine with truncated pinball loss and its application in pattern recognition. Chemometrics and Intelligent Laboratory Systems. 2018;177:89-99.

32. Hazarika D, Gupta D, Borah P. Robust support vector quantile regression with truncated pinball loss (RSVQR). Computational and Applied Mathematics. 2023 08;42.

33. Wang K, Zhu W, Zhong P. Robust support vector regression with generalized loss function and applications. Neural Processing Letters. 2015;41:89-106. Available from: https://api.semanticscholar.org/CorpusID:16729338.

34. Hoai Minh L, Le Thi HA, Nguyen M. Sparse semi-supervised support vector machines by DC programming and DCA. Neurocomputing. 2015 04;153.

35. An LTH, Tao PD. The DC (Difference of Convex Functions) Programming and DCA Revisited with DC Models of RealWorld Nonconvex Optimization Problems. Annals of Operations Research. 2005;133(1):23-46. Available from: https://doi.org/10.1007/s10479- 004-5022-1.

36. Yang L, Zhang S. A sparse extreme learning machine framework by continuous optimization algorithms and its application in pattern recognition. Engineering Applications of Artificial Intelligence. 2016;53:176-89.

37. Sigillito WSHL V, Baker K. Ionosphere; 1989. DOI: https://doi.org/10.24432/C5W01B. UCI Machine Learning Repository.

38. Smith J, Everhart J, Dickson W, Knowler W, Johannes R. Using the ADAP Learning Algorithm to Forcast the Onset of Diabetes Melli- tus. Proceedings - Annual Symposium on Computer Applications in Medical Care. 1988 11;10.

39. Antal B, Hajdu A. Diabetic Retinopathy Debrecen; 2014. DOI: https://doi.org/10.24432/C5XP4P. UCI Machine Learning Repos- itory.

40. Liver Disorders; 1990. DOI: https://doi.org/10.24432/C54G67. UCI Machine Learning Repository.

41. Sejnowski T, Gorman R. Connectionist Bench (Sonar, Mines vs. Rocks);. DOI: https://doi.org/10.24432/C5T01Q. UCI Machine Learning Repository.

42. Heart Failure Clinical Records; 2020. DOI: https://doi.org/10.24432/C5Z89R. UCI Machine Learning Repository.

43. Patrcio PJCJMPSR Miguel, Caramelo F. Breast Cancer Coimbra; 2018. DOI: https://doi.org/10.24432/C52P59. UCI Machine Learn- ing Repository.

44. Rice (Cammeo and Osmancik); 2019. DOI: https://doi.org/10.24432/C5MW4Z. UCI Machine Learning Repository.

45. Wolberg MOSN William, Street W. Breast Cancer Wisconsin (Di- agnostic); 1995. DOI: https://doi.org/10.24432/C5DW2B. UCI Machine Learning Repository.

46. NA N. National Health and Nutrition Health Survey 2013-2014 (NHANES) Age Prediction Subset; 2023. DOI: https://doi.org/10.24432/C5BS66. UCI Machine Learning Repository.

47. García S, Fernández A, Luengo J, Herrera F. Advanced nonparamet- ric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Information sciences. 2010;180(10):2044-64.

48. Bottou L, Curtis FE, Nocedal J. Optimization methods for large-scale machine learning. SIAM Rev. 2018;60(2):223–311.

49. Guo TD, Liu Y, Han CY. An overview of stochastic quasi-Newton methods for large-scale machine learning. J Oper Res Soc China. 2023;11(2):245–75.

50. Sakr C, Patil A, Zhang S, Kim Y, Shanbhag N. Minimum precision requirements for the SVM-SGD learning algorithm. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2017. p. 1138–42.

51. Wang Z, Shao YH, Bai L, Li CN, Liu LM, Deng NY. Insensitive stochastic gradient twin support vector machines for large-scale problems. Inf Sci. 2018;462:114–31.

52. Kaur R, Singh S. A comprehensive review of object detection with deep learning. Digit Signal Process. 2023;132:103812.

53. Pei J, Wu X, Liu X, Gao L, Yu S, Zheng X. SGD-YOLOv5: a small object detection model for complex industrial environments. In: 2024 International Joint Conference on Neural Networks (IJCNN). IEEE; 2024. p. 1–10.

54. Wijnhoven R, With PHN. Fast training of object detection using stochastic gradient descent. In: Proceedings of IEEE International Conference on Pattern Recognition (ICPR). 2010. p. 424–7.

55. Razaviyayn M, Sanjabi M, Luo ZQ. A stochastic successive minimization method for nonsmooth nonconvex optimization with applications to transceiver design in wireless communication networks. Math Program. 2016;157:515–45.

