ENHANCED PATTERN CLASSIFICATION VIA NOISE-INSENSITIVE LOSS FUNCTIONS IN SUPPORT VECTOR MACHINES OPTIMIZED BY FIRST AND SECOND ORDER METHODS

DAWRAWEE MAKMUANG

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 2024

This thesis entitled "Enhanced Pattern Classification via Noise-Insensitive Loss Functions in Support Vector Machines Optimized by First and Second Order Methods"

by Dawrawee Makmuang

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

(Professor Somyot Plubtieng, Ph.D.)

..... Internal Examiner

(Associate Professor Kasamsuk Ungchittrakool, Ph.D.)

..... External Examiner

(Associate Professor Cholwich Nattee, Ph.D.)

Approved

.....

(Associate Professor Krongkarn Chootip, Ph.D.)

Dean of the Graduate School

13 MAR 2024

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 the Science Achievement Scholarship of Thailand (SAST) 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, Professor Dr.Somyot Plubtieng, Associate Professor Dr.Kasamsuk Ungchittrakool, and Associate Professor Dr.Cholwich Nattee. It is my pleasure to express my thanks 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.

Dawrawee Makmuang

Title	ENHANCED PATTERN CLASSIFICATION VIA NOISE-INSENSITIVE LOSS FUNCTIONS IN SUPPORT VECTOR MACHINES OPTIMIZED BY FIRST AND SECOND ORDER METHODS
Author	Dawrawee Makmuang
Advisor	Professor Rabian Wangkeeree, Ph.D.
Academic Paper	Ph.D. Dissertation in Mathematics, Naresuan University, 2023.
Keywords	Quasi-Newton method, stochastic optimization method, nesterov's accelerated gradient, momentum coefficient, support vector machine, generalized pinball loss function, least squares twin support vector machine, smooth approximations, asymmetric loss function, noise sensitivity, large-scale problems, image classification, pre-trained model.

ABSTRACT

This thesis introduces novel advancements in three key areas of optimization and machine learning techniques.

First, a regularized stochastic Nesterov's accelerated quasi-Newton method is proposed, aiming to resolve the near-singularity issues of the approximate Hessian matrix and expedite convergence in strongly convex optimization problems. By combining Nesterov acceleration with a novel momentum coefficient, this method significantly enhances convergence speed. Experimental validations on synthetic and benchmark datasets substantiate its superiority in solving classification problems, outperforming other existing methods.

Second, the study addresses the limitations of the generalized pinball loss function within support vector machines (SVM). As the original problem lacks differentiability, hindering practical algorithm selection, a new smooth approximation function of the generalized pinball loss function is developed for SVM. This enhancement allows the utilization of the quasi-Newton-Armijo method, exhibiting estimative capabilities for the generalized pinball loss function. Comprehensive experimental analyses across various benchmark datasets showcase the proposed

method's superior performance, yielding up to a 1.59% improvement for linear kernels and 1.45% for nonlinear kernels compared to baseline SVM methods.

Finally, a novel variant of the least square twin bounded support vector machine (LSTBSVM-linex) is introduced, incorporating the asymmetric LINEX loss function. This variant applies varying penalties to samples based on their positions, significantly penalizing points between central planes while assigning lesser penalties to points outside these planes. The proposed LSTBSVM-linex is further optimized for large-scale problems using the Adam optimization method. Experimentation across benchmark and image datasets demonstrates enhanced classifier efficiency and remarkably robust performance.

These contributions collectively advance the efficiency, robustness, and applicability of optimization and machine learning techniques, offering promising avenues for future research and practical implementation.

LIST OF CONTENTS

Chapter **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES (CONT.)

Figure **Page**

LIST OF TABLES

Table **Page**

LIST OF TABLES (CONT.)

Table **Page**

CHAPTER I

INTRODUCTION

In the current landscape of technological advancements, the immediate progress in machine learning and optimization techniques is grabbing attention for its importance and relevance. With the increasing reliance on computational models for decision-making and problem-solving, there is a need for more intelligent and efficient algorithms. These algorithms must handle extensive data while solving complex problems quickly and accurately. The interest in these advancements is not just academic; it extends to practical applications across industries. These developments hold the potential to transform processes, streamline resources, and offer valuable insights. In this context, this thesis aims to contribute to these ongoing advancements by focusing on refining and advancing essential optimization and machine learning techniques, striving to tackle modern challenges and enhance the field's effectiveness and real-world use.

The focus of my interest in the optimization algorithm lies in the field of quasi-Newton methods. Quasi-Newton methods, like the steepest descent, require only the gradient of the objective function to be supplied at each iterate. By measuring the changes in gradients, they construct a model of the objective function that is good enough to produce superlinear convergence. The improvement over the steepest descent is dramatic, especially on complex problems. Moreover, since second derivatives are not required, quasi-Newton methods are sometimes more efficient than Newton. Nowadays, optimization software libraries contain a variety of quasi-Newton algorithms for solving unconstrained, constrained, and large-scale optimization problems.

On the other hand, the stochastic quasi-Newton methods [2], referred to as oBFGS, adapt these classical quasi-Newton techniques to handle noisy or large-scale data scenarios. They leverage stochastic gradients, which are unbiased estimates of true gradients computed from subsets of data, to approximate Hessian matrix. By incorporating these stochastic estimates, stochastic quasi-Newton methods aim to offer quicker convergence rates and reduced computational demands. However, its performance is still limited. For example, the problem of the approximate Hessian

matrix does not guarantee its positive definiteness, which might result in unstable or erratic optimization directions. This instability can hinder convergence, causing the optimization algorithm to oscillate or deviate from the optimal path. As a result, many research works have proposed solutions to address these problems. In particular, Mokhtari and Ribeiro [3] have proposed a regularized stochastic BFGS method (RES). This innovative approach incorporates a regularization technique to ensure that the approximate Hessian matrix exceeds the positive constant. Recently, a stochastic Nesterov's accelerated quasi-Newton method (oNAQ) [5] was proposed to accelerate the convergence rate of stochastic quasi-Newton-based methods by using a momentum coefficient term [6]. Experimental results show that oNAQ could reduce computational and memory requirements compared to the oBFGS. However, in practical training, oNAQ still has some challenges. The first challenge is that the momentum coefficient in most practical training is a problem-dependent parameter, that is, a hyperparameter. Therefore, it must be chosen carefully. The second challenge is the near-singularity of the approximate Hessian matrix. Although oNAQ provides a term that prevents near-singularity, it still needs to be theoretically supported.

Another area of interest in this thesis is Support Vector Machines (SVMs), widely recognized for their robust classification capabilities. The fundamental concept of SVM [9] is to search parallel hyperplanes with the maximum margin between two classes of samples to achieve classification by minimizing the regularization and classification errors. The models of SVMs are convex optimization problems that give the global solution, and several practical algorithms [10–12] have been developed over the past several decades. Nowadays, SVMs are widely used to solve a variety of real-world problems, such as text categorization [13], scene classification [14], face recognition [15], and medical image classification [16].

In classical SVM, it penalises misclassification by employing the hinge loss function, which suffers from noise sensitivity and is unstable for re-sampling. To address the problems, Huang et al. [17] introduced the pinball loss into the SVM (Pin-SVM), which effectively solved the problems of noise sensitivity and unstable resampling. However, Pin-SVM does have a downside: it loses the existence of the sparse solution. As a result, [17] also proposed a modified ϵ-insensitive zone in the Pin-SVM to allow the existence of the sparse solution. Unfortunately, although

the ϵ-insensitive zone Pin-SVM improves the sparsity of Pin-SVM and remains noise-sensitive and unstable for resampling, its formulation requires the value of ϵ to be specified beforehand; therefore, a wrong choice may affect its performance. With motivation from these developments, Reshma et al. [18] proposed a new generalized pinball loss function in the SVM, referred to as GP-loss-SVM. This loss function allows asymmetric insensitive zone by (ϵ_1 and ϵ_2) to differ. Therefore, an exciting feature of the GP-loss-SVM is that it appropriately trade-offs the generalization ability, the sparsity, and the noise insensitivity. However, the mentioned loss functions lack differentiability at 0. Consequently, it restricts the SVM's ability to select algorithms for achieving the optimal solution. In this case, the development of an approximate smooth loss function for SVM is a promising research area, e.g., [19–21].

On the other hand, one of the main challenges of SVMs is their high computational complexity, because the objective function involves dealing with a single large-scale quadratic programming problem (QPP) to find an optimal separation hyperplane. As a result, handling large datasets might be limited, especially in scenarios with numerous features or instances. To overcome this issue, Jayadeva et al. [22] proposed a twin support vector machine (TSVM) that generates two nonparallel hyperplanes such that each hyperplane is closer to one of two classes and is at least one far from the other classes. The strategy of TSVM aimed at solving a pair of smaller-sized QPPs instead of solving a large one as in the classical SVM. Therefore, it makes the computational time of TSVM approximately faster than the standard SVM. Based on the concept of development TSVM, various TSVM-based algorithms were proposed. One such method is the Least Squares Twin SVM (LSTBSVM), introduced by Yitian Xu et al. [23]. LSTBSVM obtains two non-parallel planes directly by solving two systems of linear equations. Additionally, In [24] introduced a novel Large-Scale LSTBSVM, which can efficiently be solved using a fast iterative scheme named SMO, which made it more suitable for large-scale analysis. However, solving two systems of linear equations in LSTBSVM can render the model sensitive to outliers or noise. This sensitivity arises due to the model's deficiency of consideration for the location information of the sample points.

Motivated by all the mentioned above, we introduce a multifaceted approach

to refining existing methodologies. Initially, we focus on enhancing the efficiency of the stochastic quasi-Newton method by proposing a novel method. This method integrates the RES concept with the Nesterov accelerating technique, introducing a new momentum coefficient to augment performance further. Turning our attention to improving the SVM-based model, we present a dual-part approach. In the first part, we introduce a smooth approximation function for the generalized pinball loss function. This substitution enables the utilization of the quasi-Newton algorithm, significantly boosting the efficiency of solving the optimization problem. Expanding our efforts, we propose a novel large-scale LSTBSVM tailored to treat instances based on individual point importance. We then adopt the Adam algorithm to solve it, which offers significant advantages in handling large-scale classification problems. Note that Adam is developed from various types of quasi-Newton methods; we will analyze this in more detail in the next chapter.

This thesis is organized in the following way.

Chapter II. We review fundamental definition, Lemmas, and Theorems relevant to the problems addressed within our framework. Additionally, we conduct a comprehensive review of prior research about the stochastic quasi-Newton method and support vector machine, establishing a robust foundation for our work.

Chapter III. We present a novel regularized stochastic Nesterov's accelerated BFGS method and analyse the convergence result of our proposed method. We also give applications and numerical results.

Chapter IV. We present two typical support vector machines in this study. Initially, we incorporate a smooth generalized pinball loss function into the support vector machine framework, showcasing its effectiveness in estimating the generalized pinball loss function. To validate our method, we conduct numerical experiments across various datasets. Additionally, we introduce a large-scale LSTBSVM utilizing the asymmetric loss function and verify its performance using large-scale datasets. Finally, we demonstrate its application in medical image classification.

Chapter V. We give the conclusion of this thesis.

CHAPTER II

PRELIMINARIES

In this section, we embark on a comprehensive exploration of fundamental theories, notably focusing on convexity, which is the cornerstone of this thesis. Subsequently, we will describe the development guidelines for the stochastic quasi-Newton method and the support vector machine, elucidating the creative approaches and improvements.

Throughout this thesis, we consider the Euclidean space $\mathbb{R}^n$, and $\|\cdot\|$ denotes the Euclidean norm of a vector in $\mathbb{R}^n$. All vectors are assumed to be column vectors, which can be converted to row vectors by applying the transpose operation, 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 dot product of $\mathbf{x}$ and $\mathbf{y}$ is denoted by $\mathbf{x}^\top\mathbf{y} = \sum_{i=1}^n x_i y_i$, while the norm of $\mathbf{x}$ is given by $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^n x_i^2}$.

2.1 Mathematical background

Definition 2.1.1. [25] Let A be an $n \times n$ real square matrix. A nonzero vector $\mathbf{v}$ is called an **eigenvector** of A if there exists a scalar λ such that $A\mathbf{v} = \lambda\mathbf{v}$. The scalar λ is called an **eigenvalue** corresponding to eigenvector $\mathbf{v}$.

Understanding the relationship between a matrix and its eigenvectors and eigenvalues lays the foundation for exploring more complex matrix properties, such as the spectral norm.

Definition 2.1.2 (Spectral norm). [30] Let $\mathbf{A}$ be an $(m \times n)$-matrix. Then

$$\|\mathbf{A}\| := \max_{\mathbf{x}\in\mathbb{R}^n, \mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| \tag{2.1.1}$$

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,

$$\|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\|\|\mathbf{x}\|, \tag{2.1.2}$$

for all $\mathbf{x}$.

Definition 2.1.3. [29] A sequence $\{\mathbf{x}_n\}$ in $\mathbb{R}^n$ is said to be convergent to an element $\mathbf{x} \in \mathbb{R}^n$ if $\lim_{n\to\infty} \|\mathbf{x}_n - \mathbf{x}\| = 0$. We usually write $\lim_{n\to\infty} \mathbf{x}_n = \mathbf{x}$ or $\mathbf{x}_n \to \mathbf{x}$ as $n \to \infty$ and call the element $\mathbf{x}$ the limit of the sequence $\{\mathbf{x}_n\}$.

Theorem 2.1.4 (Squeeze theorem). [29] Let $A \subset \mathbb{R}$, let $f, g, h : A \to \mathbb{R}$, and let $c \in \mathbb{R}$ be a point of A. If

$$f(x) \leq g(x) \leq h(x) \text{ for all } x \in A, x \neq c, \tag{2.1.3}$$

and if $\lim_{x\to c} f(x) = L = \lim_{x\to c} h(x)$, then $\lim_{x\to c} g(x) = L$.

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 2.1.5. [30] Let F be a function defined on a set $S \subseteq \mathbb{R}^n$. Let $\mathbf{x} \in \text{int}\,(S)$ and let $0 \neq \mathbf{d} \in \mathbb{R}^n$. If the limit

$$\lim_{t\to 0^+} \frac{F(\mathbf{x}+t\mathbf{d}) - F(\mathbf{x})}{t}$$

exists, then it is called the directional derivative of F at $\mathbf{x}$ along the direction $\mathbf{d}$ and is denote by $F'(\mathbf{x};\mathbf{d})$. For any $i = 1, 2, \ldots, n$, the directional derivative at $\mathbf{x}$ along the direction $\mathbf{e}_i$ (the ith vector in the standard basis) is call the ith partial derivative and is denoted by $\frac{\partial F}{\partial \mathbf{x}_i}(x)$

$$\frac{\partial F}{\partial x_i}(\mathbf{x}) = \lim_{t\to 0^+} \frac{F(\mathbf{x}+t\mathbf{e}_i) - F(\mathbf{x})}{t}.$$

If all the partial derivatives of a function F exist at point $\mathbf{x} \in \mathbb{R}^n$, then the gradient of F at $\mathbf{x}$ is defined to be the column vector consisting of all the partial derivatives:

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial F}{\partial x_1}(\mathbf{x}) \\ \frac{\partial F}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial F}{\partial x_n}(\mathbf{x}) \end{bmatrix}. \tag{2.1.4}$$

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.

Similar to the first-order differentiability, we have the second-order differentiability notion as follows. 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

$$\nabla^2 F(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 F}{\partial x_1^2}(\mathbf{x}) & \frac{\partial^2 F}{\partial x_1 \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \frac{\partial^2 F}{\partial x_2 \partial x_1}(\mathbf{x}) & \frac{\partial^2 F}{\partial x_2^2}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1}(\mathbf{x}) & \frac{\partial^2 F}{\partial x_n \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 F}{\partial x_n^2}(\mathbf{x}) \end{pmatrix} \tag{2.1.5}$$

The matrix $\nabla^2 F(\mathbf{x})$ is called the Hessian matrix of F at $\mathbf{x}$, and is often also denoted by $\mathbf{H}$. Since F is twice continuously differentiable, the Hessian matrix is symmetric. A main approximation result (quadratic) is a direct consequence of Taylor's approximation theorem. An important property of Taylor's theorem arises from the form of the remainder in $\mathbb{R}^m$. We introduce the order symbols O and o to discuss this property further [26].

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

1. $f(\mathbf{x}) = O(g(\mathbf{x}))$ 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.

2. $f(\mathbf{x}) = o(g(\mathbf{x}))$ 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 $\lim_{\mathbf{x} \to \mathbf{0}} \|o(g(\mathbf{x}))\|/|g(\mathbf{x})| = 0$.

Theorem 2.1.6. [30] Let $F : \mathbb{R}^n \to \mathbb{R}$ be a function that is twice continuously differentiable at a point $\mathbf{x} \in \mathbb{R}^n$. The quadratic approximation of F at $\mathbf{x}$ is given by

$$F(\mathbf{y}) = F(\mathbf{x}) + \nabla F(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{1}{2}(\mathbf{y} - \mathbf{x})^\top H (\mathbf{y} - \mathbf{x}) + o(\|\mathbf{y} - \mathbf{x}\|^2),$$

for $\mathbf{y}$ near $\mathbf{x}$, where $\nabla F(\mathbf{x})$ is the gradient of F at $\mathbf{x}$, and $\mathbf{H}$ is the Hessian matrix of F at $\mathbf{x}$. The term $o(\|\mathbf{y}-\mathbf{x}\|^2)$ represents the error of the approximation, which becomes negligible as $\mathbf{y}$ approaches $\mathbf{x}$.

Theorem 2.1.7 (first order optimality condition). [30] Let $F : \mathbb{R}^n \to \mathbb{R}$ be a function. Suppose that $\mathbf{x}^*$ is a local optimum point and that all the partial derivatives of F exist at $\mathbf{x}^*$. Then $\nabla F(\mathbf{x}^*) = 0$.

Definition 2.1.8. [29] if $A \subset \mathbb{R}$, then a function $f : A \to \mathbb{R}$ is said to be *increasing* on A if whenever $x, y \in A$ and $x \leq y$, then $f(x) \leq f(y)$. The function f is said to be *strictly increasing* on A if whenever $x, y \in A$ and $x < y$, then $f(x) < f(y)$. Similarly, $g : A \to \mathbb{R}$ is said to be *decreasing* on A if whenever $x, y \in A$ and $x \leq y$, then $g(x) \geq g(y)$. The function g is said to be *strictly decreasing* on A if whenever $x, y \in A$ and $x < y$, then $g(x) > g(y)$.

Theorem 2.1.9. [29] Let $f : A \to \mathbb{R}$ be differentiable on the interval I. Then:

1. f is increasing on I if and only if $f'(x) \geq 0$ for all $x \in I$,
2. f is decreasing on I if and only if $f'(x) \leq 0$ for all $x \in I$.

In order to characterize the second-order optimality conditions, which are expressed via the Hessian matrix, the notion of positive definiteness must be defined.

Definition 2.1.10. [30] 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 $0 \neq \mathbf{x} \in \mathbb{R}^n$.

Rayleigh's Inequalities. [26] If an $n \times n$ matrix $\mathbf{P}$ is real symmetric positive definite, then

$$\lambda_{\min}(\mathbf{P})\|\mathbf{x}\|^2 < \mathbf{x}^\top \mathbf{P}\mathbf{x} < \lambda_{\max}(\mathbf{P})\|\mathbf{x}\|^2, \tag{2.1.6}$$

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

Next, we will review the definitions of infimum and supremum.

Definition 2.1.11. [29] Suppose that $A \subseteq \mathbb{R}$ is a set of real numbers. If $M \in \mathbb{R}$ is an upper bound of A such that $M \leq M'$ for every upper bound M' of A, then M is called the supremum of A, denoted $M = \sup A$. If $m \in \mathbb{R}$ is a lower bound of A such that $m \geq m'$ for every lower bound m' of A, then m is called the infimum of A, denoted $m = \inf A$.

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

Definition 2.1.12. [27] 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 2.1.13. [27] A function $F : C \to \mathbb{R}$ defined on a convex set $C \subseteq \mathbb{R}^n$ is convex if and only if for all $\mathbf{x}_1, \mathbf{x}_2 \in C$ and all $\lambda \in (0, 1)$, we have

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

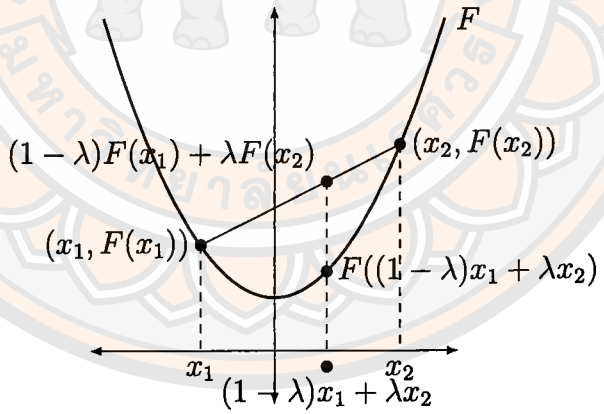


Figure 1 Geometric interpretation of convex functions.

Using Theorem 2.1.13, it is straightforward to obtain that the sum of convex functions is convex.

Theorem 2.1.14. [27] Suppose that F_1, F_2 are convex functions. Then, $F_1 + F_2$ is convex.

The following theorem gives another possible characterization of convexity for twice continuously differentiable functions.

Theorem 2.1.15. [27] Let $F : C \to \mathbb{R}$ be a twice continuously differentiable function defined over a convex set $C \subseteq \mathbb{R}^n$. Then F is convex if and only if $\nabla^2 F(\mathbf{x}) \succeq 0$ for all $\mathbf{x} \in C$.

Simplifying the idea for convex optimization problems: the theorem establishes that if a point is a local minimizer, it is also a global minimizer.

Theorem 2.1.16. [27] Let $F : C \to \mathbb{R}$ be a convex function defined over a convex set $C \subseteq \mathbb{R}^n$. Then, a point is a global minimizer of F over C if and only if it is a local minimizer of F.

Lemma 2.1.17. [27] Suppose that $F : \mathbb{R}^n \to \mathbb{R}$ is a convex and continuously differentiable function. Let $\mathbf{x} \in dom(F)$. If $\nabla F(\mathbf{x}) = 0$, then $\mathbf{x}$ is a global minimum.

Definition 2.1.18. [27] The level set of a function $F : \mathbb{R}^n \to \mathbb{R}$ at level c is the set of points

$$S = \{\mathbf{x} : F(\mathbf{x}) = c\}. \tag{2.1.7}$$

Corollary 2.1.19. [27] Let $F : C \to \mathbb{R}$ be a convex function defined over a convex set $C \subseteq \mathbb{R}^n$. Then, the set of all global minimizers of F over C is a convex set.

The subsequent step involves defining the concept of strong convexity.