56. Bastin F, Cirillo C, Toint PL. Convergence theory for nonconvex stochastic programming with an application to mixed logit. Math Program. 2006;108:207–34.

57. Chong EKP, Zak SH. An introduction to optimization. 4th ed. New York: John Wiley & Sons; 2013.

58. Luo J, Qiao H, Zhang B. Learning with smooth hinge losses. Neurocomputing. 2021;463:379–87. doi:10.1016/j.neucom.2021.08.060.

59. Zhu W, Song Y, Xiao Y. Support vector machine classifier with huberized pinball loss. Eng Appl Artif Intell. 2020;91:103635.

60. Makmuang D, Ratiphaphongthon W, Wangkeeree R. Smooth support vector machine with generalized pinball loss for pattern classification. J Supercomput. 2023;79(11):11684–706. doi:10.1007/s11227-023-05082-w.

61. Chen H, Guo C, Xiong H, Wang Y. Sparse additive machine with ramp loss. Anal Appl. 2021;19(3):509–28.

62. Yuan P, You X, Chen H, Wang Y, Peng Q, Zou B. Sparse additive machine with the correntropy-induced loss. IEEE Trans Neural Netw Learn Syst. 2023;1–15. doi:10.1109/TNNLS.2023.3280349.

63. Singh A, Pokharel R, Principe J. The c-loss function for pattern classification. Pattern Recognit. 2014;47(1):441–53.

64. Xu G, Cao Z, Hu BG, Principe JC. Robust support vector machines based on the rescaled hinge loss function. Pattern Recognit. 2017;63:139–48.

65. Yang L, Dong H. Robust support vector machine with generalized quantile loss for classification and regression. Appl Soft Comput. 2019;81:105483.

66. Nocedal J, Wright SJ. Numerical Optimization. 2nd ed. Springer, New York, NY, USA; 2006.

67. Li DH, Fukushima M. On the global convergence of the BFGS method for nonconvex unconstrained optimization problems. SIAM J Optim. 2001;11(4):1054–64.

68. Quinlan, R.: Statlog (Australian Credit Approval). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C59012 (1987).

69. Weiss, S.M., Kulikowski, C.A.: Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems. Morgan Kaufmann Publishing, San Mateo (1991).

70. Janosi, A., Steinbrunn, W., Pfisterer, M., Detrano, R.: Heart Disease. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52P4X (1989).

71. Wolberg, W., Mangasarian, O., Street, N., Street, W.: Breast Can- cer Wisconsin (Diagnostic). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5DW2B (1995).

72. Ordoni, E., Bach, J., Fleck, A.-K.: Auction Verification. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52K6N (2022).

73. Bottou L, Curtis FE, Nocedal J. Optimization methods for large-scale machine learning. SIAM Rev. 2018;60(2):223-311.

74. Guo TD, Liu Y, Han CY. An overview of stochastic quasi-Newton methods for large-scale machine learning. J Oper Res Soc China. 2023;11(2):245-275.

75. Sakr C, Patil A, Zhang S, Kim Y, Shanbhag N. Minimum precision requirements for the SVM-SGD learning algorithm. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2017. p. 1138-42.

76. Wang Z, Shao YH, Bai L, Li CN, Liu LM, Deng NY. Insensitive stochastic gradient twin support vector machines for large-scale problems. Inf Sci. 2018;462:114-31.

77. Kaur R, Singh S. A comprehensive review of object detection with deep learning. Digit Signal Process. 2023;132:103812.

78. Pei J, Wu X, Liu X, Gao L, Yu S, Zheng X. SGD-YOLOv5: A small object detection model for complex industrial environments. In: 2024 International Joint Conference on Neural Networks (IJCNN). IEEE; 2024. p. 1-10.

79. Wijnhoven R, With PHN. Fast training of object detection using stochastic gradient descent. In: Proceedings of IEEE International Conference on Pattern Recognition (ICPR). 2010. p. 424-7.

80. Razaviyayn M, Sanjabi M, Luo ZQ. A stochastic successive minimization method for nonsmooth nonconvex optimization with applications to transceiver design in wireless communication networks. Math Program. 2016;157:515-45.

81. Bastin F, Cirillo C, Toint PL. Convergence theory for nonconvex stochastic programming with an application to mixed logit. Math Program. 2006;108:207-34.

82. Brownstone D, Bunch DS, Train K. Joint mixed logit models of stated and revealed preferences for alternative-fuel vehicles. Transp Res B. 2000;34(5):315-38.