Definition 2.1.20. [27] Let $F : C \to \mathbb{R}$ be a function defined over a convex set $C \subseteq \mathbb{R}^n$. The function F is called strongly convex over C if there exists $\rho > 0$ such that the function $F(\mathbf{x}) - \frac{\rho}{2}\|\mathbf{x}\|^2$ is convex over C.

Definition 2.1.21. [27] Let $F : C \to \mathbb{R}$ be a function defined over a convex set $C \subseteq \mathbb{R}^n$. The function F is called strongly convex over C if there exists $\rho > 0$ such that the function $F(\mathbf{x}) - \frac{\rho}{2}\|\mathbf{x}\|^2$ is convex over C.

Theorem 2.1.22. [27] If $F : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable, then f is strongly convex if and only if for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$, there exists $\rho > 0$ such that

$$F(\mathbf{x}_2) - F(\mathbf{x}_1) \geq \nabla F(\mathbf{x}_1)^\top (\mathbf{x}_2 - \mathbf{x}_1) + \rho\|\mathbf{x}_2 - \mathbf{x}_1\|^2, \tag{2.1.8}$$

or if and only if for any $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n$, there exists $\rho > 0$ such that

$$(\nabla F(\mathbf{x}_2) - \nabla F(\mathbf{x}_1))^\top (\mathbf{x}_2 - \mathbf{x}_1) \geq \rho\|\mathbf{x}_2 - \mathbf{x}_1\|^2, \tag{2.1.9}$$

or if and only if there exists $\rho > 0$ such that

$$H(\mathbf{x}) \succeq \rho \mathbf{I}, \quad \forall x \in \mathrm{dom}(F). \tag{2.1.10}$$

Corollary 2.1.23. [27] For strongly convex optimization problems, a unique optimal solution exists.

2.2 Statistical background

Next, we will delve into the statistical theories that support our work, offering insights into the methodologies that guide our analysis. A fundamental concept in this exploration is the probability space, which provides a framework for understanding random processes and consists of three primary components: the sample space, the set of events, and the probability function. To elucidate these components, we present the following definitions and theories.

Definition 2.2.1. [28] 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. [28] 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. [28] A probability function is a function $\mathbb{P} : \mathcal{F} \to [0, 1]$ of an event E to a real number in $[0, 1]$.

With the foundational concepts outlined, we delve into more nuanced aspects of probability theory, such as the notion of events occurring almost surely, the definition and properties of random variables, and the mathematical expectations of these variables.

Definition 2.2.4 (Almost Surely). [28] An event E in a probability space $(\Omega, \mathcal{F}, P)$ is said to hold almost surely (a.s.) if

$$\mathbb{P}(E) = 1. \tag{2.2.1}$$

This definition underscores the distinction between events that are guaranteed to happen and those that are merely probable, an important consideration in probabilistic modelling.

A significant step in applying probability theory is the introduction of random variables, which serve as bridges between outcomes and numerical values.

Definition 2.2.5. [28] 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 allow us to model and analyze the randomness in numerical terms, making them indispensable statistical tools. Next, we move to the expectation concept, which is crucial in understanding the average outcome of random processes.

Definition 2.2.6. [28] The expectation (or expected value) of a random variable X is defined as follows:

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

$$\mathbb{E}[X] = \sum_{x \in X(\Omega)} x \cdot p_X(x), \tag{2.2.2}$$

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

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

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x \cdot f_X(x)\, dx, \tag{2.2.3}$$

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

Note that in our thesis, the notation $\mathbb{E}_X[\cdot]$ represents 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 possesses several useful properties, which are listed below. These properties apply to both discrete and continuous random variables.

Theorem 2.2.7. [28] 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. [28] 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

$$\mathbb{E}[X|Y = y] = \sum_{x \in X(\Omega)} x \cdot p_X(x|Y = y), \tag{2.2.4}$$

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

$$\mathbb{E}[X|Y = y] = \int_{-\infty}^{\infty} x \cdot f_X(x|Y = y)\, dx, \tag{2.2.5}$$

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

Theorem 2.2.9. [31] 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) $\mathbb{E}[X|Y, g(Y)] = \mathbb{E}[X|Y]$;

Theorem 2.2.10. [31] If X is a non-negative random variable (or a random variable with a non-negative absolute value), then

$$\mathbb{E}[X] = 0 \implies X = 0 \text{ (a.s.)}. \tag{2.2.6}$$

Theorem 2.2.11. [31] Let $\mathcal{G} \subset \mathcal{F}$ be a σ-algebra and $\{X_n\}$ be a sequence of non-negative random variables. Then,

$$\mathbb{E}[\liminf_{n\to\infty} X_n \mid \mathcal{G}] \leq \liminf_{n\to\infty} \mathbb{E}[X_n \mid \mathcal{G}] \text{ (a.s.)}. \tag{2.2.7}$$

Finally, this section turns to the notions of convergence for sequences of random variables.

Definition 2.2.12. [33] Let $\{X_n(\xi)\}$ be a sequence of random variables on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. Then X_n converges almost surely (a.s.) to X, written $X_n \to X$ (a.s.), if

$$\mathbb{P}\left(\xi : \lim_{n\to\infty} X_n(\xi) = X(\xi)\right) = 1.$$

Theorem 2.2.13. [33] If $\{V_{n-1}, \beta_n, \alpha_n\}$ are sequences of non-negative random variables adapted to a sequence of increasing $\sigma-$algebra $\mathcal{F}_n$ and

$$\mathbb{E}[V_n|\mathcal{F}_n] \leq V_{n-1} - \beta_n + \alpha_n$$

then if $\sum_{n=1}^{\infty} \alpha_n < \infty$ (a.s.), then

(I) V_n converges (a.s.).

(II) $\sum_{n=1}^{\infty} \beta_n < \infty$ (a.s.).

2.3 Optimization methods

In this section, we will briefly delve into some prior research surrounding optimization methods within the field of machine learning. In particular, we use a random seed θ to represent a sample or a set of samples. For instance, a realization of θ could be a single sample $\mathbf{x}$, or it might represent a set of independent and identically distributed (i.i.d.) samples $\{\mathbf{x}_i\}_{i\in D}$. It is important to note that i.i.d. means each sample is independent and comes from the same distribution D. In machine learning, the goal often involves minimizing the expected risk for a given $\mathbf{w}$, which is the expected value of this composite function with respect to the distribution of θ:

$$\mathbb{E}[f(\mathbf{w},\theta)], \tag{2.3.1}$$

where f represents the Loss function. However, minimizing (2.3.1) becomes untenable without complete knowledge of θ's distribution. Thus, in practice, the focus shifts to solving a problem that involves estimating the expected risk (2.3.1).

In supervised learning, access to a set of realizations $\{\theta_{[i]}\}_{i=1}^{N}$ of θ, corresponding to a sample set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, allows for the definition of the empirical risk function:

$$\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w},\theta_i). \tag{2.3.2}$$

Minimization of (2.3.2) is thus considered the practical optimization problem of interest. To highlight the broad applicability of the results presented in this section, it is important to note that the objective function under scrutiny could represent either the expected risk (2.3.3) or the empirical risk (2.3.4). Specifically, we consider the objective function $F : \mathbb{R}^n \to \mathbb{R}$, defined as either

$$F(\mathbf{w}) := \mathbb{E}[f(\mathbf{w},\theta)] \tag{2.3.3}$$

or

$$F(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w},\theta_i). \tag{2.3.4}$$

Our analyses apply equally to both objectives; the only difference lies in how one picks the stochastic gradient estimates in the method.

After presenting the form of the objective function, it is clear that optimization methods play a crucial role in minimizing an objective, particularly when finding a suitable model given a finite amount of data. In the next section, we delve into optimization methods, categorizing them into first-order and second-order methods. First-order methods, including Gradient Descent, Stochastic Gradient Descent, Momentum Stochastic Gradient Descent, and Adam, utilize gradients to update model parameters iteratively. Conversely, second-order methods, such as Newton and quasi-Newton, take advantage of both gradients and second derivatives to offer a more nuanced and precise optimization process.

Our initial discussions will focus on how these methods are applied to minimize empirical risk (2.3.4), a prevalent practice in machine learning for fitting models with available data.

2.3.1 First-order methods

In first-order optimization methods, the first derivative or gradient of the objective function with respect to the parameters is employed to guide the optimization process. To provide a foundation for understanding this class of methods, we begin by elucidating the concept of gradient descent (GD).

Given a set of realizations $\{\theta_{[i]}\}_{i=1}^{N}$ of θ, corresponding to a sample set $\{\mathbf{x}_i\}_{i=1}^{N}$, the gradient of the objective function at $\mathbf{w}$ is calculated as follows:

$$\nabla F(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \nabla f(\mathbf{w}, \theta_i). \tag{2.3.5}$$

Gradient Descent then sets $\mathbf{w}_0$ as the initial point and uses η_t as the step size for each t^{th} iteration. The parameters are iteratively updated through the sequence:

$$\mathbf{z}_{t+1} = -\eta_t \nabla F(\mathbf{w}_t),$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{z}_{t+1}.$$

In the following figure, we illustrate the trajectory of $\mathbf{w}$ through gradient descent by constructing the objective function $F : \mathbb{R}^2 \to \mathbb{R}$ defined by $F(\mathbf{w}) = w_1^2 + 2w_2$.

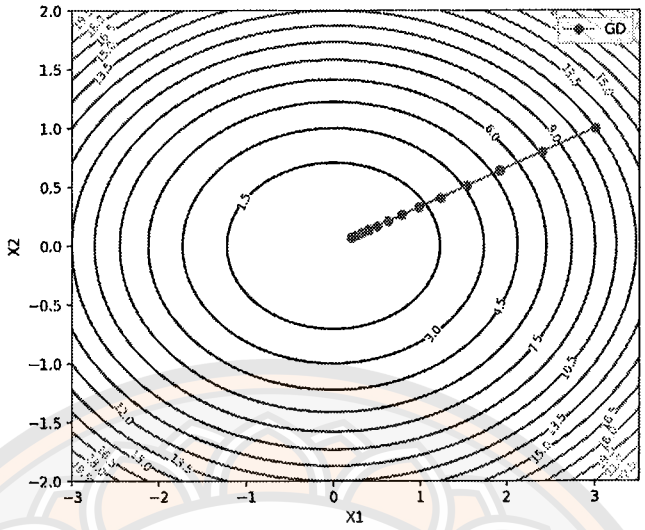


Figure 2 Optimization trajectory for GD method.

The figure visually represents the iterative steps of the algorithm, starting from the initial position $(5, 2)$.

The transition from traditional Gradient Descent (GD) to Stochastic Gradient Descent (SGD) represents a significant shift in updating model parameters. Instead of relying on the average gradient computed from the entire dataset, as is customary in GD, SGD introduces an element of randomness into the optimization process. This is achieved by selecting random subsets of data for each iteration, enhancing the method's efficiency, especially when dealing with large datasets.

During the t^{th} iteration of the stochastic gradient method, a set of realizations $\{\hat{\theta}_{t[l]}\}_{l=1}^{L}$ of $\hat{\theta}_t$, corresponding to a set of samples $\{\mathbf{x}_l\}_{l=1}^{L}$ chosen uniformly at random from a dataset of size N, is utilized. This approach distinguishes the stochastic gradient method based on the size of L; with $L = 1$, it is referred to as the stochastic gradient method, while for $L > 1$, it adopts the name of the minibatch stochastic gradient method. For clarity and consistency in our discussion, we will refer to both variations as the stochastic gradient method, irrespective of whether it involves updates from a single sample or minibatch processing. The stochastic gradient computation for each iteration is then formulated as:

$$\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t) := \frac{1}{L} \sum_{l=1}^{L} \nabla f(\mathbf{w}_t, \hat{\theta}_{tl}). \tag{2.3.6}$$

However, the gradient $\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t)$ may be far from the true gradient and exhibit

high variance. Despite this, in expectation over the random choice of $\hat{\theta}_t$, it coincides with the full gradient of F. We formalize this as:

$$\mathbb{E}[\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t)|\mathbf{w}_t = \mathbf{w}] = \frac{1}{N}\sum_{i=1}^{N} \nabla f(\mathbf{w}, \theta_i) = \nabla F(\mathbf{w}). \tag{2.3.7}$$

Here, $\mathbb{E}[\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t)|\mathbf{w}_t = \mathbf{w}]$ represents the conditional expectation of $\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t)$, given the event $\mathbf{w}_t = \mathbf{w}$. Thus, by (2.3.7), the stochastic gradient $\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t)$ is an unbiased estimate of the full gradient $\nabla F(\mathbf{w})$. This indicates that, on average, the stochastic gradient provides a reliable estimate of the gradient. Therefore, stochastic gradient is used to update $\mathbf{w}$ according to the following: at iteration $t = 0$,

$$\mathbf{z}_{t+1} = -\eta_t \hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{z}_{t+1}.$$

When the number of samples in the subset, denoted as L, is less than the total number of samples in the dataset (N), we observe that the computational cost of computing the gradient in SGD for each iteration is $O(L)$. This represents a reduction from the $O(N)$ computational cost associated with computing the gradient in GD. This reduction in computational cost is a notable advantage of SGD, especially in scenarios involving large datasets.

However, using SGD has its challenges. One significant issue is the tricky task of picking the right stepsize for the best convergence. Choosing a learning rate that is too high can lead to oscillations or divergence, while one that's too low can make the convergence slow, as shown in Figure3. Additionally, the noisy updates in SGD, caused by randomly sampling subsets in each iteration, make the optimization path less predictable and add variability to the convergence process. Despite these challenges, there's a solution: an improved method called SGD with Momentum.

SGD with Momentum introduces a momentum term to the optimization process, enabling the algorithm to accumulate inertia from past gradients. This momentum serves to smooth the search path, mitigating oscillations and expediting convergence as the model iteratively updates its parameters. The iteration of SGD with Momentum is defined as follows: at iteration $t = 0$,

$$\mathbf{z}_{t+1} = \beta \mathbf{z}_t - \eta_t \hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t),$$

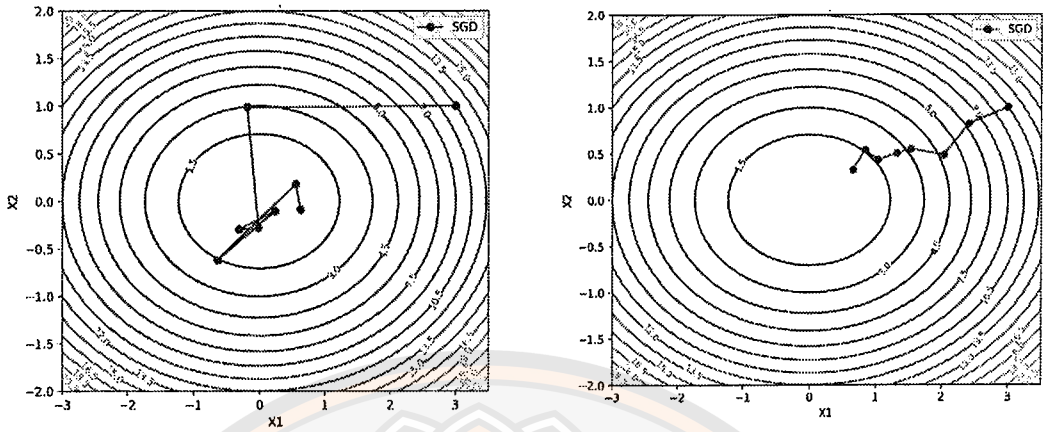


Figure 3 Comparison of SGD optimization trajectory with different step size. Left: SGD with large step size ($\eta_t = 1.0$) and right: SGD with small step size ($\eta_t = 0.1$).

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{z}_{t+1}.$$

Here, η_t is the stepsize, β is the momentum term, $\mathbf{z}_t$ is the velocity at iteration t with the initial condition $\mathbf{z}_0 = \mathbf{0}$. This initialization ensures that the momentum term starts at zero and gradually accumulates over iterations, influencing the refinement of parameter updates.

Although generally advantageous, introducing a momentum term in optimization algorithms has potential issues. A notable concern is the possibility of overshooting or oscillations, wherein the accumulated momentum may result in quick and unpredictable updates, potentially impeding the stability and convergence of the algorithm. In Figure 4, we illustrate SGD with Momentum with various values of β, shedding light on how this parameter influences the optimization dynamics.

Recently, Adam has stood out as the most renowned optimization algorithm in machine learning and deep learning. Adam effectively combines the characteristics of Momentum and adaptive learning rates at a high level, offering solutions to challenges encountered in both SGD and SGD with Momentum. The iteration of Adam is defined as follows: at iteration $t = 0$,

$$\begin{aligned}
\mathbf{m}_{t+1} &= \beta_{1,t}\mathbf{m}_t + (1-\beta_{1,t})\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t) \\
\mathbf{v}_{t+1} &= \beta_2 \mathbf{v}_t + (1-\beta_2)(\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t))^2 \\
\hat{\mathbf{m}}_{t+1} &= \frac{\mathbf{m}_{t+1}}{1-\beta_1^{t+1}}
\end{aligned}$$

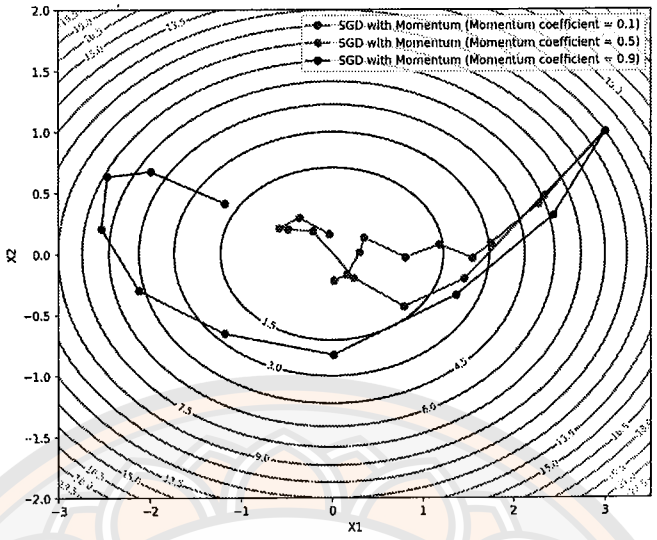


Figure 4 Comparison of SGD with momentum optimization trajectory with different momentum coefficient.

$$\hat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{1 - \beta_2^{t+1}}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta_t \hat{\mathbf{m}}_{t+1}}{\sqrt{\hat{\mathbf{v}}_{t+1}} + \epsilon},$$

Here β_1 and β_2 are the decay rates for the first and second moments, respectively, $\mathbf{m}_t$ and $\mathbf{v}_t$ are the mean and variance of the gradient of the previous steps, respectively with $\mathbf{m}_0 = \mathbf{0}$ and $\mathbf{v}_0 = \mathbf{0}$, and $\hat{\mathbf{m}}_t$ and $\hat{\mathbf{v}}_t$ are the biased corrected first and second moments, respectively. This iterative process characterizes Adam's dynamic updating mechanism, involving estimating and correcting the first and second moments to adjust the learning rates for each parameter.

In the final part of this section, we illustrate a comparison of the optimization paths taken by Gradient Descent (GD), Stochastic Gradient Descent (SGD), SGD with Momentum, and Adam, as shown in Figure 5. Next, we will explore second-order optimization algorithms in the following section.

2.3.2 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

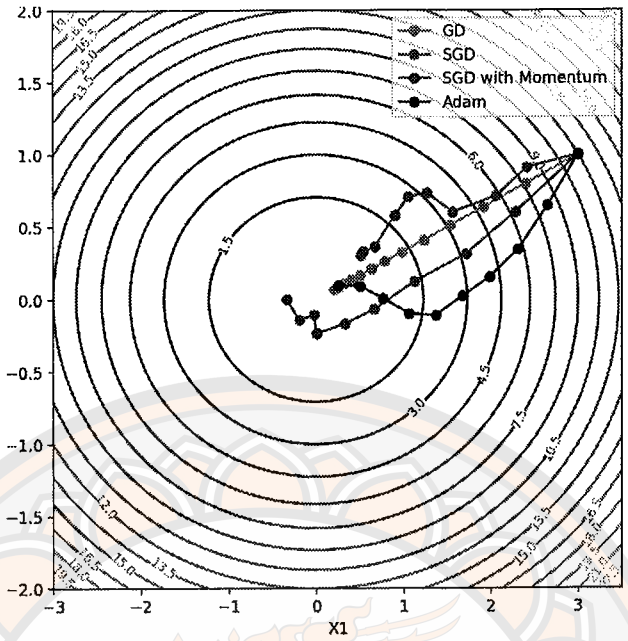


Figure 5 Comparison optimization trajectory.

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:

$$F(\mathbf{w}) \approx F(\mathbf{w}_t) + \nabla F(\mathbf{w}_t)^\top (\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^\top \mathbf{H}_t (\mathbf{w} - \mathbf{w}_t) := m_t(\mathbf{w}). \tag{2.3.8}$$

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 $m_t(\cdot)$ yields:

$$0 = \nabla m_t(\mathbf{w}) = \nabla F(\mathbf{w}_t) + \mathbf{H}_t(\mathbf{w} - \mathbf{w}_t). \tag{2.3.9}$$

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1} \nabla F(\mathbf{w}_t). \tag{2.3.10}$$

The vector $-\mathbf{H}_t^{-1}\nabla F(\mathbf{w}_t)$ is called the Newton direction, and the algorithm induced by the update formula (2.3.10) 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. Conversely, the approximation estimates the true minimizer's position if the objective function is not quadratic.

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 utilize the curvature information while simultaneously reducing computational complexity, approximated Hessian matrices are used in quasi-Newton methods, such as DFP, SR1, and Broyden-Fletcher-Goldfarb-Shanno (BFGS). In this thesis, our primary objective is to conduct a comprehensive exploration and analysis of the BFGS method, introduced by Broyden et al., with particular interest in its application within the framework of stochastic optimization techniques.