83. Hensher DA, Greene WH. The mixed logit model: The state of practice. Transp. 2003;30(2):133-76.

84. Singh H, Bhatt P, Jacob S, Kaur A, Vashist A, Vij D. Stock prediction on historical data based on SGD and LSTM. In: 2022 2nd International Conference on Innovative Practices in Technology and Management (ICIPTM). IEEE; 2022. p. 200-4.

85. Kamalov F, Smail L, Gurrib I. Stock price forecast with deep learning. In: 2020 International Conference on Decision Aid Sciences and Application (DASA). IEEE; 2020. p. 1098-1102.

86. Ji S. Application research of SGD algorithm based on PCA dimensionality reduction technique for stock price prediction. In: 2023 International Conference on Electronics and Devices, Computational Science (ICEDCS). IEEE; 2023. p. 265-9.

87. Robbins H, Monro S. A stochastic approximation method. Ann Math Stat. 1915;22(3):400-7.

88. Boyd S, Vandenberghe L. Convex optimization. Cambridge (UK): Cambridge University Press; 2004.

89. Nocedal J, Wright SJ. Numerical optimization. 2nd ed. Springer; 2006.

90. Schraudolph NN, Yu J, Günter S. A stochastic quasi-Newton method for online convex optimization. In: Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AISTATS). 2007. p. 436-43.

91. Byrd RH, Hansen SL, Nocedal J, Singer Y. A stochastic quasi-Newton method for large-scale optimization. SIAM J Optim. 2016;26(2):1008-31. doi:10.1137/140954362.

92. Mokhtari A, Ribeiro A. RES: Regularized stochastic BFGS algorithm. IEEE Trans Signal Process. 2014;62(23):6089-104.

93. Indrapriyadarsini S, Mahboubi S, Ninomiya H, Asai H. A stochastic quasi-Newton method with Nesterov's accelerated gradient. In: Machine learning and knowledge discovery in databases. Lect Notes Comput Sci. 2019;11906.

94. Makmuang D, Suppalap S, Wangkeeree R. The regularized stochastic Nesterov's accelerated quasi-Newton method with applications. J Comput Appl Math. 2023;428:115190.

95. Wang X, et al. Stochastic quasi-Newton methods for nonconvex stochastic optimization. SIAM J Optim. 2017;27(2):927-56.

96. Chen H, Wu HC, Chan SC, Lam WH. A stochastic quasi-Newton method for large-scale nonconvex optimization with applications. IEEE Trans Neural Netw Learn Syst. 2020;31(11):4776–90. doi:10.1109/TNNLS.2019.2957843.

97. Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning. 2013;28:1139-47.

98. Nemirovski AS, Juditsky A, Lan G, Shapiro A. Robust stochastic approximation approach to stochastic programming. SIAM J Optim. 2009;19:1574–609.

99. Durrett R. Probability: theory and examples. London: Cambridge University Press; 2010.

100. Luo J, Qiao H, Zhang B. Learning with smooth hinge losses. Neurocomputing. 2021;463:379–87.

101. Feng Y, Yang Y, Huang X, Mehrkanoon S, Suykens JAK. Robust support vector machines for classification with nonconvex and smooth losses. Neural Comput. 2016;28(6):1217–47.

102. Kroese, Dirk P., Zdravko Botev, and Thomas Taimre. Data science and machine learning: mathematical and statistical methods. Chapman and Hall/CRC; 2019.

103. Dhara A, Dutta J. Optimality Conditions in Convex Optimization: A Finite-Dimensional View (1st ed.). CRC Press; 2011.

104. Friedberg, Stephen H., Arnold J. Insel, and Lawrence E. Spence. Linear algebra. Vol. 4. Essex, NJ, USA: Pearson; 2014.

105. Edwin K. P. Chong, Stanislaw H. Żak. An Introduction to Optimization. John Wiley & Sons, Inc; 2008.

106. Dhara A, Dutta J. Optimality Conditions in Convex Optimization: A Finite-Dimensional View (1st ed.). CRC Press; 2011.

107. Kroese, Dirk P., Zdravko Botev, and Thomas Taimre. Data science and machine learning: mathematical and statistical methods. Chapman and Hall/CRC; 2019.

108. Bartle, R.G. and Sherbert, D.R. Introduction to Real Analysis. Wiley; 2011.

109. Beck, Amir. Introduction to Nonlinear Optimization. Society for Industrial and Applied Mathematics; 2014.

110. David Williams. Probability with martingales. Cambridge Mathematical Textbooks.Cambridge University Press. Cambridge; 1991.