In the spirit of Newton's method (2.3.10), the BFGS iteration for minimizing a twice continuously differentiable function F has the form

$$\begin{aligned} \mathbf{z}_{t+1} &= -\eta_t \mathbf{B}_t^{-1}\nabla F(\mathbf{w}_t), \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \mathbf{z}_{t+1}, \end{aligned} \tag{2.3.11}$$

where η_t is a stepsize and $\mathbf{B}_t$ is a symmetric positive definite approximation of $\mathbf{H}_t$. This form of iteration is consistent with (2.3.10), but the distinguishing feature of a quasi-Newton scheme is that the sequence $\mathbf{H}_t$ is updated dynamically by the algorithm rather than through a second-order derivative computation at each iterate. Specifically, in the BFGS method, the new Hessian approximation is obtained by defining the iterate and gradient displacements

$$\mathbf{p}_t = \mathbf{w}_{t+1} - \mathbf{w}_t, \quad \mathbf{u}_t = \nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t), \tag{2.3.12}$$

then setting

$$\mathbf{B}_{t+1} = \mathbf{B}_t - \frac{\mathbf{B}_t \mathbf{p}_t \mathbf{p}_t^\top \mathbf{B}_t}{\mathbf{p}_t^\top \mathbf{B}_t \mathbf{p}_t} + \frac{\mathbf{u}_t \mathbf{u}_t^\top}{\mathbf{u}_t^\top \mathbf{p}_t}. \tag{2.3.13}$$

One important aspect of this update is that it ensures that the secant equation $\mathbf{B}_{t+1}\mathbf{p}_t = \mathbf{u}_t$ holds, meaning that a second-order Taylor expansion is satisfied along

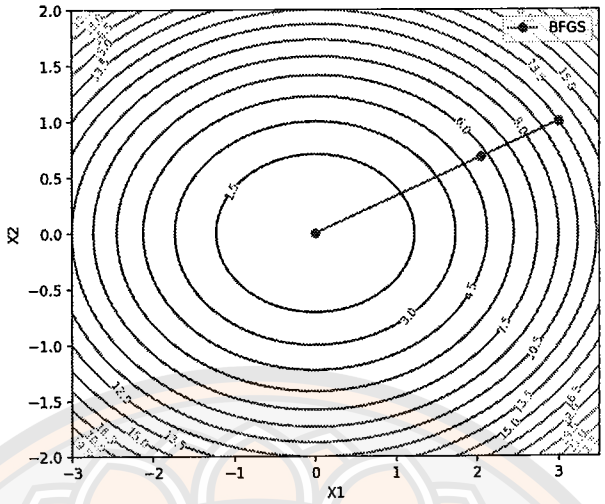


Figure 6 Optimization trajectory for BFGS method.

the most recent displacement. Moreover, this update maintains the positive definiteness property; that is, if F is strongly convex, $\nabla F(\mathbf{w}_t) \neq 0$ and $\mathbf{B}_t$ is positive definite, then $\mathbf{B}_{t+1}$ will also be positive definite. The BFGS method thus elegantly balances the accuracy of Hessian approximation with computational tractability, making it one of the most popular quasi-Newton methods in optimization. Following detailed elucidation of the BFGS update formula outlined in Equation (2.3.13), we present the pseudocode for the BFGS algorithm.

Algorithm 1 BFGS Method

Data: $\mathbf{w}_0$ as the initial point, H_0 as the initial Hessian approximation, $\epsilon > 0$ as the convergence tolerance, η_t as step size, which is computed from a line search and T as max iteration.

Set : $t = 0$ as initial time stamp

while $\|\nabla f(w_t)\| > \epsilon$ **do**

Compute search direction $\mathbf{d}_t = -B_t^{-1}\nabla f(\mathbf{w}_t)$

Update $\mathbf{z}_{t+1} = -\eta_t \mathbf{B}_t^{-1}\nabla f(\mathbf{w}_t)$, where η_t is computed from a line search procedure to satisfy the Armijo conditions;

Update $\mathbf{w}_{t+1} = \mathbf{w}_{t+1} + \mathbf{z}_{t+1}$

Define $\mathbf{p}_t = \mathbf{w}_{t+1} - \mathbf{w}_t$ and $\mathbf{u}_t = \nabla F(\mathbf{w}_{t+1}) - \nabla F(\mathbf{w}_t)$

Compute $\mathbf{B}_{t+1}$ by (2.3.13)

$t = t + 1$

end

Return: $\mathbf{w}_{t+1}$

Next, we investigate the global convergence of the BFGS method alongside a practical line search when applied to a smooth convex function from any arbitrary starting point $\mathbf{w}_0$ and utilizing an initial Hessian approximation $\mathbf{B}_0$ that is both symmetric and positively definite. Our specific assumptions regarding the objective function are precisely stated. These analyses align with the study quoted in [1].

Assumption 2.3.1.

(i) The objective function F is twice continuously differentiable.

(ii) The level set $L = \{\mathbf{w} \in \mathbb{R}^n | F(\mathbf{w}) \leq F(\mathbf{w}_0)\}$ is convex, and there exist positive constants m and M such that

$$m\|\mathbf{z}\|^2 \leq \mathbf{z}^\top \nabla^2 F(\mathbf{w})\mathbf{z} \leq M\|\mathbf{z}\|^2, \tag{2.3.14}$$

for all $\mathbf{z} \in \mathbb{R}^n$ and $\mathbf{w} \in L$.

The presentation of the BFGS method's global convergence outcome is imminent.

Theorem 2.3.2. [1] Let $\mathbf{B}_0$ be any symmetric positive definite initial matrix, and let $\mathbf{w}_0$ be a starting point for which Assumption 2.3.1 is satisfied. Then the sequence $\{\mathbf{w}_t\}$ generated by Algorithm 1 (with $\epsilon = 0$) converges to the minimizer $\mathbf{w}^*$ of F.

Having explored convergence theory, let us now focus on the practical application of the BFGS Method in handling stochastic problems while acknowledging the common drawbacks in such scenarios. One critical limitation relates to the approximated Hessian matrix $\mathbf{B}$, which might become positive semidefinite ($\mathbf{B}_t \succeq 0$) without strictly being positive definite ($\mathbf{B}_t \not\succ 0$). While direct operations from (2.3.13) assure that $\mathbf{B}_t$ remains positive definite if the initial matrix $\mathbf{B}_0$ is positive definite, the curvature estimate with $\mathbf{B}_t$ does not guarantee its positive definiteness throughout all iterations in a stochastic process. Therefore, it can not guarantee the descent direction. Thus, this aspect poses a substantial challenge in exploring the stochastic variant.

In the following, we will delve into two distinct methods aimed at addressing the challenge of ensuring the positive definiteness of the curvature estimate B_t in every iteration of a stochastic process. This includes a detailed examination of the development and nuances of the stochastic Broyden-Fletcher-Goldfarb-Shanno (BFGS) method alongside another complementary approach to highlight their respective roles and effectiveness in stochastic optimization contexts.

A challenge arises with the stochastic quasi-Newton method when the approximate Hessian matrix $\mathbf{B}_t$ tends towards values approaching zero. To address this, Mokhtari and Ribeiro [3] proposed the regularized stochastic BFGS method (RES). This innovative technique incorporates regularization to ensure that the eigenvalues of $\mathbf{B}_t$ exceed a positive constant δ. This regularization constraint explicitly defines $\mathbf{B}_t$, formulated as.

$$\hat{\mathbf{B}}_{t+1} = \hat{\mathbf{B}}_t - \frac{\hat{\mathbf{B}}_t \mathbf{p}_t \mathbf{p}_t^\top \hat{\mathbf{B}}_t}{\mathbf{p}_t^\top \hat{\mathbf{B}}_t \mathbf{p}_t} + \frac{\mathbf{y}_t \mathbf{y}_t^\top}{\mathbf{y}_t^\top \mathbf{p}_t} + \delta \mathbf{I}, \tag{2.3.15}$$

where $\mathbf{p}_t = \mathbf{w}_{t+1} - \mathbf{w}_t, \mathbf{y}_t = \hat{\mathbf{u}}_t - \delta \mathbf{p}_t$ and $\hat{\mathbf{u}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t)$. Subsequently, based on their propositions under the specified assumption, it is demonstrated that $\hat{\mathbf{B}}_{t+1}$ consistently remains greater than $\mathbf{0}$.

Proposition 2.3.3 ([3]). *If the inner product $\boldsymbol{y}_t^\top \boldsymbol{p}_t$ is positive, all the eigenvalues of $\hat{\boldsymbol{B}}_{t+1}$ are larger than δ,*

$$\hat{\boldsymbol{B}}_{t+1} \succeq \delta \boldsymbol{I}.$$

Specifically, starting at iteration $t = 0$, the updated RES is depicted as

$$\mathbf{z}_{t+1} = -\eta_t(\hat{\mathbf{B}}_t^{-1} + \Gamma\mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t, \hat{\theta}_t) \tag{2.3.16}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{z}_{t+1}, \tag{2.3.17}$$

where $\Gamma\mathbf{I}$ is the identity matrix multiplied by the positive constant Γ, serving as the identity bias term.

Diverging from the development path of RES, Indrapriyadarsini, Mahboubi, Ninomiya, and Asai introduced the accelerated stochastic BFGS method, recognized as the Nesterov-accelerated stochastic quasi-Newton method (oNAQ) [5]. Contrarily, the development of the oNAQ method takes a divergent approach from RES. This method implements an accelerated Nesterov momentum strategy, enabling more efficient stochastic gradient estimation. oNAQ adeptly balances momentum and gradient information, resulting in accelerated convergence rates and improved handling of stochastic variations within the objective function landscape. Therefore, the iteration of oNAQ is defined as follows: at iteration $t = 0$,

$$\begin{aligned}
\mathbf{g}_t &= -\hat{\mathbf{B}}_t^{-1}\hat{\mathbf{s}}(\mathbf{w}_t + \mu\mathbf{z}_t, \tilde{\theta}_t) \\
\hat{\mathbf{g}}_t &= \mathbf{g}_t/\|\mathbf{g}_t\| \\
\mathbf{z}_{t+1} &= \mu\mathbf{z}_t + \eta_t\hat{\mathbf{g}}_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \mathbf{z}_{t+1}
\end{aligned}$$

where $\hat{\mathbf{B}}_t$ is updated by

$$\hat{\mathbf{B}}_{t+1} = \hat{\mathbf{B}}_t - \frac{\hat{\mathbf{B}}_t\mathbf{v}_t\mathbf{v}_t^\top\hat{\mathbf{B}}_t}{\mathbf{v}_t^\top\hat{\mathbf{B}}_t\mathbf{v}_t} + \frac{\tilde{\mathbf{q}}_t\tilde{\mathbf{q}}_t^\top}{\tilde{\mathbf{q}}_t^\top\mathbf{v}_t} \tag{2.3.18}$$

with $\mathbf{v}_t = \mathbf{w}_{t+1} - (\mathbf{w}_t + \mu\mathbf{z}_t)$, $\tilde{\mathbf{q}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t + \mu\mathbf{z}_t, \hat{\theta}_t) + \lambda\mathbf{v}_t$, μ is a momentum coefficient, and λ is a positive parameter. Furthermore, comprehensive experiments have illustrated that oNAQ exhibits notable acceleration within the range of μ from 0 to 1, significantly diminishing computational and memory requirements. However, despite these advancements, oNAQ encounters substantial limitations under

certain circumstances. A critical observation is that μ remains fixed for each iteration, potentially leading to solution overshoots. Therefore, careful consideration is required in selecting this parameter. To reduce that limitation in practice and improve effectiveness, we introduce a novel momentum coefficient within oNAQ to reduce its limitations in practice. Additionally, we have incorporated concepts from the RES method to prevent issues with matrix B. The resulting method is the regularized stochastic Nesterov's accelerated BFGS method (RES-NAQ).

2.4 Support vector machine

In the Support Vector Machine (SVM) context, this scenario entails constructing linear classification models; our initial focus is on linearly separable problems. A problem is considered linearly separable when the training set can be accurately divided by a hyperplane, defined as $H = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{w}^\top \mathbf{x} + b = 0\}$, where $\mathbf{w} \in \mathbb{R}^n$ with $\mathbf{w} \neq 0$, $b \in \mathbb{R}$. The definition of a linearly separable problem is as follows:

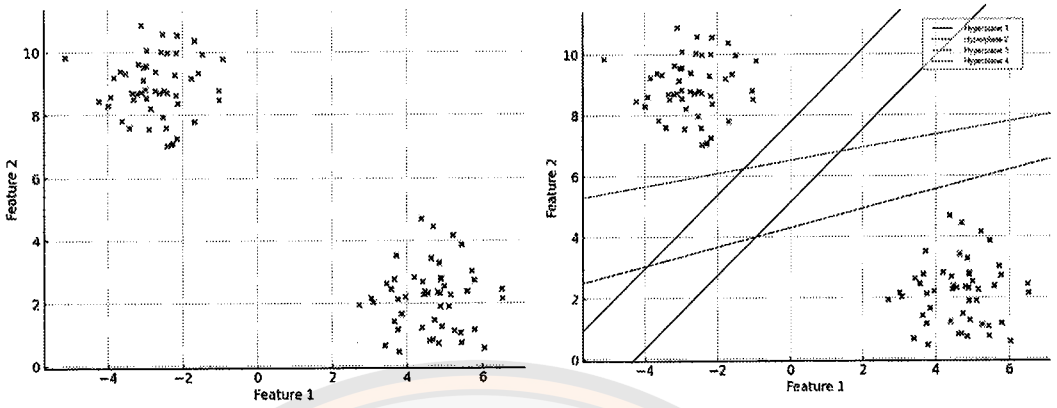
Definition 2.4.1. The training dataset $X = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{1, -1\} \mid i = 1, 2, \ldots, N\}$ is termed linearly separable if there exists $\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$, and $\varepsilon \geq 0$ such that for every sample $i = 1, 2, \ldots, m$:

$$\begin{cases} \mathbf{w}^\top \mathbf{x}_i + b > \varepsilon, & y_i = 1 \\ \mathbf{w}^\top \mathbf{x}_i + b \leq \varepsilon, & y_i = -1 \end{cases},$$

where $\mathbf{w}$ is referred to as the weight vector and b as the bias.

However, we can separate the two classes in the training data with various separating hyperplanes (as illustrated in Figure 7). Therefore, to find the best hyperplane, we select one that maximizes the distance between the two supporting hyperplanes.

Let us now formulate the problem of finding the separating hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ as an optimization problem for the variables $\mathbf{w}$ and b. Assume the separating hyperplane can be represented as $\tilde{\mathbf{w}}^\top \mathbf{x} + \tilde{b} = 0$. Considering the separating hyperplane should be equidistant between two supporting hyperplanes, these can



(a) (b)

Figure 7 Linearly separable data with various possible dividing lines.

be expressed as $\tilde{\mathbf{w}}^\top \mathbf{x} + \tilde{b} = \varepsilon$ and $\tilde{\mathbf{w}}^\top \mathbf{x} + \tilde{b} = -\varepsilon$. Let $\mathbf{w} = \frac{\tilde{\mathbf{w}}}{\varepsilon}$ and $b = \frac{\tilde{b}}{\varepsilon}$, the two support hyperplanes are equivalently expressed as

$$\mathbf{w}^\top \mathbf{x} + b = 1 \quad \text{and} \quad \mathbf{w}^\top \mathbf{x} + b = -1. \tag{2.4.1}$$

Accordingly, the expression for the separating hyperplanes becomes

$$\mathbf{w}^\top \mathbf{x} + b = 0. \tag{2.4.2}$$

It yields from the direct calculation that the distance between the two support hyperplanes, is $\frac{2}{\|\mathbf{w}\|}$. Hence, the concept of maximizing the two support hyperplanes leads to the following optimization problem for $\mathbf{w}$ and b:

$$\begin{aligned} \max_{\mathbf{w},b} \quad & \frac{2}{\|\mathbf{w}\|} \\ \text{subject to} \quad & (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i : y_i = 1, \\ & (\mathbf{w}^\top \mathbf{x}_i + b) \leq -1, \quad i : y_i = -1, \end{aligned} \tag{2.4.3}$$

or equivalently,

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, N. \end{aligned} \tag{2.4.4}$$

This formulation is the hard margin SVM, implying a convex optimization problem due to its convex objective function and linear constraints.

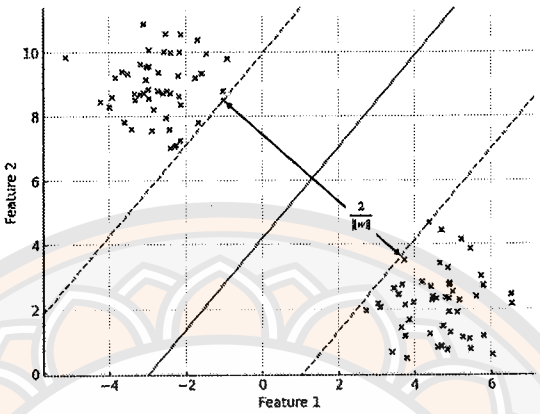
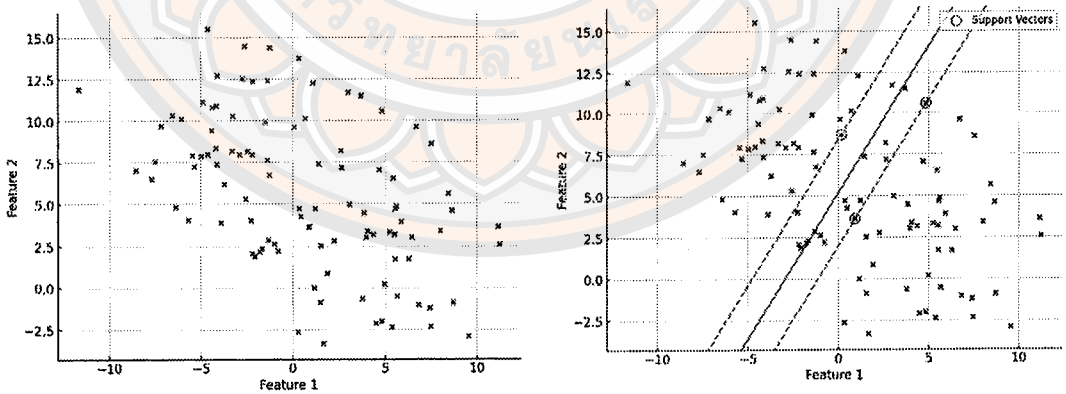


Figure 8 Separating line with maximal margin.



(a) **(b)**

Figure 9 Linearly nonseparable data with the optimal hyperplane and supporting hyperplanes.

However, A hyperplane might not correctly separate all the points for general classification tasks, mainly when the data cannot be divided. To work around this, we use two main strategies: First, we relax the strict rules for separation by using slack variables (ξ_i), which allows some flexibility. Second, we ensure these slack variables are not too big by adding a penalty in our calculations. This consequences in an optimization problem for linearly nonseparable problems, known as the soft margin SVM:

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{subject to } \ y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \ i = 1, \ldots, N \tag{2.4.5}$$

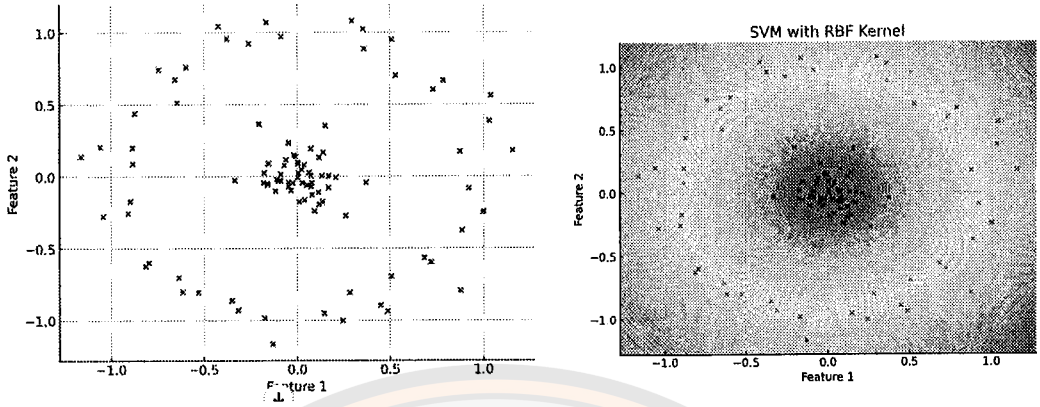
$$\xi_i \geq 0, \ i = 1, \ldots, N, \tag{2.4.6}$$

where ξ_i are slack variables and C is a trade-off parameter. Figure 9 illustrates the optimal hyperplane and supporting hyperplanes obtained through solving the problem. For consistency, this method is commonly used and is referred to as the Support Vector Machine (SVM).

The decision function takes centre stage once the solution for (2.4.5) is derived. This function, represented as $y = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$. It evaluates the input features of new data points $\mathbf{x}$ by computing the dot product of the weight vector $\mathbf{w}$ learned during training and the input features, adding the bias term b. The resulting value, passed through the sign function, determines the predicted class label. If positive, the data point belongs to one class; if negative, it belongs to the other in a binary classification scenario.

As mentioned, in cases where the data are not linearly separable, SVM models can be extended using the concept of a soft margin to allow for some misclassifications. However, when dealing with complex datasets that exhibit nonlinear patterns, allowing misclassifications may not be sufficient for achieving optimal performance. To address this challenge, SVM can use kernel tricks to map the data points to higher feature space. Instead of finding a hyperplane, the goal is to find a hypersurface that creates a kernel.

$$K(\mathbf{x}^\top, \mathbf{X}^\top)\mathbf{u} + b = 0, \tag{2.4.7}$$



(a) (b)

Figure 10 Effect of RBF kernel on non-linearly separable data.

where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$ represent matrices of input data and $K(\cdot, \cdot)$ is a given kernel mapping and $\mathbf{u} \in \mathbb{R}^N, b \in \mathbb{R}$ are weight vector and bias. The following example gives kernel functions commonly used in SVM.

Example 2.4.2. (RBF kernel) For any constant $\sigma > 0$, a Gaussian kernel or radial basis function (RBF) is the kernel k defined over $\mathbb{R}^n$ as

$$K(\mathbf{x}, \bar{\mathbf{x}}) = \exp\Big(-\gamma\|\bar{\mathbf{x}} - \mathbf{x}\|^2\Big). \tag{2.4.8}$$

Therefore, The optimization of the support vector machine with kernel trick is formulated as follows:

$$\begin{aligned} \min_{\mathbf{u},b} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & y_i(K(\mathbf{x}_i^\top, \mathbf{X}^\top)\mathbf{u} + b) \geq 1 - \xi_i,\ i = 1, \ldots, N, \\ & \xi_i \geq 0,\ i = 1, \ldots, N. \end{aligned}$$

Furthermore, Figure 10 shows an SVM with an RBF kernel applied to a dataset that is not linearly separable. It demonstrates the SVM's ability to draw a boundary that correctly separates the two classes, even though they have complex and non-linear relationships.

Further, within the framework of SVM, understanding the role of the loss function becomes integral, as it guides the model in penalizing misclassifications and optimizing the margin between classes for effective classification. Let us consider

$L : \mathbb{R} \to [0, \infty)$ as a loss function. For the classical SVM, the cost of the error is measured using the hinge loss function, which is defined as

$$L_{hinge}(u) = \begin{cases} u, & u \geq 0, \\ 0, & u < 0, \end{cases} \tag{2.4.9}$$

where $u = 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)$. The hinge loss function calculates the shortest distance between sets, focusing on penalizing only the misclassified points. However, this approach renders the model sensitive to noise. Addressing this concern, Huang et al. [17] introduced the pinball loss function into SVM, creating the Pin-SVM model. This function is formally defined as:

$$L_\tau(u) = \begin{cases} u, & u \geq 0, \\ -\tau u, & u < 0, \end{cases} \tag{2.4.10}$$

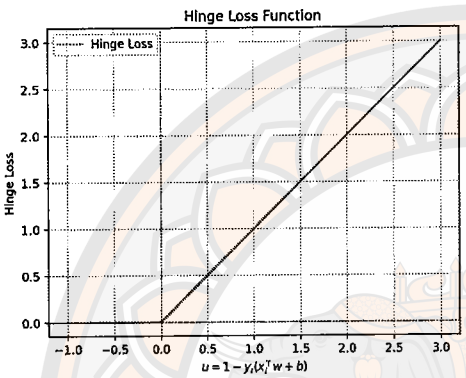
where $\tau > 0$ is hyper-parameter. The pinball loss imposes an additional penalty on the correctly classified points, indicating that $\mathcal{L}_{hinge}$ and $\mathcal{L}_\tau$ differ. However, noise insensitivity gained from pinball loss leads to losing the sparsity of the solution. In order to resolve this problem, a ϵ-insensitive pinball loss function is developed and is defined as

$$L_\tau^\epsilon(u) = \begin{cases} u - \epsilon, & u > \epsilon, \\ 0, & -\frac{\epsilon}{\tau} \leq u \leq \epsilon, \\ -\tau\left(u + \frac{\epsilon}{\tau}\right), & u < -\frac{\epsilon}{\tau}, \end{cases} \tag{2.4.11}$$

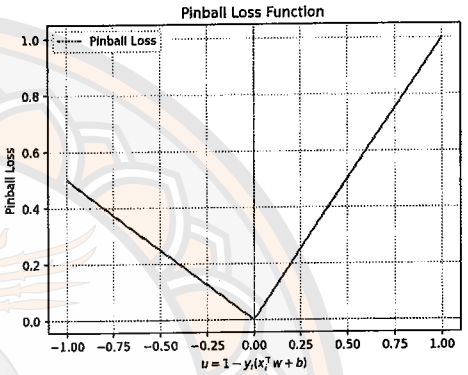
where $\tau > 0, \epsilon > 0$ are hyper-parameters. Just recently, a brand-new loss function from a concept of development resembling the prior loss function is presented, called the generalized pinball loss function. The definition is given as follows

$$L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(u) = \begin{cases} \tau_1(u - \frac{\epsilon_1}{\tau_1}), & u > \frac{\epsilon_1}{\tau_1}, \\ 0, & -\frac{\epsilon_2}{\tau_2} \leq u \leq \frac{\epsilon_1}{\tau_1}, \\ -\tau_2(u + \frac{\epsilon_2}{\tau_2}), & u < -\frac{\epsilon_2}{\tau_2}, \end{cases} \tag{2.4.12}$$

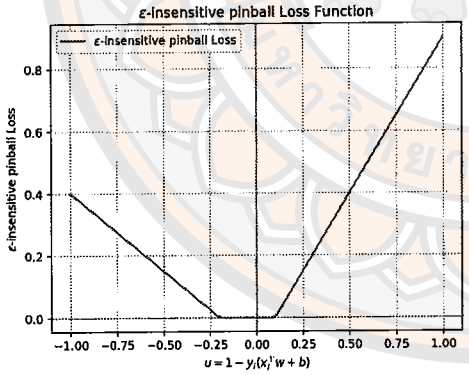
where $\tau_1 > 0, \tau_2 > 0, \epsilon_1 > 0, \epsilon_2 > 0$ are hyper-parameters. The generalized pinball loss function can maintain the insensitivity to noise, stability of resampling, and sparsity and is a generalization of the ϵ-insensitive pinball loss function when we set $\tau_1 = 1$, $\tau_2 = \tau$ and $\epsilon_1 = \epsilon_2 = \epsilon$.



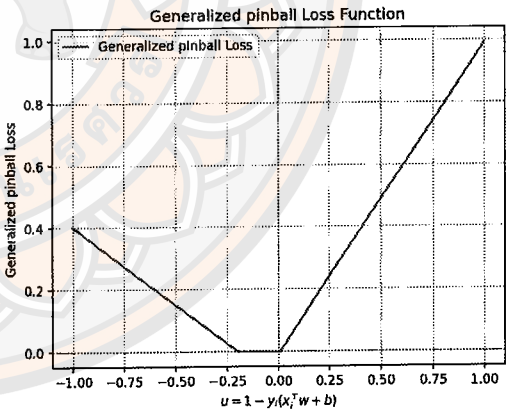
(a) Hinge loss function



(b) Pinball loss function



(c) ϵ-insensitive pinball loss functio



(d) Generalized pinball loss function

Figure 11 Comparative visualization of four loss functions.

Looking through these loss functions, we can express the SVM problem as the unconstrained optimization problem, represented by

$$\min_{\mathbf{w},b} \ \frac{1}{2}(\|\mathbf{w}\|^2 + b^2) + \frac{c}{m}\sum_{i=1}^{m} L(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)), \tag{2.4.13}$$

where L can be replaced by the hinge loss function, pinball loss function, ϵ-insensitive pinball loss function or generalized pinball loss function. This problem is strongly convex, ensuring a unique solution exists. However, within the problem (2.4.13), a component incorporates a loss function that is not differentiable, rendering it non-differentiable. Consequently, this limitation hinders the direct applicability of efficient algorithms like the quasi-Newton BFGS method in solving. As a result, in this thesis, we are interested in searching for a smooth approximation function of the generalized pinball loss function and using a new smooth approximate generalized pinball loss function instead of a generalized pinball loss function. This approach allows us to employ the quasi-Newton algorithm to solve this optimization problem, enabling a more efficient solution methodology.

On the other hand, solving the SVM objective function involves dealing with a large-scale problem, increasing computational complexity and restricting its suitability for large-scale data analysis. Subsequently, various SVM-based algorithms were proposed to address this limitation. One such method is the Least Squares Twin SVM (LSTBSVM), introduced by Yitian Xu et al. LSTBSVM serves as a binary classifier, seeking a pair of non-parallel hyperplanes for pattern classification. These hyperplanes, expressed as

$$\mathbf{x}^\top \mathbf{w}_1 + b_1 = 0 \ \text{ and } \ \mathbf{x}^\top \mathbf{w}_2 + b_2 = 0 \tag{2.4.14}$$

are determined to be proximal to samples of one class while farthest from samples of the other class.

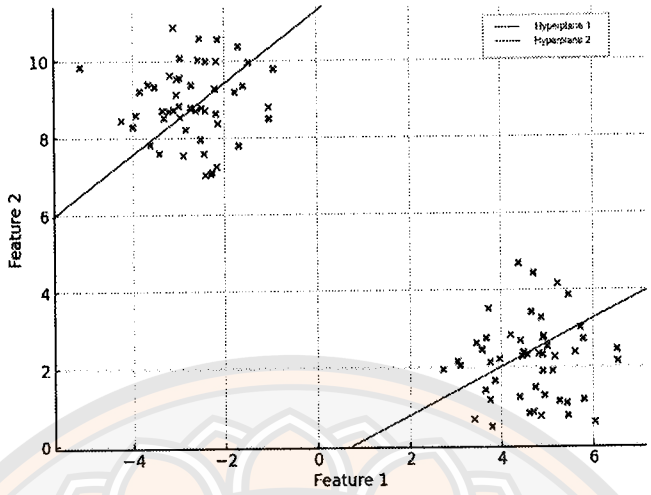


Figure 12 The non-parallel hyperplanes of LSTBSVM.

Figure12 illustrates the non-parallel hyperplanes of a LSTBSVM, achieved through the solution of problems defined as:

$$\min_{\mathbf{w}_1,b_1,q_j} \quad \frac{c_3}{2}(\|\mathbf{w}_1\|^2 + b_1^2) + \frac{1}{2}\sum_{i=1}^{m_1}(\mathbf{w}_1^\top x_i^+ + b_1)^2 + \frac{c_1}{2}\sum_{j=1}^{m_2}(q_j)^2$$
$$\text{s.t.} \quad q_j = 1 + (\mathbf{w}_1^\top x_j^- + b_1), \quad j = 1, 2, \ldots, m_2 \tag{2.4.15}$$

and

$$\min_{\mathbf{w}_2,b_2,p_i} \quad \frac{c_4}{2}(\|\mathbf{w}_2\|^2 + b_2^2) + \frac{1}{2}\sum_{j=1}^{m_2}(\mathbf{w}_2^\top x_j^- + b_2)^2 + \frac{c_2}{2}\sum_{i=1}^{m_1}(p_i)^2$$
$$\text{s.t.} \quad p_i = 1 - (\mathbf{w}_2^\top x_i^+ + b_2), \quad i = 1, 2, \ldots, m_1, \tag{2.4.16}$$

where m_1 and m_2 represent the quantities of data points belonging to classes $+1$ and -1, respectively and $m_1 + m_2 = m$. Additionally, c_1, c_2, c_3, and c_4 are positive penalty parameters, while q_j and p_i denotes a slack variable. For convenience, we only discuss problem (2.4.15), and problem (2.4.16) is similar. Here, the objective function of the problem (2.4.15) contains three terms. The first term is the regularization term intended to prevent over-fitting. The second term aims to bring positive points close to the positive decision hyperplane. Lastly, the final term is minimizing errors resulting from the requirement of positioning the positive decision hyperplane exactly 1 unit away from points in the negative class. The classification of new data point as $+1$ or -1 depends on its proximity to these determined

hyperplanes, a relationship expressed by the equation

$$y = \arg\min_{i=1,2} \frac{|\mathbf{w}_i^\top \mathbf{x} + b_i|}{\|\mathbf{w}_i\|}. \tag{2.4.17}$$

This decision is determined by evaluating the distance of the sample point to these optimized hyperplanes, subsequently assigning the appropriate class label based on this proximity.

Compared to SVM, LSTBSVM solves two quadratic programming problems (QPPs) with equality constraints of smaller size, reducing the training cost. This efficiency makes LSTBSVM faster than SVM. However, LSTBSVM penalizes sample points from opposing classes equally without considering the location information of points. As a result, it makes the models sensitive to outliers or noise, which means that the classification hyperplanes can easily move to outliers or noise samples. To address this issue, we introduce a new loss function into the LSTBSVM, which considers the locations of sample points, providing a more refined approach to handling this problem.

2.5 Convolution network

Classifiers typically transform raw data into a model to predict categories. For images, this involves converting their matrix format into vectors. However, this conversion can lead to overly complex vectors that lack certain essential details. The solution lies in image feature extraction, a technique that breaks down and simplifies data into smaller, more manageable pieces, making it easier to analyze and classify. Extracting features manually from images is a daunting and time-consuming task. Automated methods, such as Convolutional Neural Networks (CNNs), offer a significant advantage by identifying features directly from images. CNNs are especially effective for image classification as they combine feature extraction and classification into one integrated process.

The convolutional layers in a Convolutional Neural Network (CNN) systematically apply learned filters to input images. This process creates feature maps that highlight specific features within the input. Pooling layers then down-sample these feature maps, summarizing the features within patches of the map. The two common pooling methods are average pooling and max pooling, each providing a

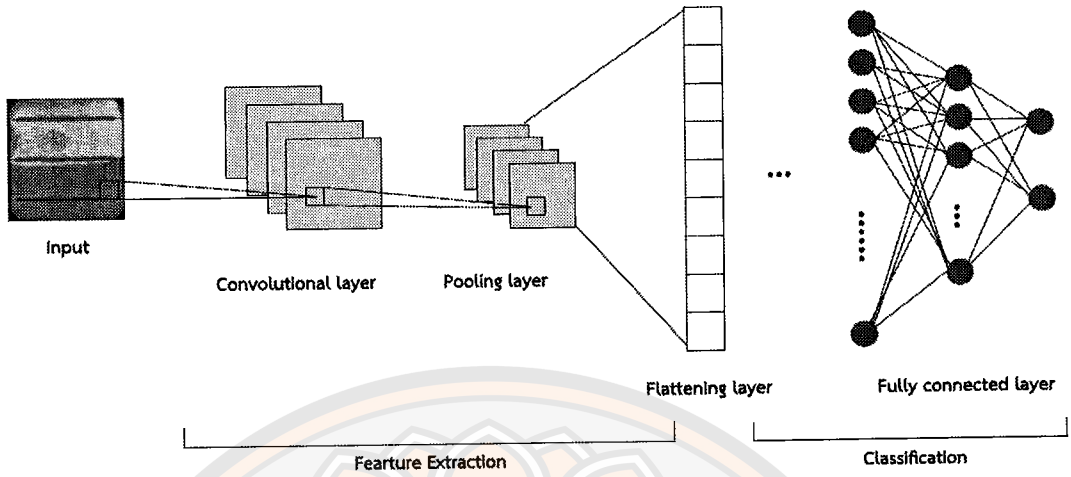


Figure 13 Convolution network.

unique way to reduce data size. The flattening layer converts the data into a single vector to facilitate easier integration into the model sequence. Following the feature extraction phase, this prepared data is passed to a classification model. This model predicts the label that best represents the primary content observed in the image. Traditionally, a CNN employs dense multi-layer perceptrons (fully connected layers) as the classification model, as pointed out in Figure 13. CNN architectures like VGG, ResNet and InceptionV3 have become renowned for their accuracy in image classification. These models have established new standards in the field of image recognition.

Generally, we can adapt the classification model to utilize other models, such as SVM. This adaptation involves feature extraction using pre-trained models from these well-known architectures instead of traditional feature extraction methods commonly used in machine learning studies. The features extracted in this manner serve as the primary input for an SVM classifier, thereby facilitating efficient classification. Numerous studies have employed this technique to classify various image data, including X-ray and texture images.

Inspired by these developments, we propose a novel approach that leverages SVMs suitable for real-world image classification challenges. This method aims to streamline the classification process and enhance accuracy, offering a promising new direction for image analysis in diverse applications. The flow diagram of this method is shown in Figure 15.

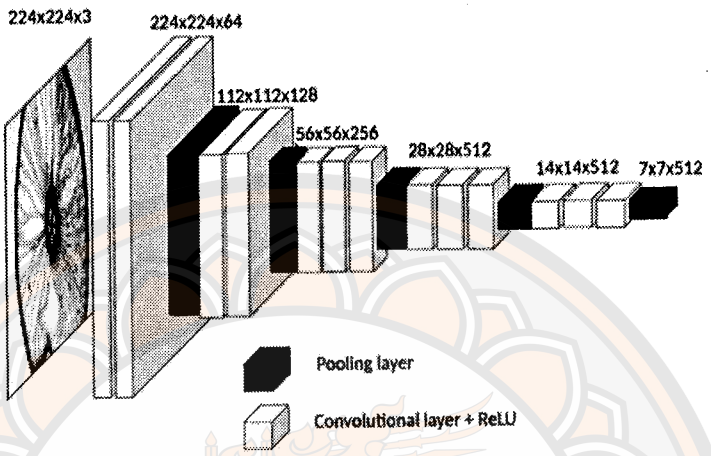


Figure 14 Architecture of convolution layer in VGG16.

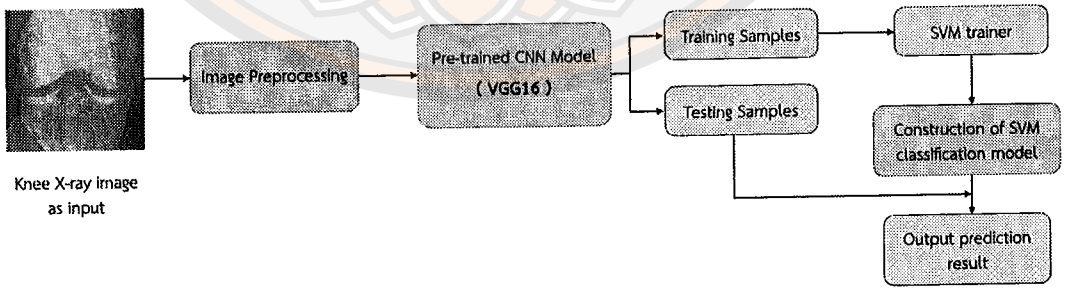


Figure 15 Flow diagram of the proposed image recognition system.

Finally, note that the code in this thesis requires Python version 3.8.10, along with NumPy version 1.20.3, Pandas version 1.3.2, and scikit-learn version 0.24.2.

CHAPTER III

THE REGULARIZED STOCHASTIC NESTEROV'S ACCELERATED QUASI-NEWTON METHOD WITH APPLICATIONS

This chapter presents a novel method for solving a strongly convex unconstrained optimization problem: a regularized stochastic Nesterov's accelerated quasi-Newton method. This method notably enhances convergence speed while mitigating the near-singularity issue often encountered in approximate Hessian updates within the stochastic BFGS method. Our discussion will begin with the idea of the proposed method. Following that, we will present the convergence theorem, validating its stability, culminating in the practical demonstration of its efficacy through solving classification problems.

As mentioned in the last section (see Section 2.3), the parameter μ in the oNAQ algorithm is a fixed value that is computed in every iteration. Accordingly, oNAQ may fail to satisfy the descent property (i.e., $F(\mathbf{w}_t + \mu \mathbf{z}_t) \not< F(\mathbf{w}_t)$ for some iteration t). Therefore, it can be the potential for an update overshoot. In the following discussion, we will delve into this scenario by applying the oNAQ algorithm to solve the Support Vector Machine (SVM) problem. We will analyse the algorithm's performance across different parameter settings of μ and its behaviour in terms of the descent property.

Let us consider a training dataset $T = \{\theta_i\}_{i=1}^{1000} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{1000}$, where $\mathbf{x}_i \in \mathbb{R}^{40}$ for $i = 1, 2, \ldots, 1000$. Each component of $\mathbf{x}_1, \ldots, \mathbf{x}_{500}$ and $\mathbf{x}_{501}, \ldots, \mathbf{x}_{1000}$ are uniformly distributed over the intervals $[-0.2, 0.8]$ and $[-0.8, 0.2]$, respectively. The labels are assigned as $y_i = 1$ for $i = 1, \ldots, 500$ and $y_i = -1$ for $i = 501, \ldots, 1000$. Therefore, the SVM problem is defined by:

$$\begin{aligned} F(\mathbf{w}) &:= \frac{1}{1000} \sum_{i=1}^{1000} f(\mathbf{w}, \theta_i) \\ &:= \frac{10^{-4}}{2} \|\mathbf{w}\|^2 \end{aligned}$$

$$+\frac{1}{1000}\left(\sum_{i=1}^{500}\max(0,1-(\mathbf{w}^\top\mathbf{x}_i))^2+\sum_{i=501}^{1000}\max(0,1+(\mathbf{w}^\top\mathbf{x}_i))^2\right), \tag{3.0.1}$$

where $f(\mathbf{w},\theta_i)=\frac{10^{-4}}{2}\|\mathbf{w}\|^2+\max(0,1+y_i(\mathbf{w}^\top\mathbf{x}_i))^2$ and $\mathbf{w}\in\mathbb{R}^{40}$. In this experiment, we use stepsize is $\epsilon_0 T_0/(T_0+t)$, with $\epsilon_0=10^{-3}$ and $T_0=10^8$ and minibatch size L for iteration t is 5. The experiment's results are shown in Figure 16 and Figure 17.

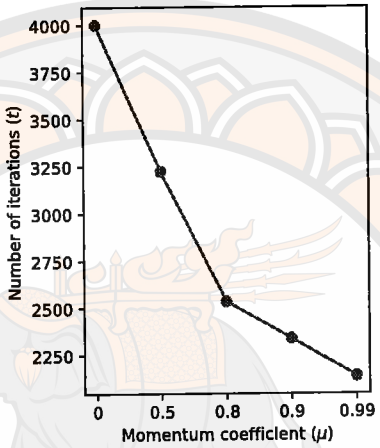


Figure 16 The number of iterations satisfied the descent property for different values of μ.

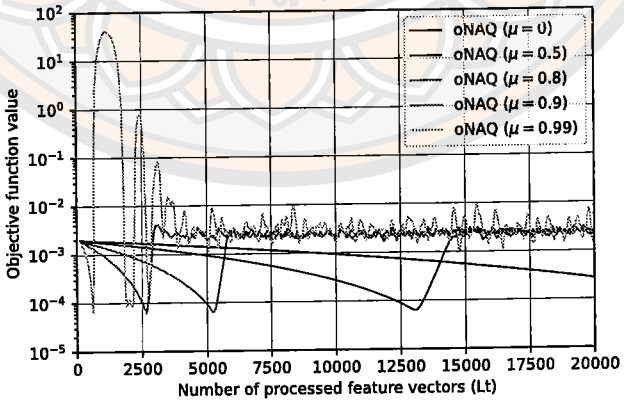


Figure 17 The performance of oNAQ with different values of μ (The number of feature vectors processed is determined by the product between the sample size used to compute stochastic gradients (L) and iteration index (t)).

In Figure 16, it is observed that as μ approaches 1, the frequency of iterations satisfying the descent property decreases. Initially, this leads to a fast decline in the values of the objective function. However, subsequent oscillations and minimal overshoots become evident, as depicted in Figure 17. Conversely, when μ is closer to 0, there is an increase in the iterations that satisfy the descent property, which contributes to reduced oscillations and prevents overshooting the minimum, as illustrated in Figure 17. Nonetheless, achieving the objective function value of 10^{-4} requires more iterations. This indicates a trade-off between convergence speed and the method's stability. The observed oscillation challenges the method's long-term effectiveness, emphasizing the need for a balanced approach in selecting μ to optimize performance and stability.

Based on the study mentioned above, we have observed that adjusting μ under $F(\mathbf{w}_t + \mu\mathbf{z}_t) < F(\mathbf{w}_t)$ for each iteration t significantly affects performance. Therefore, a new momentum coefficient is defined as follows.

$$\mu_t = \begin{cases} \mu & \text{if } F(\mathbf{w}_t + \mu\mathbf{z}_t) < F(\mathbf{w}_t), \\ 0 & \text{if } F(\mathbf{w}_t + \mu\mathbf{z}_t) \geq F(\mathbf{w}_t), \end{cases} \tag{3.0.2}$$

where $\mu \in [0, 1)$, $\mathbf{w}_t, \mathbf{z}_t \in \mathbb{R}^n$, and F is defined by (2.3.3) or (2.3.4). Thus, for every iteration t, we have

$$F(\mathbf{w}_t + \mu_t\mathbf{z}_t) \leq F(\mathbf{w}_t). \tag{3.0.3}$$

Furthermore, we also include the regularized parameters to update the approximate Hessian matrix $\hat{\mathbf{B}}_t$ similarly to RES to prevent the near-singular problem. Therefore, at iteration t and initial $\mathbf{z}_0 = \mathbf{0}$, the proposed RES-NAQ method is updated by

$$\begin{aligned} \mathbf{z}_{t+1} &= \mu_t\mathbf{z}_t - \epsilon_t(\hat{\mathbf{B}}_t^{-1} + \Gamma\mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t\mathbf{z}_t, \hat{\theta}_t), \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \mathbf{z}_{t+1}, \end{aligned} \tag{3.0.4}$$

where $\epsilon_t > 0$ is the stepsize and $\hat{\mathbf{B}}_t$ is an approximate Hessian matrix updated by

$$\hat{\mathbf{B}}_{t+1} = \hat{\mathbf{B}}_t - \frac{\hat{\mathbf{B}}_t\mathbf{v}_t\mathbf{v}_t^\top\hat{\mathbf{B}}_t}{\mathbf{v}_t^\top\hat{\mathbf{B}}_t\mathbf{v}_t} + \frac{\tilde{\mathbf{r}}_t\tilde{\mathbf{r}}_t^\top}{\tilde{\mathbf{r}}_t^\top\mathbf{v}_t} + \delta\mathbf{I}, \tag{3.0.5}$$

where $\mathbf{v}_t = \mathbf{w}_{t+1} - (\mathbf{w}_t + \mu_t \mathbf{z}_t)$ and $\tilde{\mathbf{r}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t) - \delta \mathbf{v}_t$. Moreover, we define the stepsize $\epsilon_t = \epsilon_0 T_0/(T_0 + t)$, where ϵ_0 and $T_0 > 0$, satisfying the conditions

$$\sum_{t=0}^{\infty} \epsilon_t = \infty, \quad \text{and} \quad \sum_{t=0}^{\infty} \epsilon_t^2 < \infty. \tag{3.0.6}$$

The algorithmic details of the proposed RES-NAQ method are presented in Algorithm 2.

Algorithm 2 RES-NAQ

Require: Variable $\mathbf{w}_0$, Hessian approximation $\hat{\mathbf{B}}_0 \succ \delta\mathbf{I}$, $\mu \in [0,1)$ and $\mathbf{z}_0 = 0$.
Determines $F(\mathbf{w}) := \mathbb{E}[f(\mathbf{w},\theta)]$ or $F(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w},\theta_i)$.

1: **for** $t = 0, 1, 2, \ldots$ **do**

2: Compute momentum coefficient

$$\mu_t = \begin{cases} \mu & \text{if } F(\mathbf{w}_t + \mu\mathbf{z}_t) < F(\mathbf{w}_t), \\ 0 & \text{if } F(\mathbf{w}_t + \mu\mathbf{z}_t) \geq F(\mathbf{w}_t). \end{cases}$$

3: Acquire L independent sample $\hat{\theta} = \{\hat{\theta}_l\}_{l=1}^{L}$.

4: Compute stochastic gradient

$$\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t\mathbf{z}_t, \hat{\theta}_t) := \frac{1}{L}\sum_{l=1}^{L} \nabla f(\mathbf{w}_t + \mu_t\mathbf{z}_t, \theta_{t,l}).$$

5: Update

$$\mathbf{z}_{t+1} = \mu_t\mathbf{z}_t - \epsilon_t(\hat{\mathbf{B}}_t^{-1} + \Gamma\mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t\mathbf{z}_t, \hat{\theta}_t),$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{z}_{t+1}.$$

6: Compute stochastic gradient

$$\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) = \frac{1}{L}\sum_{l=1}^{L} \nabla f(\mathbf{w}_{t+1}, \theta_{t,l}).$$

7: Compute modified stochastic gradient variation

$$\tilde{\mathbf{r}}_t = \hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t + \mu_t\mathbf{z}_t, \hat{\theta}_t) - \delta\mathbf{v}_t.$$

8: Update Hessian approximation matrix

$$\hat{\mathbf{B}}_{t+1} = \hat{\mathbf{B}}_t - \frac{\hat{\mathbf{B}}_t\mathbf{v}_t\mathbf{v}_t^\top\hat{\mathbf{B}}_t}{\mathbf{v}_t^\top\hat{\mathbf{B}}_t\mathbf{v}_t} + \frac{\tilde{\mathbf{r}}_t\tilde{\mathbf{r}}_t^\top}{\tilde{\mathbf{r}}_t^\top\mathbf{v}_t} + \delta\mathbf{I}.$$

9: **end for**

3.1 Convergence analysis

In this section, we delve into the convergence characteristics of the RES-NAQ method. The convergence analysis of the method starts with the definition of the stochastic objective function, $\hat{f}(\mathbf{w}, \hat{\theta})$, based on realizations $\hat{\theta} = \{\hat{\theta}_l\}_{l=1}^{L}$. These realizations correspond to a set of samples $\{\mathbf{x}_l\}_{l=1}^{L}$, chosen uniformly at random from the dataset and are formulated as follows:

$$\hat{f}(\mathbf{w}, \hat{\theta}) := \frac{1}{L} \sum_{l=1}^{L} f(\mathbf{w}, \hat{\theta}_l). \tag{3.1.1}$$

Given this definition of the stochastic objective function $\hat{f}(\mathbf{w}, \hat{\theta})$ and considering the previously established definition that $F(\mathbf{w}) := \mathbb{E}[f(\mathbf{w}, \theta)]$ in (2.3.3), it follows that:

$$F(\mathbf{w}) = \mathbb{E}_{\theta}[\hat{f}(\mathbf{w}, \hat{\theta})]. \tag{3.1.2}$$

Our objective is to demonstrate that, over time, the sequence of variable iterates converges to the optimal solution $\mathbf{w}^*$. We recall the process underlying our proposed method to substantiate this claim and introduce the following assumptions.

During the iteration process at index t, the RES-NAQ method updates according to the following procedure:

$$\mathbf{z}_{t+1} = \mu_t \mathbf{z}_t - \epsilon_t (\hat{\mathbf{B}}_t^{-1} + \Gamma \mathbf{I}) \hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{z}_{t+1},$$

where the stochastic gradient $\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)$ is defined as:

$$\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t) := \nabla \hat{f}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t) = \frac{1}{L} \sum_{l=1}^{L} \nabla f(\mathbf{w}_t + \mu_t \mathbf{z}_t, \theta_{t,l}). \tag{3.1.3}$$

Assumption 3.1.1. The stochastic objective function $\hat{f}(\mathbf{w}, \hat{\theta})$ is twice differentiable and the eigenvalues of the stochastic Hessian $\hat{\mathbf{H}}(\mathbf{w}, \hat{\theta})$ are bounded between constants $0 < \hat{m}$ and $\hat{M} < \infty$ for all random variables $\hat{\theta}$,

$$\hat{m}\mathbf{I} \preceq \hat{\mathbf{H}}(\mathbf{w}, \hat{\theta}) \preceq \hat{M}\mathbf{I}. \tag{3.1.4}$$

Assumption 3.1.1 similarly applies to the function $F(\mathbf{w})$, resulting in comparable eigenvalue bounds. Indeed, the Hessian $\mathbf{H}(\mathbf{w})$ is equal to $\mathbb{E}_{\theta}[\hat{\mathbf{H}}(\mathbf{w}, \hat{\theta})]$ due to

the linearity of the expectation operator, as indicated in the condition expressed by (3.1.2). This observation, combined with the bounds presented in (3.1.4), leads to the conclusion that there exist constants $m \geq \hat{m}$ and $M \leq \hat{M}$, ensuring

$$\hat{m}\mathbf{I} \preceq m\mathbf{I} \preceq \mathbf{H}(\mathbf{w}) \preceq M\mathbf{I} \preceq \hat{M}\mathbf{I}. \tag{3.1.5}$$

Assumption 3.1.2. For any iteration t, there exists a positive constant S such that

$$\mathbb{E}_\theta[\|\hat{s}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)\|^2 | \mathbf{w}_t + \mu_t \mathbf{z}_t] \leq S^2.$$

Assumption 3.1.3. The constant δ is smaller than the smallest Hessian eigenvalue $\hat{m}$, i.e., $\delta < \hat{m}$.

The following lemma demonstrates that the approximation Hessian matrix $\hat{\mathbf{B}}_{t+1}$ of RES-NAQ possesses positive definite properties.

Lemma 3.1.4. Let $\mathbf{v}_t$ and $\tilde{\mathbf{r}}_t$ be defined by $\mathbf{w}_{t+1} - (\mathbf{w}_t + \mu_t \mathbf{z}_t)$ and $\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t) - \delta \mathbf{v}_t$, respectively. Under Assumptions 3.1.1, and 3.1.3, the following property holds:

$$\tilde{\mathbf{r}}_t^\top \mathbf{v}_t > 0.$$

Proof. As per (3.1.4) in Assumption 3.1.1, the eigenvalues of the stochastic Hessian are bounded below by $\hat{m}$. This condition is equivalent to stating that the stochastic objective functions $\hat{f}(\mathbf{w}_t, \hat{\theta}_t)$, associated with samples $\hat{\theta}_t$, are $\hat{m}$-strongly convex with respect to $\mathbf{w}$. Therefore, by (2.1.9), it follows that

$$\left(\nabla \hat{f}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \nabla \hat{f}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)\right)^\top (\mathbf{w}_{t+1} - (\mathbf{w}_t + \mu_t \mathbf{z}_t)) \geq \hat{m}\|\mathbf{w}_{t+1} - (\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2.$$

From (3.1.3), we obtain that

$$\left(\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)\right)^\top (\mathbf{w}_{t+1} - (\mathbf{w}_t + \mu_t \mathbf{z}_t)) \geq \hat{m}\|\mathbf{w}_{t+1} - (\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2. \tag{3.1.6}$$

By definitions of $\mathbf{v}_t$, $\tilde{\mathbf{r}}_t$ and (3.1.6), we have

$$\tilde{\mathbf{r}}_t^\top \mathbf{v}_t = \left(\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t) - \delta \mathbf{v}_t\right)^\top \mathbf{v}_t$$

$$
\begin{aligned}
&= \Big(\hat{\mathbf{s}}(\mathbf{w}_{t+1}, \hat{\theta}_t) - \hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)\Big)^\top \mathbf{v}_t - \delta \|\mathbf{v}_t\|^2 \\
&\geq \hat{m}\|\mathbf{v}_t\|^2 - \delta\|\mathbf{v}_t\|^2 \\
&= (\hat{m} - \delta)\|\mathbf{v}_t\|^2.
\end{aligned}
$$

So, by Assumption (3.1.3), we conclude that

$$\tilde{\mathbf{r}}_t^\top \mathbf{v}_t > 0,$$

for any iteration t. Thus, the proof is complete. □

Remark 3.1.5. If we take Lemma 3.1.4 with Proposition 2.3.3, then all eigenvalues of $\hat{\mathbf{B}}_{t+1}$ are larger than δ, guaranteeing that $\hat{\mathbf{B}}_{t+1}$ is positive definite for every iteration t. It follows that $0 \prec \delta\mathbf{I} \prec \hat{\mathbf{B}}_t$ for any iteration t. So it gives us that all the eigenvalues of $\hat{\mathbf{B}}_t^{-1}$ are between 0 and $\frac{1}{\delta}$. This implies that for the positive constant Γ,

$$\Gamma\mathbf{I} \prec \hat{\mathbf{B}}_t^{-1} + \Gamma\mathbf{I} \prec \Big(\frac{1}{\delta} + \Gamma\Big)\mathbf{I}. \tag{3.1.7}$$

Therefore, matrix $\hat{\mathbf{B}}_t^{-1} + \Gamma\mathbf{I}$ is positive definite.

Subsequently, we will demonstrate the convergence results for RES-NAQ. To establish these results, we rely on the following lemma.

Lemma 3.1.6. Assume that Assumptions 3.1.1 and 3.1.2 are satisfied. Then, for any iteration t,

$$\mathbb{E}[F(\mathbf{w}_{t+1} + \mu_{t+1}\mathbf{z}_{t+1})|\mathbf{w}_t + \mu_t\mathbf{z}_t] \leq F(\mathbf{w}_t + \mu_t\mathbf{z}_t) - \epsilon_t\Gamma\|\nabla F(\mathbf{w}_t + \mu_t\mathbf{z}_t)\|^2 + K\epsilon_t^2, \tag{3.1.8}$$

where $K := \frac{M}{2}\Big(\frac{1}{\delta} + \Gamma\Big)^2 S^2$.

Proof. It follows from Assumption 3.1.1(2) that the eigenvalues of the Hessian $\mathbf{H}(\mathbf{w}) = \mathbb{E}_\theta[\hat{\mathbf{H}}(\mathbf{w}, \hat{\theta})]$ are bounded between $0 < m$ and $M < \infty$ as stated in (3.1.5). Taking Taylor's expansion of $F(\mathbf{w})$ around $\mathbf{w} = \mathbf{w}_t + \mu_t\mathbf{z}_t$ and using the upper bound in the Hessian eigenvalues we can write, we have

$$F(\mathbf{w}_{t+1}) \leq F(\mathbf{w}_t + \mu_t\mathbf{z}_t) + \nabla F(\mathbf{w}_t + \mu_t\mathbf{z}_t)^\top(\mathbf{w}_{t+1} - (\mathbf{w}_t + \mu_t\mathbf{z}_t))$$

$$+\frac{M}{2}\|\mathbf{w}_{t+1}-(\mathbf{w}_t+\mu_t\mathbf{z}_t)\|^2. \tag{3.1.9}$$

By definition of $\mathbf{w}_{t+1}$ updated in (3.0.4) we can write the difference of two consecutive variables $\mathbf{w}_{t+1}-(\mathbf{w}_t+\mu_t\mathbf{z}_t)$ as $-\epsilon_t(\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)$. Making this substitution in (3.1.9), we have

$$\begin{aligned}
F(\mathbf{w}_{t+1}) &\leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t)-\epsilon_t\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)^\top\Big((\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\Big)\\
&\quad+\frac{\epsilon_t^2M}{2}\|(\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\|^2\\
&\leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t)-\epsilon_t\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)^\top\Big(\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\Big)\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\\
&\quad+\frac{\epsilon_t^2M}{2}\|\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\|^2\|\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\|^2.
\end{aligned}\tag{3.1.10}$$

The third term on the right-hand side of (3.1.10) is derived from the principle that the 2-norm of a product is not larger than the product of the 2-norms. Then, by taking the expectation conditioned on $\mathbf{w}_t+\mu_t\mathbf{z}_t$ in (3.1.10), and noting that the Hessian approximation $\hat{\mathbf{B}}_t^{-1}$ becomes deterministic when $\mathbf{w}_t+\mu_t\mathbf{z}_t$ is given, we can express (3.1.10) as follows:

$$\begin{aligned}
&\mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t]\\
&\quad\leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t)-\epsilon_t\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)^\top\Big(\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\Big)\mathbb{E}[\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)|\mathbf{w}_t+\mu_t\mathbf{z}_t]\\
&\qquad+\frac{\epsilon_t^2M}{2}\|\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\|^2\mathbb{E}[\|\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\|^2|\mathbf{w}_t+\mu_t\mathbf{z}_t].
\end{aligned}$$

By using the fact that $\mathbb{E}[\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)|\mathbf{w}_t+\mu_t\mathbf{z}_t]=\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)$, we have

$$\begin{aligned}
&\mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t]\\
&\quad\leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t)-\epsilon_t\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)^\top\Big(\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\Big)\Big(\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)\Big)\\
&\qquad+\frac{\epsilon_t^2M}{2}\|\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\|^2\mathbb{E}[\|\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\|^2|\mathbf{w}_t+\mu_t\mathbf{z}_t].
\end{aligned}\tag{3.1.11}$$

Since $\hat{\mathbf{B}}_t$ is positive definite, we have $\hat{\mathbf{B}}_t^{-1}$ is also positive semidefinite. In turn, this implies that all the eigenvalues of $\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}$ are not smaller than Γ since $\Gamma\mathbf{I}$ increases all the eigenvalues of $\hat{\mathbf{B}}_t^{-1}$ by Γ. This lower bound for the eigenvalues of $\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}$ implies that

$$\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)^\top\Big(\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\Big)\Big(\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)\Big)\geq\Gamma\|\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)\|^2.$$

Thus, from (3.1.11), we have

$$\mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t] \leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t) - \epsilon_t\Gamma\|\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)\|^2 + \frac{\epsilon_t^2 M}{2}\|\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}\|^2\mathbb{E}[\|\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\|^2|\mathbf{w}_t+\mu_t\mathbf{z}_t].$$

Notice that, as stated in (3.1.7), $\frac{1}{\delta}+\Gamma$ is an upper bound for the eigenvalues of $\hat{\mathbf{B}}_t^{-1}+\Gamma\mathbf{I}$. Therefore,

$$\mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t] \leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t) - \epsilon_t\Gamma\|\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)\|^2 + \frac{\epsilon_t^2 M}{2}\left(\frac{1}{\delta}+\Gamma\right)^2\mathbb{E}[\|\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\|^2|\mathbf{w}_t+\mu_t\mathbf{z}_t].$$

Further observe that the second moment of the norm of the stochastic gradient is bounded by $\mathbb{E}[\|\hat{\mathbf{s}}(\mathbf{w}_t+\mu_t\mathbf{z}_t,\hat{\theta}_t)\|^2|\mathbf{w}_t+\mu_t\mathbf{z}_t] \leq S^2$, as stated in Assumption 3.1.2, we have that

$$\begin{aligned}&\mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t] \\ &\quad\leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t) - \epsilon_t\Gamma\|\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)\|^2 + \frac{\epsilon_t^2 M}{2}\left(\frac{1}{\delta}+\Gamma\right)^2 S^2.\end{aligned} \tag{3.1.12}$$

Since μ_t is defined by (3.0.2), we have that

$$\mathbb{E}[F(\mathbf{w}_{t+1}+\mu_{t+1}\mathbf{z}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t] \leq \mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t]. \tag{3.1.13}$$

Finally, by (3.1.12) and (3.1.13) implies that

$$\mathbb{E}[F(\mathbf{w}_{t+1}+\mu_{t+1}\mathbf{z}_{t+1})|\mathbf{w}_t+\mu_t\mathbf{z}_t] \leq F(\mathbf{w}_t+\mu_t\mathbf{z}_t) - \epsilon_t\Gamma\|\nabla F(\mathbf{w}_t+\mu_t\mathbf{z}_t)\|^2 + \frac{\epsilon_t^2 M}{2}\left(\frac{1}{\delta}+\Gamma\right)^2 S^2.$$

Thus, the proof is complete. □

We are now ready to give convergence results for RES-NAQ. Note that we use the acronym a.s. as mean almost surely.

Theorem 3.1.7. Let $\mathbf{w}^*$ be an optimal solution for problems (3.1.2). Assume that Assumptions 3.1.1 and 3.1.2 hold for $\mathbf{w}_{t+1}$ generated by RES-NAQ. If the stepsize ϵ_t satisfies (3.0.6), then it holds that

$$\liminf_{t\to\infty}\|\mathbf{w}_{t+1}-\mathbf{w}^*\|^2 = 0 \text{ (a.s.)} \tag{3.1.14}$$

Proof. First, we prove that

$$\liminf_{t\to\infty} \|(\mathbf{w}_t + \mu_t \mathbf{z}_t) - \mathbf{w}^*\|^2 = 0 \text{ (a.s.)}$$

Define $\gamma_t := F(\mathbf{w}_t + \mu_t \mathbf{z}_t) + K \sum_{u=t}^{\infty} \epsilon_u^2$ and $\beta_t := \epsilon_t \Gamma \|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2$. Let $\mathcal{F}_t$ be a σ-algebra measuring γ_t, β_t, and $\mathbf{w}_t + \mu_t \mathbf{z}_t$. From Lemma 3.1.6 we know that for any iteration t, it holds that

$$\begin{aligned}
\mathbb{E}[\gamma_{t+1}|\mathcal{F}_t] &= \mathbb{E}[F(\mathbf{w}_{t+1} + \mu_{t+1}\mathbf{z}_{t+1})|\mathcal{F}_t] + K \sum_{u=t+1}^{\infty} \epsilon_u^2 \\
&\leq F(\mathbf{w}_t + \mu_t \mathbf{z}_t) - \epsilon_t \Gamma \|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2 + K \sum_{u=t}^{\infty} \epsilon_u^2 \\
&= \gamma_t - \beta_t.
\end{aligned}$$

Since γ_t and β_t are nonnegative, we obtain that they satisfy the conditions of Theorem 2.2.13. Therefore, we conclude that (I) The sequence γ_t converges almost surely. (II) The sum $\sum_{t=0}^{\infty} \beta_t < \infty$ is almost surely finite. Using the explicit form of β_t, it follows that

$$\sum_{t=0}^{\infty} \epsilon_t \Gamma \|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2 < \infty, \text{ (a.s.)} \tag{3.1.15}$$

Since the sequence of stepsizes is nonsummable for (3.1.15) to be true, we need to have a vanishing subsequence embedded in $\|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2$. This implies that

$$\liminf_{t\to\infty} \|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2 = 0, \text{ (a.s.)} \tag{3.1.16}$$

To transform the gradient bound stated in 3.1.16 into a bound related to the squared distance to optimality $\|\mathbf{w}_t + \mu_t \mathbf{z}_t - \mathbf{w}^*\|^2$, we simply observe that, for the optimal argument $\mathbf{w}^*$, the lower bound in (3.1.5) implies

$$\begin{aligned}
F(\mathbf{w}^*) \geq F(\mathbf{w}_t + \mu_t \mathbf{z}_t) &+ \nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)^\top (\mathbf{w}^* - (\mathbf{w}_t + \mu_t \mathbf{z}_t)) \\
&+ \frac{m}{2} \|\mathbf{w}^* - (\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2.
\end{aligned} \tag{3.1.17}$$

From $\mathbf{w}^*$ is the minimizing argument of $F(\mathbf{w})$, such that $F(\mathbf{w}^*) - F(\mathbf{w}) \leq 0$ for all $\mathbf{w}$. As a result of this fact and the rearrangement of terms, we can simplify (3.1.17) to

$$\frac{m}{2} \|\mathbf{w}^* - (\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2 \leq \nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)^\top ((\mathbf{w}_t + \mu_t \mathbf{z}_t) - \mathbf{w}^*).$$

By the Cauchy-Schwarz inequality, we have

$$\frac{m}{2}\|\mathbf{w}^* - (\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2 \leq \|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|\|(\mathbf{w}_t + \mu_t \mathbf{z}_t) - \mathbf{w}^*\|. \tag{3.1.18}$$

Substitution of this bound in (3.1.18) and simplification of a $\|\mathbf{w}^* - (\mathbf{w}_t + \mu_t \mathbf{z}_t)\|$ factor yields

$$\frac{m}{2}\|(\mathbf{w}_t + \mu_t \mathbf{z}_t) - \mathbf{w}^*\| \leq \|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|. \tag{3.1.19}$$

Taking the limit infimum as t approaches ∞ in (3.1.19) and applying (3.1.16), we have

$$\liminf_{t\to\infty} \|(\mathbf{w}_t + \mu_t \mathbf{z}_t) - \mathbf{w}^*\|^2 = 0 \quad \text{(a.s.)} \tag{3.1.20}$$

Using the definition of $\mathbf{w}_{t+1}$ as $\mathbf{w}_t + \mu_t \mathbf{z}_t - \epsilon_t(\hat{\mathbf{B}}_t^{-1} + \Gamma \mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)$, we obtain

$$\begin{aligned}\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 &= \|\mathbf{w}_t + \mu_t \mathbf{z}_t - \epsilon_t(\hat{\mathbf{B}}_t^{-1} + \Gamma \mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t) - \mathbf{w}^*\|^2 \\ &\leq 2\|\mathbf{w}_t + \mu_t \mathbf{z}_t - \mathbf{w}^*\|^2 + 2\epsilon_t^2\|(\hat{\mathbf{B}}_t^{-1} + \Gamma \mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)\|^2.\end{aligned} \tag{3.1.21}$$

Now, taking expectation conditioned on $\mathbf{w}_t + \mu_t \mathbf{z}_t$ in (3.1.21) we have that

$$\begin{aligned}&\mathbb{E}[\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 | \mathbf{w}_t + \mu_t \mathbf{z}_t] \\ &\quad \leq 2\|\mathbf{w}_t + \mu_t \mathbf{z}_t - \mathbf{w}^*\|^2 + 2\epsilon_t^2\, \mathbb{E}[\|(\hat{\mathbf{B}}_t^{-1} + \Gamma \mathbf{I})\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)\|^2 | \mathbf{w}_t + \mu_t \mathbf{z}_t] \\ &\quad \leq 2\|\mathbf{w}_t + \mu_t \mathbf{z}_t - \mathbf{w}^*\|^2 + 2\epsilon_t^2\|\hat{\mathbf{B}}_t^{-1} + \Gamma \mathbf{I}\|^2\, \mathbb{E}[\|\hat{\mathbf{s}}(\mathbf{w}_t + \mu_t \mathbf{z}_t, \hat{\theta}_t)\|^2 | \mathbf{w}_t + \mu_t \mathbf{z}_t] \\ &\quad \leq 2\|\mathbf{w}_t + \mu_t \mathbf{z}_t - \mathbf{w}^*\|^2 + 2\epsilon_t^2\Big(\frac{1}{\delta} + \Gamma\Big)^2 S^2.\end{aligned} \tag{3.1.22}$$

Taking the limit infimum as t approaches ∞ in (3.1.22) and applying (3.1.20), we have

$$\liminf_{t\to\infty} \mathbb{E}[\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 | \mathbf{w}_t + \mu_t \mathbf{z}_t] \leq 0 \quad \text{(a.s.)} \tag{3.1.23}$$

By applying Theorem 2.2.11 to (3.1.23), we obtain

$$\begin{aligned}&\mathbb{E}[\liminf_{t\to\infty} \|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 | \mathbf{w}_t + \mu_t \mathbf{z}_t] \\ &\qquad \leq \liminf_{t\to\infty} \mathbb{E}[\|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 | \mathbf{w}_t + \mu_t \mathbf{z}_t] \leq 0 \quad \text{(a.s.)}\end{aligned} \tag{3.1.24}$$

By taking expected values on both sides of (3.1.24) and utilizing the fact that the expected value of 0 is 0, along with Theorem 2.2.9(C5), we can conclude that (3.1.24) implies the following:

$$\mathbb{E}\left[\liminf_{t\to\infty} \|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2\right] = \mathbb{E}\left[\mathbb{E}\left[\liminf_{t\to\infty} \|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 \middle| \mathbf{w}_t + \mu_t \mathbf{z}_t\right]\right] \leq \mathbb{E}[0] = 0. \tag{3.1.25}$$

Applying Theorem 2.2.10 to (3.1.25) allows us to conclude that

$$\liminf_{t\to\infty} \|\mathbf{w}_{t+1} - \mathbf{w}^*\|^2 = 0 \quad \text{(a.s.)}$$

Thus, the proof is complete. □

Remark 3.1.8. According to Theorem 3.1.7, the limit inferior of squared distance between iterates $\mathbf{w}_{t+1}$ and the optimal solution $\mathbf{w}^*$ almost surely converges to zero. This indicates that there exists a subsequence of iterates $\mathbf{w}_{t+1}$ converges to the optimal solution $\mathbf{w}^*$.

In the following theorem, we expand upon the findings of convergence presented in Theorem 3.1.7 by introducing a characterization of the expected convergence rate.

Theorem 3.1.9. For iteration t, let step size ϵ_t be given by $\frac{\epsilon_0 T_0}{T_0+t}$ with the positive constants ϵ_0 and T_0 to satisfy the inequality

$$2\epsilon_0 T_0 \Gamma > 1. \tag{3.1.26}$$

Then, under Assumptions 3.1.1 and 3.1.2, the following property hold:

$$\mathbb{E}[F(\mathbf{w}_{t+1})] - F(\mathbf{w}^*) \leq \frac{C_0 + \epsilon_0^2 T_0^2 K}{T_0 + t}, \tag{3.1.27}$$

where

$$C_0 := \max\left\{\frac{\epsilon_0^2 T_0^2 K}{2\epsilon_0 T_0 \Gamma - 1}, T_0(F(\mathbf{w}_0) - F(\mathbf{w}^*))\right\}.$$

Proof. By using the lower bound in (3.1.5), for any $\mathbf{x} \in \mathbb{R}^n$, we have

$$F(\mathbf{x}) \geq F(\mathbf{w}) + \nabla F(\mathbf{w})^\top (\mathbf{x} - \mathbf{w}) + \frac{m}{2}\|\mathbf{x} - \mathbf{w}\|^2. \tag{3.1.28}$$

For fixed $\mathbf{w}$, the right-hand side of (3.1.28) is a quadratic function of $\mathbf{x}$ whose minimum argument we can by setting its gradient to zero. Doing this yields the minimizer $\hat{\mathbf{x}} = \mathbf{w} - \frac{1}{m}\nabla F(\mathbf{w})$ implying that for all $\mathbf{x}$ we must have

$$\begin{aligned} F(\mathbf{x}) &\geq F(\mathbf{w}) + \nabla F(\mathbf{w})^{\top}(\hat{\mathbf{x}} - \mathbf{w}) + \frac{m}{2}\|\hat{\mathbf{x}} - \mathbf{w}\|^2 \\ &\geq F(\mathbf{w}) - \frac{1}{2m}\|\nabla F(\mathbf{w})\|^2. \end{aligned} \tag{3.1.29}$$

The bound in (3.1.29) is true for all $\mathbf{w}$ and $\mathbf{x}$. Setting $\mathbf{x} = \mathbf{w}^*$ and $\mathbf{w} = \mathbf{w}_t + \mu_t \mathbf{z}_t$, we have

$$F(\mathbf{w}^*) \geq F(\mathbf{w}_t + \mu_t \mathbf{z}_t) - \frac{1}{2m}\|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2. \tag{3.1.30}$$

Note by Lemma 3.1.6, we known that

$$\mathbb{E}[F(\mathbf{w}_{t+1} + \mu_t \mathbf{z}_{t+1})|\mathbf{w}_t + \mu_t \mathbf{z}_t] \leq F(\mathbf{w}_t + \mu_t \mathbf{z}_t) - \epsilon_t \Gamma \|\nabla F(\mathbf{w}_t + \mu_t \mathbf{z}_t)\|^2 + K\epsilon_t^2, \tag{3.1.31}$$

where $K := \frac{M}{2}\left(\frac{1}{\delta} + \Gamma\right)^2 S^2$. Substitute (3.1.30) in (3.1.31) and rearrange terms as follows:

$$\begin{aligned} &\mathbb{E}[F(\mathbf{w}_{t+1} + \mu_{t+1}\mathbf{z}_{t+1})|\mathbf{w}_t + \mu_t \mathbf{z}_t] - F(\mathbf{w}^*) \\ &\qquad \leq F(\mathbf{w}_t + \mu_t \mathbf{z}_t) - F(\mathbf{w}^*) - 2\epsilon_t \Gamma m\Big(F(\mathbf{w}_t + \mu_t \mathbf{z}_t) - F(\mathbf{w}^*)\Big) + \epsilon_t^2 K \\ &\qquad = (1 - 2\epsilon_t \Gamma m)\Big(F(\mathbf{w}_t + \mu_t \mathbf{z}_t) - F(\mathbf{w}^*)\Big) + K\epsilon_t^2. \end{aligned} \tag{3.1.32}$$

Then, taking expectation on both sides of (3.1.32) we have that

$$\mathbb{E}[F(\mathbf{w}_{t+1} + \mu_{t+1}\mathbf{z}_{t+1})] - F(w^*) \leq (1 - 2\epsilon_t \Gamma m)(\mathbb{E}[F(\mathbf{w}_t + \mu_t \mathbf{z}_t)] - F(\mathbf{w}^*)) + K\epsilon_t^2. \tag{3.1.33}$$

Further substituting $\epsilon_t = \frac{\epsilon_0 T_0}{T_0 + t}$, we can rewrite (3.1.33) as

$$\begin{aligned} &\mathbb{E}[F(\mathbf{w}_{t+1} + \mu_{t+1}\mathbf{z}_{t+1})] - F(\mathbf{w}^*) \\ &\qquad \leq \left(1 - \frac{2\epsilon_0 T_0 \Gamma m}{T_0 + t}\right)(\mathbb{E}[F(\mathbf{w}_t + \mu_t \mathbf{z}_t)] - F(\mathbf{w}^*)) + \left(\frac{\epsilon_0 T_0}{T_0 + t}\right)^2 K. \end{aligned}$$

By combining condition (3.1.26) with Lemma 3 in [3], we get

$$\mathbb{E}[F(\mathbf{w}_t + \mu_t \mathbf{z}_t)] - F(\mathbf{w}^*) \leq \frac{C_0}{T_0 + t}, \tag{3.1.34}$$

where

$$C_0 := \max\left\{\frac{\epsilon_0^2 T_0^2 K}{2\epsilon_0 T_0 \Gamma - 1}, T_0(F(\mathbf{w}_0) - F(\mathbf{w}^*))\right\}.$$

On the other hand, from (3.1.12) we have

$$\mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t + \mu_t \mathbf{z}_t] - F(\mathbf{w}^*) \leq F(\mathbf{w}_t + \mu_t \mathbf{z}_t) - F(\mathbf{w}^*) + \left(\frac{\epsilon_0 T_0}{T_0 + t}\right)^2 K. \quad (3.1.35)$$

Taking expectation on both side of (3.1.35), we have that

$$\mathbb{E}[\mathbb{E}[F(\mathbf{w}_{t+1})|\mathbf{w}_t + \mu_t \mathbf{z}_t]] - F(\mathbf{w}^*) \leq \mathbb{E}[F(\mathbf{w}_t + \mu_t \mathbf{z}_t)] - F(\mathbf{w}^*) + \left(\frac{\epsilon_0 T_0}{T_0 + t}\right)^2 K. \quad (3.1.36)$$

By applying Theorem 2.2.9(C5) to (3.1.36) and utilizing the fact that $\left(\frac{\epsilon_0 T_0}{T_0+t}\right)^2 K \leq \frac{\epsilon_0^2 T_0^2}{T_0+t} K$, we obtain the following:

$$\mathbb{E}[F(\mathbf{w}_{t+1})] - F(\mathbf{w}^*) \leq \mathbb{E}[F(\mathbf{w}_t + \mu_t \mathbf{z}_t)] - F(\mathbf{w}^*) + \frac{\epsilon_0^2 T_0^2 K}{T_0 + t}. \quad (3.1.37)$$

By applying (3.1.34) to (3.1.37), we obtain

$$\mathbb{E}[F(\mathbf{w}_{t+1})] - F(\mathbf{w}^*) \leq \frac{C_0 + \epsilon_0^2 T_0^2 K}{T_0 + t}.$$

Hence, the proof is complete. □

Remark 3.1.10. Theorem 3.1.9 demonstrates that, with the assumptions mentioned earlier, the resulting expected error in a random iteration t is bounded by $\mathcal{O}(\frac{1}{t})$. In stochastic optimization methods, including classical SGD, an expected convergence rate of order $\mathcal{O}(\frac{1}{t})$ is standard. Although the convergence rate of our method remains unchanged, the convergence time improves substantially, as shown in the next section.

3.2 Numerical experiments

In this section, we evaluate the performance of the proposed method through a support vector machine problem. The experiments have been performed on Python3 on a macOS with an Intel i5 Processor 2.3 GHz with a memory of 8

GB 2133 MHz LPDDR3. In the following paragraphs, the support vector machine problem is briefly discussed.

In the supervised learning framework, we denote the training set by $T = \{\theta_i\}_{i=1}^N = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^n$ represents a sample and $y_i \in \{-1, 1\}$ its corresponding label. Specifically, we aim to solve the following SVM problem:

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^n} F(\mathbf{w}) &:= \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{w}, \theta_i) \\ &:= \frac{C}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N} l((\mathbf{x}_i, y_i); \mathbf{w}), \end{aligned} \tag{3.2.1}$$

where $f(\mathbf{w}, \theta_i) = \frac{C}{2}\|\mathbf{w}\|^2 + \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i))^2$ and $C > 0$ is a hyperparameter. To evaluate the effectiveness of our proposed method, we apply it to both synthetic and real-world datasets. The following Python code implements the algorithm for this experiment:

```
import numpy as np
import time
from numpy.linalg import norm
class ResNaq():
    def __init__(self, C = 1, delta = 0.0001, gamma = 0.0001,
                 epsilon_0 = 0.0001, tau_0 = 0.0001,
                 mu = 0.99, max_epochs = 1, n_batches = 5,
                 time = 100):
        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)
```

```
24          self.n_batches=n_batches
25          assert time > 0
26          self.time = time
27
28      '''gradient of squareHingeloss'''
29      def gradient(self, w, x, t):
30          u = np.maximum(0, 1-((t[:, np.newaxis]*x).dot(w)))
31          p = -2/len(t)*u.dot(t[:,np.newaxis]*x)
32          grad = self.C * w + p
33          return u,p,grad
34
35      ''' cost function (squareHingeloss SVM) '''
36      def cost_function(self,w,x,t):
37          loss = 1/len(t) *
38                  (np.sum((np.maximum(0, 1-((t[:,
39                  np.newaxis]*x).dot(w)))**2)))
40          cost = self.C/2 * np.linalg.norm(w)**2  + loss
41          return loss, cost
42
43      ''' approximate ResNaq_BFGS matrix '''
44      def Update_RES_BFGS(self,B, dw, dg, I):
45          dg_t =  dg[:, np.newaxis]
46          Bdw = np.dot(B, dw)
47          Bdw = np.dot(B, dw)
48          dw_t_B = np.dot(dw, B)
49          dwBdw = np.dot(np.dot(dw, B), dw)
50          p = dg_t*dg
51          u = Bdw[:, np.newaxis] * dw_t_B
52          B_new = B + p / np.dot(dg, dw) - u /
53                  dwBdw + self.delta * I
54          return p, u, B_new
55
56      def fit(self, x, t):
57          #initail variables k, stop, w_0
58          k = 0
59          w = np.ones(len(x[0]))
60          z = np.zeros(len(x[0]))
61
62          #set up matrices
63          obj_resnaq = []
```

```
64        objnat_resnaq = []
65        iter_resnaq = []
66        mu_ = []
67        time_resnaq = np.empty(self.time)
68        B = np.identity(len(x[0]))  #inverse hessian
69        I = np.identity(len(x[0]))  #identity
70
71
72        for epoch in range(self.max_epochs):
73            idx = np.random.permutation(len(t))
74            print("Epoch: %d" %(epoch+1), idx)
75            for i in range(len(t)):
76
77                r = idx[i*self.n_batches:(i+1)*self.n_batches]
78                if r.size==0: break
79
80                k = k + 1
81                iter_resnaq.append(k)
82
83                # stamp time
84                start_time = time.perf_counter()
85
86                # compute cost function at w
87                _, cost = self.cost_function(w,x,t)
88                obj_resnaq.append(cost)
89
90                # compute cost function at w+(mu*z)
91                _, cost_naq = self.cost_function(w+
92                              (self.mu*z),x,t)
93                objnat_resnaq.append(cost_naq)
94
95
96                print("----Iteration: %d" %(i+1), r)
97
98
99                if cost < cost_naq:
100                    mu_t = 0
101                    print('----mu:', mu_t)
102                    mu_.append(mu_t)
103
```

```
104                 # compute nesterov'gradient
105                 _, _, g_naq = self.gradient(w+
106                                 (mu_t*z),x[r,:],t[r])
107
108                 #if np.linalg.norm(g_naq)==0: break
109
110                 # update hessian matrix
111                 H = np.linalg.inv(B)
112                 H_RES = H + self.gamma*I
113
114                 # step size
115                 eta =((self.epsilon_0)+
116                     (self.tau_0))/((self.tau_0)+k)
117
118
119                 # compute nesterov'direction
120                 z_new = mu_t*z - eta * np.matmul(H_RES,g_naq)
121
122                 # update weight
123                 w_new = w + z_new
124
125                 # compute new gradient
126                 _, _, g_new = self.gradient(w_new,x[r,:],t[r])
127
128                 # get difference of values
129                 dw = w_new - (w + mu_t * z)
130                 # get difference of gradients
131                 dg = g_new - g_naq - (self.delta * dw)
132
133
134                 _, _, B_new = self.Update_RES_BFGS
135                                 (B, dw, dg, I)
136
137                 # update step
138                 B = B_new
139                 w = w_new
140                 z = z_new
141
142
143             else:
```

```
144                 mu_t = self.mu
145                 print('----mu:', mu_t)
146                 mu_.append(mu_t)
147
148                 # compute nesterov'gradient
149                 _, _, g_naq = self.gradient(w+
150                             (mu_t*z),x[r,:],t[r])
151
152                 #if np.linalg.norm(g_naq)==0: break
153
154                 # update hessian matrix
155                 H = np.linalg.inv(B)
156                 H_RES = H + self.gamma*I
157
158                 # step size
159                 eta =((self.epsilon_0)*
160                       (self.tau_0))/((self.tau_0)+k)
161
162
163                 # compute nesterov'direction
164                 z_new = mu_t*z - eta * np.matmul(H_RES,g_naq)
165
166                 # update weight
167                 w_new = w + z_new
168
169                 # compute new gradient
170                 _, _, g_new = self.gradient(w_new,x[r,:],t[r])
171
172                 # get difference of values
173                 dw = w_new - (w + mu_t * z)
174                 # get difference of gradients
175                 dg = g_new - g_naq - (self.delta * dw)
176
177                 _, _, B_new = self.Update_RES_BFGS
178                             (B, dw, dg, I)
179
180                 # update step
181                 B = B_new
182                 w = w_new
183                 z = z_new
```

```
                # cpu time used
                stop_time = time.perf_counter()
                time_resnaq[i] = stop_time - start_time


        self.final_iter = k
        self._coef = w
        self.obj_resnaq = obj_resnaq
        self.iter_resnaq = iter_resnaq
        self.objnat_resnaq = objnat_resnaq
        self.mu_ = mu_
        self.time_resnaq = time_resnaq
        return self

    def predict(self, x):
        p = np.sign(np.matmul(x,self._coef))
        p[p==0] = 1
        return p.astype(int)
```

3.2.1 Performance Comparisons with synthetic dataset

The numerical experiments in this part are performed on a synthetic binary classification dataset. We generate 1000 synthetic data points with a dimension of 40. Half of the data points have label 1, and the other half have label -1. The components of samples labelled 1 and -1 are uniformly at random from the intervals $[-0.2, 0.8]$ and $[-0.8, 0.2]$, respectively. This script how the training data were generated in Python:

```
import numpy as np
import pandas as pd

# Generate random data for two classes
data_class1 = np.random.uniform(low=-0.8, high=0.2, size=(500, 40))
data_class2 = np.random.uniform(low=-0.2, high=0.8, size=(500, 40))

# Create DataFrames for each class
```

```
df_class1 = pd.DataFrame(data_class1)
df_class2 = pd.DataFrame(data_class2)

# Add a 'class' column to represent the class label
df_class1['class'] = 0
df_class2['class'] = 1

# Concatenate the two DataFrames
data = pd.concat([df_class1, df_class2], ignore_index=True)

print(data)
```

Therefore, to evaluate the proposed method's effectiveness, we compared it with the following related methods.

1. The stochastic gradient descent method [7]. We refer to this method as SGD.
2. The Nesterov's accelerated gradient method [6]. We refer to this method as NAG.
3. The regularized stochastic BFGS method [3]. We refer to this method as RES.
4. The stochastic quasi-Newton with Nesterov's accelerated gradient method [5], with the model-trust region parameter λ, is set to 1. We refer to this method as oNAQ.

For this subsection, parameters were determined as follows. For all methods, C in (3.2.1) was set to 10^{-3}. It follows that the Hessian eigenvalues of the objective function in (3.2.1) are at least equal to 10^{-3}. Therefore, by Assumption 3.1.3, we set the regularization parameter $\delta = 10^{-4}$. Further, $\Gamma = 10^{-4}$ is the minimum progress parameter in (2.3.16) and set $L = 5$ as the sample size for the computation of stochastic gradients. Stepsizes are of form $\epsilon_t = \epsilon_0 T_0/(T_0 + t)$, with SGD and NAG using $\epsilon_0 = 4 \times 10^{-1}$ and $T_0 = 10^6$ and the other methods using $\epsilon_0 = 10^{-3}$ and $T_0 = 10^8$. The experiment results are shown in Figure 18.

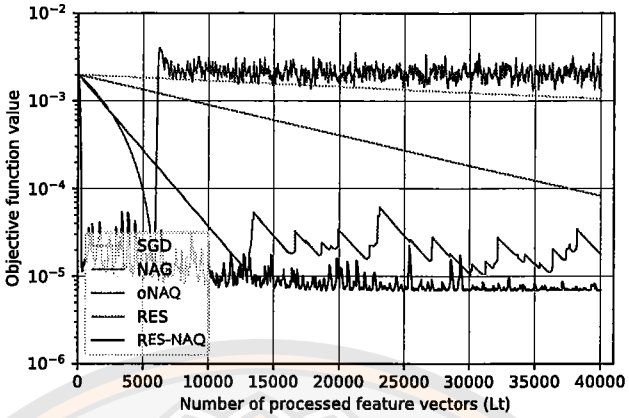


Figure 18 Comparison of SGD, NAG, oNAQ, RES, and RES-NAQ. Parameter values: NAG: $\mu = 0.8$; oNAQ: $\mu = 0.8$; RES-NAQ: $\mu = 0.99$.

The objective function values of (3.2.1) with respect to the number of feature vectors processed, which is determined by the product of the sample size used to compute stochastic gradients (L) and the iteration index (t), are used to illustrate the performances of each algorithm. As shown in Figure 18, we can observe that second-order algorithms (i.e., RES, oNAQ, RES-NAQ) outperform first-order methods (i.e., SGD, NAG). However, the momentum coefficient in oNAQ is fixed in every iteration, which leads to an overshoot after processing 6×10^3 feature vectors. Furthermore, it can be seen that RES oscillates with a higher frequency than RES-NAQ after processing 1.4×10^{-4} feature vectors. Accordingly, our method achieved the best results.

Moreover, to obtain the objective function value $F(\mathbf{w}_t) = 10^{-4}$, we also report the number of feature vectors processed and the training time of all these algorithms. Table 1 shows the experiment's results. As shown in Table 1, compared to other methods, the proposed method uses less processed feature vectors and less time to achieve the objective function value $F(\mathbf{w}_t) = 10^{-4}$. As a result, the proposed method outperforms in terms of training time.

Table 1 Number of processed feature vectors and training time (in seconds) utilized to achieve the objective value $F(w_t) = 10^{-4}$.

Algorithms	Number of processed feature vectors	Training time
SGD	$> 4 \times 10^4$	> 22.9009
NAG	3.75×10^4	10.0770
RES	5.78×10^3	0.6196
oNAQ	*	*
RES-NAQ	2.20×10^2	0.1296

Furthermore, we contrast the proposed method with those that contain structures that avoid the near-singularity problem of approximation Hessian matrix $\mathbf{B}$. Note that we utilize the same parameters in this experiment as in Figure 18. The methods that we compared are as follows.

1. The stochastic quasi-Newton method [2] dealt with near-singularity by modifying the BFGS update to estimate the inverse of $\mathbf{B} + \lambda\mathbf{I}$, where λ is a model trust-region parameter. We set $\lambda = 0.01$ and refer to this method as oBFGS.

2. The trust-region BFGS method [1], in contrast to oBFGS, is typically implemented with a trust-region; it is introduced in the following. At the t-th iteration, the trust-region BFGS method computes a new iterate by the formula

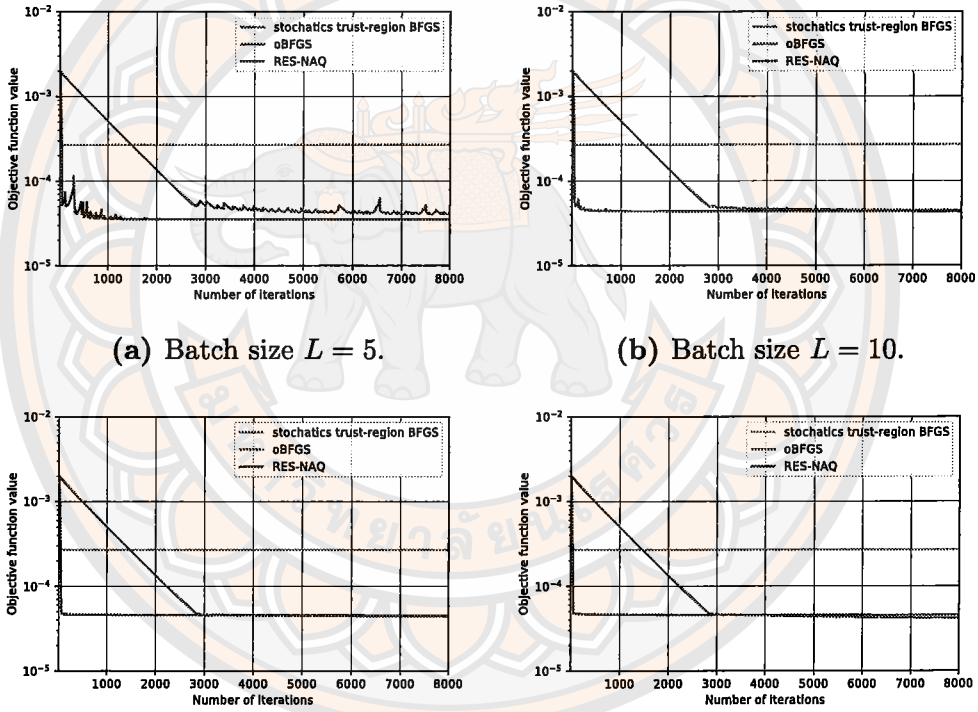
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{p}_t, \tag{3.2.2}$$

where $\mathbf{p}_t$ is the minimizer of the following subproblem

$$\begin{aligned} \min_{\mathbf{p}} \quad & m_t(\mathbf{p}) = F(\mathbf{w}_t) + \nabla F(\mathbf{w}_t)^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \mathbf{B}_t \mathbf{p}, \\ \text{s.t.} \quad & \|\mathbf{p}\| \leq \Delta_t, \end{aligned} \tag{3.2.3}$$

Δ_t is the trust-region and $\mathbf{B}_t$ is an approximate Hessian matrix updated by (2.3.13). In our comparison, the main goal of this research is the stochastic regime. We modify the trust-region BFGS approach for the stochastic regime as a result. Instead of using the exact gradient, it only employs the stochastic gradient. This method is called as stochastic trust-region BFGS.

In Figure 19, we demonstrate the behaviour of achieving the minimal value of the objective function of each method in a predetermined number of iterations via various batch sizes. As seen in Figure 19, we can see that the proposed method performs smoother and faster in convergence than other related methods. Thus, RES-NAQ performs better than other related methods. Furthermore, the proposed method performs well for various batch sizes, indicating that momentum coefficient and regularization are helpful in reducing estimation variance. Hence, RES-NAQ is also more robust to the variations of batch sizes.



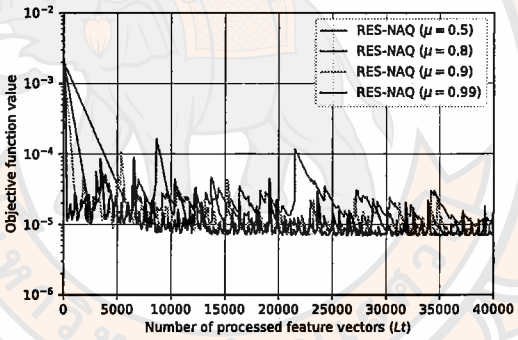
(a) Batch size $L = 5$. **(b)** Batch size $L = 10$.

(c) Batch size $L = 32$. **(d)** Batch size $L = 64$.

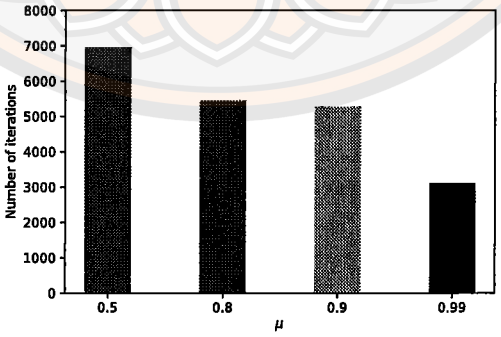
Figure 19 Comparison of stochastic trust-region BFGS, stochastic BFGS, and RES-NAQ. The proposed method and oBFGS used stepsize $\epsilon_t = \epsilon_0 T_0/(T_0 + t)$:$\epsilon_0 = 0.5/7, T_0 = 2 \times 10^4$ (oBFGS); $\epsilon_0 = 10^{-3}, T_0 = 10^8$ (RES-NAQ); C, δ, Γ as in Figure 18.

On the other hand, the momentum coefficient μ_t, as defined in (3.0.2), plays a crucial role in introducing the proposed method. It is noteworthy that, according to its definition, μ_t's value alternates between μ and 0 at each iteration. Thus, we

aim to investigate the performance of our proposed method for different values of μ. In Figure 20, the efficacy of the proposed method across various μ settings is showcased, completed by a bar chart that tallies the number of iterations, where μ_t equals μ. It is observed that as the value of μ increases, the objective function declines sharply at begin. Simultaneously, there is no overshoot in the long term due to the definition that if the descent property is not satisfied, the value of μ_t will be reset to zero. Conversely, setting μ to a lower value allows our method to leverage the momentum term, resulting in a smoother performance. This situation illustrates the method's performance trade-off between acceleration and oscillation. However, based on our adjustable momentum coefficient, μ_t plays a pivotal role here. It allows us to fine-tune the process, accelerating performance without causing instability or overshoot.



(a) Performance of RES-NAQ.



(b) The number of iterations that μ_t is equal to μ.

Figure 20 Performance of RES-NAQ with different values of μ.

3.2.2 Performance Comparisons with real-world binary datasets

In this subsection, we utilize various real-world binary datasets from the UCI machine repository [34] to evaluate the effectiveness of our method in comparison to other relevant approaches such as SGD, NAG, RES, and oNAQ. The employed datasets are detailed in Table 2.

Table 2 Description of datasets.

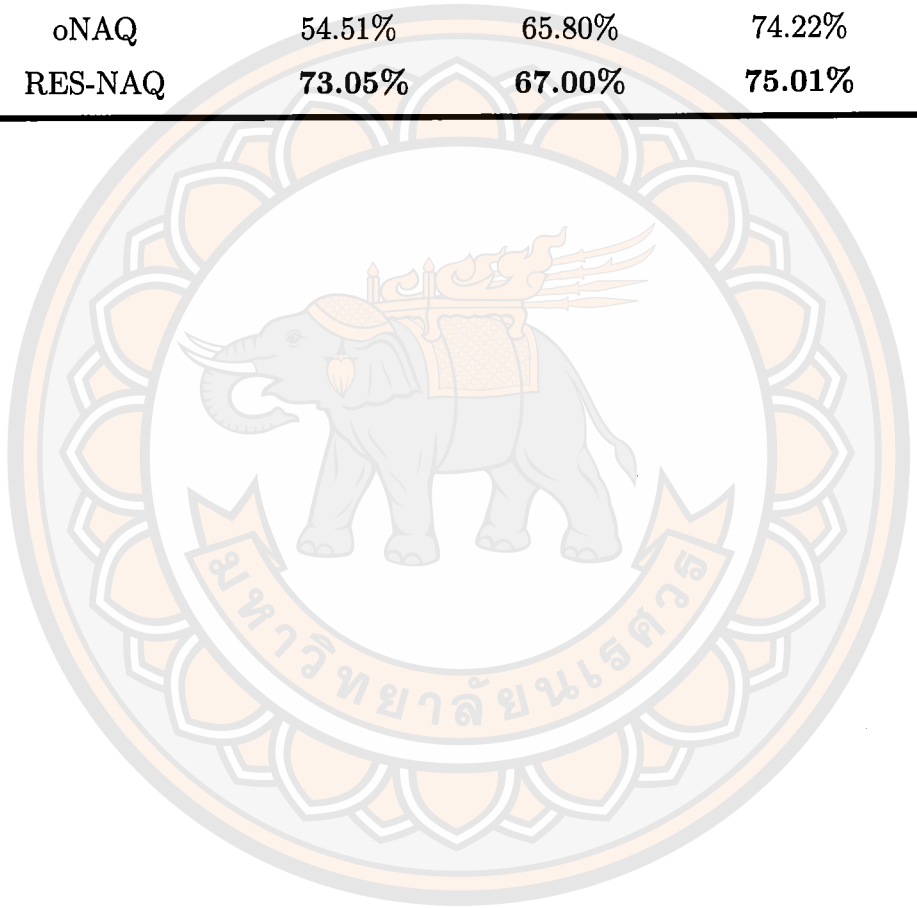
Datasets	No. of samples	Dimension
Spect Heart	266	44
Bupa	345	6
Diabates	768	8
Breast	116	44

Before training, we standardize the dataset and employ the grid search technique [37] to optimize the step-size parameters, regularized parameters, and momentum coefficient. We choose values for the step-size parameter ϵ_0 from the set $\{10^i \mid i = -1, -2, -3\}$ and for T_0 from the set $\{10^i \mid i = 7, 8, 9, 10, 11, 12, 13\}$. The regularized parameters δ and Γ are varied over the range $\{10^i \mid i = -3, -4, -5\}$. The momentum coefficient is selected from the set $\{0.8, 0.9, 0.99\}$. The classification results are obtained through a 5-fold cross-validation strategy.

The comparative results are displayed in Table 3. It is seen that the proposed method consistently demonstrates outstanding accuracy across the majority of datasets.

Table 3 Comparative results.

Algorithms	Data sets			
	Spect	**Bupa**	**Diabetes**	**Breast**
SGD	56.76%	63.77%	74.48%	61.99%
NAG	71.91%	64.64%	74.75%	67.90%
RES	67.82%	64.43%	73.71%	64.67%
oNAQ	54.51%	65.80%	74.22%	70.51%
RES-NAQ	**73.05%**	**67.00%**	**75.01%**	**70.54%**



CHAPTER IV

SUPPORT VECTOR MACHINE

In this chapter, we present a novel version of the Support Vector Machine (SVM) model, showcasing enhanced performance. The chapter is divided into two parts: the first introduces a pioneering loss function tailored for the SVM model, while the subsequent part explores the evolution and enhancement of the SVM framework, specifically focusing on large-scale data classification.

4.1 Smooth support vector machine with generalized pinball loss for Pattern Classification

Within this section, we transition from our prior exploration that involved second-order methods for solving the SVM model. We now focus on creating a smooth approximate loss function derived from the generalized pinball loss function. Initially, we introduce this novel approximation in the first part of this section. Subsequently, we aim to validate its effectiveness as an estimator for the generalized pinball loss through theorems and analysis.

As discussed in Section 2.4, the generalized pinball loss function is non-differentiable. To address this limitation, we employ the Moreau-Yosida regularization concept to approximate the generalized pinball loss function. The Moreau-Yosida regularization $Q(u;\mu)$ of the function $L(x)$ is defined as the infimum over all y of the expression $L(y) + \frac{1}{2\mu}(y-u)^2$, i.e.,

$$Q(u;\mu) = \inf_{y} \left\{ L(y) + \frac{1}{2\mu}(y-u)^2 \right\}. \tag{4.1.1}$$

This method is a valuable tool for smoothing non-differentiable functions, making them more agreeable to analysis and optimization techniques that require differentiability.

Example 4.1.1. Let $L(x) = |x|$. The Moreau-Yosida regularization of $L(x)$ is

$$Q(u;\mu) = \inf_y \left\{ |y| + \frac{1}{2\mu}(y-u)^2 \right\} = \begin{cases} u - \mu, & \text{if } u \geq \mu, \\ \frac{1}{2\mu}u^2, & \text{if } -\mu \leq u < \mu, \\ -u - \mu, & \text{if } u < -\mu. \end{cases} \tag{4.1.2}$$

Then, by applying the Moreau-Yosida Regularization to the generalized pinball loss function, we obtain a smooth approximation given by:

$$Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = \begin{cases} \tau_1(u - \frac{\epsilon_1}{\tau_1}) - \frac{\tau_1^2}{2}\mu, & \frac{\epsilon_1}{\tau_1} + \tau_1\mu < u, \\ \frac{1}{2\mu}(u - \frac{\epsilon_1}{\tau_1})^2, & \frac{\epsilon_1}{\tau_1} < u \leq \frac{\epsilon_1}{\tau_1} + \tau_1\mu, \\ 0, & \frac{-\epsilon_2}{\tau_2} \leq u \leq \frac{\epsilon_1}{\tau_1}, \\ \frac{1}{2\mu}(u + \frac{\epsilon_2}{\tau_2})^2, & \frac{-\epsilon_2}{\tau_2} - \tau_2\mu \leq u < \frac{-\epsilon_2}{\tau_2}, \\ -\tau_2(u + \frac{\epsilon_2}{\tau_2}) - \frac{\tau_2^2}{2}\mu, & u < \frac{-\epsilon_2}{\tau_2} - \tau_2\mu. \end{cases} \tag{4.1.3}$$

Here, $u \in \mathbb{R}$. The visualization of $Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot;\mu)$ is shown in Figure 21, illustrating its smoothness as a mapping for $\mathcal{L}^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot)$. The subsequent lemma ensures that our proposed mapping serves as a valid estimator for the generalized pinball loss, verifying the convergence of the sequence $Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu)$ to $L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u)$ as $\mu \to 0^+$.

Lemma 4.1.2. For any $\tau_1, \tau_2 > 0, \epsilon_1, \epsilon_2 \geq 0$ and $u \in \mathbb{R}$. Let $L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot)$ and $Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot;\mu)$ defined as in (2.4.12) and (4.1.3) respectively. Then, the following statements hold.

1. $0 \leq L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) \leq \frac{\tau_0^2}{2}\mu$, where $\tau_0 = \max\{\tau_1, \tau_2\}$,
2. $\lim_{\mu\to 0^+} Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u)$.

Proof.

1. Let $\tau_1, \tau_2, \epsilon_1, \epsilon_2 \geq 0$ and $u \in \mathbb{R}$ and fixed $\mu \in \mathbb{R}^+$, by definition of $L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot)$ and $Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot,\mu)$, we divided the proof into 5 cases.

Case 1 : If $\frac{\epsilon_1}{\tau_1} + \tau_1\mu < u$, we have

$$L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = \frac{\tau_1^2}{2}\mu \leq \frac{\tau_0^2}{2}\mu.$$

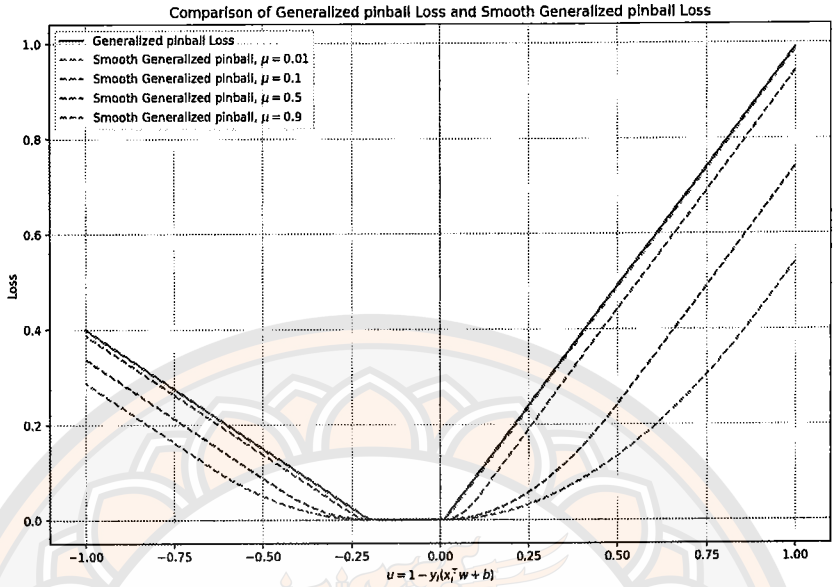


Figure 21 Graphs of $\mathcal{L}^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot)$ and $Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot;\mu)$ with different μ.

Case 2 : If $\frac{\epsilon_1}{\tau_1} < u \leq \frac{\epsilon_1}{\tau_1} + \tau_1\mu$, consider

$$L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = \tau_1(u - \frac{\epsilon_1}{\tau_1}) - \frac{1}{2\mu}(u - \frac{\epsilon_1}{\tau_1})^2 = I(u).$$

Therefore, $I'(u) = \tau_1 - \frac{1}{\mu}(u - \frac{\epsilon_1}{\tau_1}) \geq 0$ over $\left(\frac{\epsilon_1}{\tau_1}, \frac{\epsilon_1}{\tau_1} + \tau_1\mu\right]$. This implies that I is increasing over $\left(\frac{\epsilon_1}{\tau_1}, \frac{\epsilon_1}{\tau_1} + \tau_1\mu\right]$, i.e.,

$$L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) \leq \frac{\tau_1^2}{2}\mu \leq \frac{\tau_0^2}{2}\mu.$$

Case 3 : If $\frac{-\epsilon_2}{\tau_2} \leq u \leq \frac{\epsilon_1}{\tau_1}$, we have

$$L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = 0.$$

Case 4 : If $\frac{-\epsilon_2}{\tau_2} - \tau_2\mu \leq u < \frac{-\epsilon_2}{\tau_2}$, consider

$$L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = -\tau_2(u + \frac{\epsilon_2}{\tau_2}) - \frac{1}{2\mu}(u + \frac{\epsilon_2}{\tau_2})^2 = J(u).$$

Therefore, $J'(u) = -\tau_2 - \frac{1}{\mu}(u + \frac{\epsilon_2}{\tau_2}) \leq 0$ over $\left[-\frac{\epsilon_2}{\tau_2} - \tau_2\mu, -\frac{\epsilon_2}{\tau_2}\right)$. This implies that J is decreasing over $\left[-\frac{\epsilon_2}{\tau_2} - \tau_2\mu, -\frac{\epsilon_2}{\tau_2}\right)$, i.e.,

$$L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) \leq \frac{\tau_2^2}{2}\mu \leq \frac{\tau_0^2}{2}\mu.$$

Case 5 : If $u < -\frac{\epsilon_2}{\tau_2} - \tau_2\mu$, we have

$$L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = \frac{\tau_2^2}{2}\mu \leq \frac{\tau_0^2}{2}\mu.$$

2. For fixed $u \in \mathbb{R}$. Since $0 \leq L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) \leq \frac{\tau_0^2}{2}\mu$ and $\lim_{\mu\to 0^+} \frac{\tau_0^2}{2}\mu = 0$, thus, by the Squeeze theorem, we have

$$\lim_{\mu\to 0^+} Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u;\mu) = L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u).$$

□

Again, let us consider a training dataset $X = \{(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^n, y_i = \pm 1, i = 1, 2, 3, \ldots, m\}$. Then, we use smoothing approximation $Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot,\mu)$ instead of generalized pinball loss function $L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot)$, i.e., we will solve problem

$$\min_{\boldsymbol{\omega}} \varphi_\mu(\boldsymbol{\omega}) := \frac{1}{2}\|\boldsymbol{\omega}\|^2 + \frac{c}{m}\sum_{i=1}^{m} Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(1 - y_i\boldsymbol{\omega}^\top \mathbf{x}_i, \mu), \tag{4.1.4}$$

where $\boldsymbol{\omega} = [\mathbf{w}^\top, b]^\top \in \mathbb{R}^{n+1}$ and $\mathbf{x}_i = [\mathbf{x}_i, 1]$. To work with the smooth approximation, in the following we will show that the family of the optimization problem (4.1.4), for each problem, has a unique solution and the sequence of solution for a family of (4.1.4) converge to the solution of the exact problem:

$$\min_{\boldsymbol{\omega}} \quad \psi_0(\boldsymbol{\omega}) := \frac{1}{2}\|\boldsymbol{\omega}\|^2 + \frac{c}{m}\sum_{i=1}^{m} L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(1 - y_i\boldsymbol{\omega}^\top \mathbf{x}_i), \tag{4.1.5}$$

as $\mu \to 0^+$.

Theorem 4.1.3. For any $\tau_1, \tau_2 > 0, \epsilon_1, \epsilon_2 \geq 0$. Let $\varphi_\mu(\boldsymbol{\omega})$ and $\psi_0(\boldsymbol{\omega})$ be defined as in (4.1.4) and (4.1.5) respectively, and let $\boldsymbol{\omega}^*$ be an optimal solution for problem (4.1.5). Then the following hold:

1. There exists unique solution $\boldsymbol{\omega}^*_\mu$ to $\min_{\boldsymbol{\omega}} \varphi_\mu(\boldsymbol{\omega})$,
2. $\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|_2^2 \leq c\left(\frac{\tau_0^2}{2}\mu\right)$,
3. $\boldsymbol{\omega}^*_\mu \to \boldsymbol{\omega}^*$ as $\mu \to 0^+$.

Proof.

1. For fixed $\mu > 0$, we have $Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot,\mu)$ is convex. Therefore, by the convexity we have $\frac{c}{m}\sum_{i=1}^{m} Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(\cdot,\mu)$ is also convex. Since $\frac{1}{2}\|\boldsymbol{\omega}\|^2$ is strongly convex with the parameter is 1, we obtain that $\varphi_\mu(\boldsymbol{\omega})$ is strongly convex. Consider $L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u) - Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u,\cdot) \geq 0$, then we have the level sets

$$L_\nu(\varphi_\mu(\boldsymbol{\omega})) := \left\{\boldsymbol{\omega} \in \mathbb{R}^{n+1} : \varphi_\mu(\boldsymbol{\omega}) \leq \nu\right\},$$

satisfy

$$L_\nu(\varphi_\mu(\boldsymbol{\omega})) \subseteq \left\{\boldsymbol{\omega} \in \mathbb{R}^{n+1} : \boldsymbol{\omega}^\top\boldsymbol{\omega} \leq 2\nu\right\},$$

for any $\nu \geq 0$. Thus, we obtain that $L_\nu(\varphi_\mu(\boldsymbol{\omega}))$ is a compact subset in $\mathbb{R}^{n+1}$. This implies that solution $\boldsymbol{\omega}^*_\mu$ of (4.1.4) exists, and by the strong convexity, its solution is unique.

2. Support that $\boldsymbol{\omega}^*_\mu$ and $\boldsymbol{\omega}^*$ are the optimal solution of (4.1.4) and (4.1.5), respectively. By the strong convexity, we have the following

$$\begin{aligned}
\psi_0(\boldsymbol{\omega}^*_\mu) - \psi_0(\boldsymbol{\omega}^*) &\geq \nabla\psi_0(\boldsymbol{\omega}^*)(\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*) + \frac{1}{2}\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|^2, \\
\varphi_\mu(\boldsymbol{\omega}^*) - \varphi_\mu(\boldsymbol{\omega}^*_\mu) &\geq \nabla\varphi_\mu(\boldsymbol{\omega}^*_\mu)(\boldsymbol{\omega}^* - \boldsymbol{\omega}^*_\mu) + \frac{1}{2}\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|^2.
\end{aligned}$$

From the first-order necessary optimality condition, we have

$$\begin{aligned}
\psi_0(\boldsymbol{\omega}^*_\mu) - \psi_0(\boldsymbol{\omega}^*) &\geq \frac{1}{2}\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|^2, \\
\varphi_\mu(\boldsymbol{\omega}^*) - \varphi_\mu(\boldsymbol{\omega}^*_\mu) &\geq \frac{1}{2}\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|^2.
\end{aligned}$$

Since

$$\psi_0(\boldsymbol{\omega}) - \varphi_\mu(\boldsymbol{\omega}) = \frac{c}{m}\sum_{i=1}^{m} L^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u_i) - \frac{c}{m}\sum_{i=1}^{m} Q^{\epsilon_1,\epsilon_2}_{\tau_1,\tau_2}(u_i,\mu) \geq 0,$$

where $u_i = 1 - y_i\boldsymbol{\omega}^\top\mathbf{x}_i$. Therefore,

$$\begin{aligned}
\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|^2 &= \frac{1}{2}\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|^2 + \frac{1}{2}\|\boldsymbol{\omega}^*_\mu - \boldsymbol{\omega}^*\|^2 \\
&\leq \psi_0(\boldsymbol{\omega}^*_\mu) - \psi_0(\boldsymbol{\omega}^*) + \varphi_\mu(\boldsymbol{\omega}^*) - \varphi_\mu(\boldsymbol{\omega}^*_\mu) \\
&= \psi_0(\boldsymbol{\omega}^*_\mu) - \varphi_\mu(\boldsymbol{\omega}^*_\mu) - [\psi_0(\boldsymbol{\omega}^*) - \varphi_\mu(\boldsymbol{\omega}^*)] \\
&\leq \psi_0(\boldsymbol{\omega}^*_\mu) - \varphi_\mu(\boldsymbol{\omega}^*_\mu)
\end{aligned} \tag{4.1.6}$$

$$
\begin{aligned}
&= \frac{c}{m}\sum_{i=1}^{m} L_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(u_i^*) - \frac{c}{m}\sum_{i=1}^{m} Q_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(u_i^*,\mu) \\
&\leq c\left(\frac{\tau_0^2}{2}\mu\right).
\end{aligned}
$$

3. Since $0 \leq \|\boldsymbol{\omega}_\mu^* - \boldsymbol{\omega}^*\|^2 \leq c\left(\frac{\tau_0^2}{2}\mu\right)$, by using squeez theorem, we have

$$\boldsymbol{\omega}_\mu^* \to \boldsymbol{\omega}^* \quad \text{as} \quad \mu \to 0^+.$$

□

4.1.1 Quasi-Newton smooth generalized pinball support vector machine (QN-SSVM)

This section describes the quasi-Newton-Armijo method we used to solve our problem, denoted as QN-SSVM. Again, we consider the strongly convex differentiable problems (4.1.4) :

$$\min_{\boldsymbol{\omega}} \quad \varphi_\mu(\boldsymbol{\omega}) := \frac{1}{2}\|\boldsymbol{\omega}\|^2 + \frac{c}{m}\sum_{i=1}^{m} Q_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(1 - y_i\boldsymbol{\omega}^\top \mathbf{x}_i, \mu).$$

In the t-iteration, we compute the gradient of $\varphi_\mu(\boldsymbol{\omega}_t)$ at $\boldsymbol{\omega}_t$, that is

$$\nabla\varphi_\mu(\boldsymbol{\omega}_t) = \boldsymbol{\omega}_t - \frac{c}{m}\sum_{i=1}^{m} y_i\mathbf{x}_i\hat{Q}(u_i^t), \tag{4.1.7}$$

where

$$\hat{Q}(u_i^t) = \begin{cases} \tau_1, & \frac{\epsilon_1}{\tau_1} + \tau_1\mu < u_i^t, \\ \frac{1}{\mu}(u_i^t - \frac{\epsilon_1}{\tau_1}), & \frac{\epsilon_1}{\tau_1} < u_i^t \leq \frac{\epsilon_1}{\tau_1} + \tau_1\mu, \\ 0, & \frac{-\epsilon_2}{\tau_2} \leq u_i^t \leq \frac{\epsilon_1}{\tau_1}, \\ \frac{1}{\mu}(u_i^t + \frac{\epsilon_2}{\tau_2}), & \frac{-\epsilon_2}{\tau_2} - \tau_2\mu \leq u_i^t < \frac{-\epsilon_2}{\tau_2}, \\ -\tau_2, & u_i^t < \frac{-\epsilon_2}{\tau_2} - \tau_2\mu \end{cases}$$

and $u_i^t = 1 - y_i\boldsymbol{\omega}_t{}^\top\mathbf{x}_i$. Therefore, the quasi-Newton-Armijo method can be described as follows:

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \epsilon_t(\mathbf{B}_t^{-1})\nabla\varphi_\mu(\boldsymbol{\omega}_t), \tag{4.1.8}$$

where $\epsilon_t > 0$ is satisfying the Armijo condition: $c_1 \in (0,1)$ such that

$$\varphi_\mu(\boldsymbol{\omega}_t) - \varphi_\mu(\boldsymbol{\omega}_t - \epsilon_t \mathbf{B}_t^{-1}\nabla\varphi_\mu(\boldsymbol{\omega}_t)) \geq -c_1\epsilon_t \nabla\varphi_\mu(\boldsymbol{\omega}_t)^\top \left(\mathbf{B}_t^{-1}\nabla\varphi_\mu(\boldsymbol{\omega}_t)\right),$$

and $\mathbf{B}_t$ is an approximation to the hessian matrix $\nabla^2\varphi_\mu(\boldsymbol{\omega}_t)$. Then, for the next iteration, the hessian approximation $\mathbf{B}_k$ is defined as the matrix that satisfies the secant condition as follows:

$$\mathbf{B}_{t+1} = \mathbf{B}_t - \frac{\mathbf{B}_t\mathbf{v}_t\mathbf{v}_t^\top\mathbf{B}_t}{\mathbf{v}_t^\top\mathbf{B}_t\mathbf{v}_t} + \frac{\mathbf{r}_t\mathbf{r}_t^\top}{\mathbf{r}_t^\top\mathbf{v}_t}, \tag{4.1.9}$$

where

$$\mathbf{v}_t = \boldsymbol{\omega}_{t+1} - \boldsymbol{\omega}_t,$$
$$\mathbf{r}_t = \nabla\varphi_\mu(\boldsymbol{\omega}_{t+1}) - \nabla\varphi_\mu(\boldsymbol{\omega}_t).$$

The proposed method, quasi-Newton Support Vector Machine (QN-SSVM), is detailed in Algorithm 3. This algorithm represents the core of our method, detailing step-by-step how the QN-SSVM operates, from initialization to the iterative optimization process that enhances its performance over traditional methods.

Algorithm 3 QN-SSVM

Require: For the variable $\boldsymbol{\omega}_0$, we initialize the Hessian approximation $\mathbf{B}_0 = \mathbf{I}$, use a step size ϵ_t that satisfies an Armijo condition, and set T as the maximum number of iterations.

1: **while** $\|\nabla\varphi_\mu(\boldsymbol{\omega}_t)\| > tol$ and $t > T$ **do**
2: compute $\nabla\varphi_\mu(\boldsymbol{\omega}_t)$ using (4.1.7).
3: update

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \epsilon_t(\mathbf{B}_t^{-1})\nabla\varphi_\mu(\boldsymbol{\omega}_t).$$

4: compute $\nabla\varphi_\mu(\boldsymbol{\omega}_{t+1})$ using (4.1.7).
5: update hessian approximation matrix

$$\mathbf{B}_{t+1} = \mathbf{B}_t - \frac{\mathbf{B}_t\mathbf{v}_t\mathbf{v}_t^\top\mathbf{B}_t}{\mathbf{v}_t^\top\mathbf{B}_t\mathbf{v}_t} + \frac{\mathbf{r}_t\mathbf{r}_t^\top}{\mathbf{r}_t^\top\mathbf{v}_t}.$$

6: $t = t + 1$.
7: **end for**

Following the algorithmic description, we provide a Python code snippet that implements the QN-SSVM method. The Python implementation includes initializing parameters, the main optimization loop where updates are made according to the quasi-Newton method, and finally, the conditions under which the algorithm concludes, returning the optimized parameters.

```
import numpy as np
import math
from sklearn.base import BaseEstimator

class QN_SSVM(BaseEstimator):
    #setting initial parameters
    def __init__(self, C = 1, beta = 0.1, sigma = 0.1,
                 alpha_0 = 1, mu = 0.9, tau_1 = 1,
                 tau_2 = 1, epsl_1 = 0.25, epsl_2 = 0.5,
                 max_iter=1000, tol=0.001):

        assert C > 0
        self.C = C
        assert 0 < beta  < 1
        self.beta = beta
        assert 0 < sigma  < 1
        self.sigma  = sigma
        assert 0 <= mu < 1
        self.mu = mu
        assert tau_1 > 0
        self.tau_1 = tau_1
        assert tau_2 > 0
        self.tau_2 = tau_2
        assert epsl_1 > 0
        self.epsl_1 = epsl_1
        assert epsl_2 > 0
        self.epsl_2 = epsl_2
        assert max_iter > 0 and isinstance(max_iter, int)
        self.max_iter=max_iter
        assert tol > 0
        self.tol=tol
        self.alpha_0 = alpha_0
```

```
33
34      ''' cost function (squareHingeloss SVM) '''
35      def cost_function(self,w,x,t):
36          u = (1-t*np.matmul(x,w))
37          p = np.piecewise(u, [
38                           u >= (self.epsl_1/self.tau_1) +
39                          self.tau_1*self.mu,
40                          ((self.epsl_1/self.tau_1)<= u)  &
41                          (u <= (self.epsl_1/self.tau_1)
42                          + self.tau_1*self.mu),
43                          (-(self.epsl_2/self.tau_2) <= u) &
44                          (u <= (self.epsl_1/self.tau_1)),
45                          (-(self.epsl_2/self.tau_2) - self.tau_2*
46                          self.mu <= u) &
47                          (u <= -(self.epsl_2/self.tau_2))],
48                          [lambda u: self.tau_1*(u -
49                                    (self.epsl_1/self.tau_1))-
50                                    (((self.tau_1)**2)*self.mu)/2 ,
51                          lambda u: (u - (self.epsl_1/self.tau_1))
52                                   **2/(2*self.mu),
53                          0,
54                          lambda u: ((u + (self.epsl_2/
55                                     self.tau_2))**2)/
56                                     (2 * self.mu),
57                          lambda u:  -self.tau_2*(u + (self.epsl_2/
58                                     self.tau_2))-(((self.tau_2)**2)
59                                     *self.mu)/2 ])
60
61          loss = self.C*(np.mean(p))
62          cost = 1/2 * np.linalg.norm(w)**2  + loss
63          return cost
64
65      def gradient_in(self,margin):
66          u = 1 - margin
67          p = np.piecewise(u, [
68              u >= (self.epsl_1/self.tau_1) + self.tau_1*self.mu,
69              ((self.epsl_1/self.tau_1) <= u) &
70              (u <= (self.epsl_1/self.tau_1)
71              + self.tau_1*self.mu),
72              (-(self.epsl_2/self.tau_2) <= u) &
```

```
            (u <= (self.epsl_1/self.tau_1)),
            (-(self.epsl_2/self.tau_2) -
            self.tau_2*self.mu <= u) &
            (u <= -(self.epsl_2/self.tau_2))
            ], [
            self.tau_1,
            lambda u: (u - (self.epsl_1/self.tau_1))/(self.mu),
            0,
            lambda u: (u + (self.epsl_2/self.tau_2))/(self.mu),
            -self.tau_2
            ])
        return p

    ''' approximate ResNaq_BFGS matrix '''
    def Update_RES_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
            return p, u, B_new

    def update_step_size(self,w,d,grad,x,t) -> np.ndarray:
        """ Armijo rule. """

        h = lambda a : self.cost_function(w + a * d,x,t) - \
                       self.cost_function(w,x,t)
        assert h(0) == 0.

        # TODO: What if g >= 0 ? set g = 0 ?
        g = grad.dot(d)

        alpha = self.alpha_0

```

```
113         if h(alpha) > self.sigma * g * alpha:
114             # Decrease alpha.
115             while True:
116                 alpha = alpha * self.beta
117                 if h(alpha) <= self.sigma * g * alpha:
118                     self.alpha = alpha
119                     break
120         else:
121             # Increase alpha.
122             while True:
123                 prev_alpha = alpha
124                 alpha = alpha / self.beta
125                 if h(alpha) > self.sigma * g * alpha:
126                     self.alpha = prev_alpha
127                     break
128 
129         return self.alpha
130 
131     def fit(self, x, t):
132         # checks for labels
133         self.classes_ = np.unique(t)
134         t[t==0] = -1
135 
136         #initail variables k, stop, w_0
137         k = 1
138         update = True
139         w = np.zeros(len(x[0])+1)
140         w_ = np.zeros(len(x[0])+1)
141 
142         #set up matrices
143         B = np.identity(len(x[0])+1)  #inverse hessian
144         I = np.identity(len(x[0])+1)  #identity
145         X = np.ones((len(x),len(x[0])+1))
146         X[:,:-1] = x
147 
148         obj_res = []
149         iter_res = []
150 
151         while k <=(self.max_iter) and update :
152             #compute gradient
```

```
153             margin = t*np.matmul(X,w)
154             p = self.gradient_in(margin)
155
156             grad = w - self.C*(np.mean(p*t*X.T,axis=1))
157
158             if  np.linalg.norm(grad)<=self.tol:
159                 update = False
160                 print('DONE')
161             else:
162                 # update hessian matrix
163                 H = np.linalg.inv(B)
164
165                 #update step size (Amijo)
166                 d = -np.matmul(H,grad)
167                 eta = self.update_step_size(w,d,grad,X,t)
168
169                 # update weight
170                 w_new = w - eta * np.matmul(H,grad)
171
172                 #compute new gradient
173                 margin = t*np.matmul(X,w_new)
174                 p = self.gradient_in(margin)
175
176                 grad_new = w_new - self.C*
177                         (np.mean(p*t*X.T,axis=1))
178
179                 # get difference of values
180                 dw = w_new - w
181                 # get difference of gradients
182                 dg = grad_new - grad
183
184                 _, _, B_new = self.Update_RES_BFGS(B, dw,
185                                                    dg, I)
186
187                 # update step
188                 B = B_new
189                 w = w_new
190
191                 k += 1
192
```

```
193         self.final_iter = k
194         self._coef = w[:-1]
195         self._intercept = w[-1]
196
197
198
199     def predict(self, x):
200         p = np.sign(np.matmul(x,self._coef)+self._intercept)
201         p[p==0] = 1
202         return p.astype(int)
```

Theorem 4.1.4 (Convergence analysis). Let $\{\omega_t\}$ be the sequence generated by Algorithm 1 and ω_μ^* be a solution of problem (4.1.4). Then, the sequence $\{\omega_t\}$ converges to ω_μ^*.

Proof. Since we know that the objective function of Eq.(4.1.4) is strongly convex and infinitely continuously differentiable, it follows from (2.1.10) that $\nabla^2\varphi_\mu(\omega)$ is a positive definite for all $\omega \in \mathbb{R}^{n+1}$. By [26], we can guarantee Assumption 2.3.1 hold. The convergence of $\{\omega_k\}$ can also be obtained easily by Theorem 2.3.2 and will be omitted. □

Remark 4.1.5. In comparison between the original SVM [7] and QN-SSVM, the original SVM needs to construct a model by transforming the primal problem into a dual problem to obtain the solutions. While the QN-SSVM only needs to solve the unconstrained optimization problem (4.1.4) in primal by using the quasi-Newton–Armijo algorithm.

4.1.2 Quasi-Newton smooth generalized pinball support vector machine with non-linear kernel

In this section, we extend our approach to the non-linear case via kernel method. Instead of having direct access to the feature vector data $\mathbf{x}$, the advantage of support vector machines is that they can employ an approximate feature map. Let us consider $K(\cdot,\cdot)$ to be a kernel function, then we will solve the unconstrained

optimization problem below to obtain the hyperplane $K(\mathbf{x}^\top, X^\top)\mathbf{z} + b = 0$.

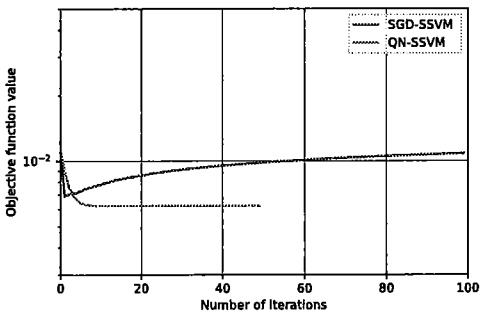
$$\min_{\boldsymbol{\nu}} \quad \frac{1}{2}\|\boldsymbol{\nu}\|^2 + \frac{c}{m}\sum_{i=1}^{m} Q_{\tau_1,\tau_2}^{\epsilon_1,\epsilon_2}(1 - y_i K(\mathbf{x}_i, X^\top)\boldsymbol{\nu}, \mu), \tag{4.1.10}$$

where $\boldsymbol{\nu} = [\mathbf{z}^\top, b]^\top \in \mathbb{R}^{m+1}$. As previously stated, Algorithm 1 can be used to compute the solution of problem (4.1.10) similarly to the linear case. As a result, the details have been skipped. Additionally, the class label y of a new sample $\mathbf{x}$ can be predicted using the same decision rule as in the linear case.

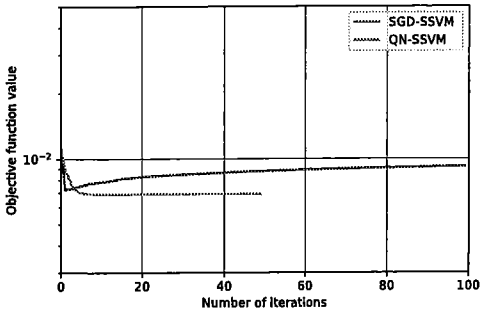
4.1.3 Numerical experiments

In this section, we will establish the performance of our proposed QN-SSVM method. The experiments have been performed on Python3 on a macOS with an Intel i5 Processor 2.3 GHz memory 8 GB 2133 MHz LPDDR3. The following will illustrate the motivation for using the quasi-Newton BFGS method to solve our problem (4.1.4).

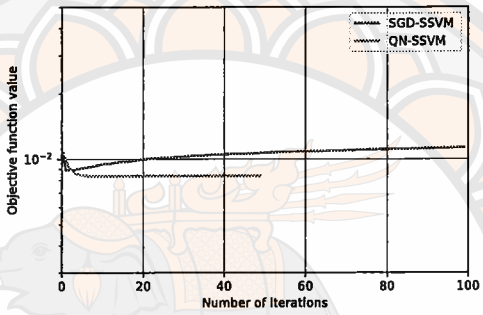
For some benchmark datasets, we compare the performance of the quasi-Newton-Armijo method with the stochastic gradient descent method for solving our problem (4.1.4). The results are shown in Figure 22. Figure 22 illustrates the rate of convergence of objective function (4.1.4) with respect to the number of iterations. As is clear from Figure 22 between the stochastic gradient smooth generalized pinball SVM (SG-SSVM) and the quasi-Newton-Armijo smooth generalized pinball SVM (QN-SSVM), the convergence rates of QN-SSVM are certainly better than SG-SSVM. As a result, compared to the stochastic gradient descent method, the quasi-Newton-Armijo method is much more effective at finding reliable directions for convergence when solving optimization problems.



(a) Make blobs dataset.



(b) Make moons dataset.



(c) Make classification dataset.

Figure 22 Convergence analysis for SG-SSVM, QN-SSVM.

4.1.4 Performance on UCI datasets

In this section, we study the performance of our proposed method using several benchmark datasets, which are binary class datasets and multi-class datasets. To show the classification efficiency of the proposed method, we evaluate it on some real-world binary datasets from the UCI machine repository [34]. Table 4 summarizes all the binary datasets. In the following experiments, we compare our proposed QN-SSVM method with the following related classification methods in terms of accuracy and computational time.

1. The original support vector machine [9]. We refer to this method as SVM.
2. The stochastic gradient support vector machine with hinge loss function model [10]. We refer to this method as SG-SVM.
3. The stochastic gradient support vector machine with generalized pinball loss

function model [37]. We refer to this method as SG-GSVM.

For hyper-parameters τ_1,τ_2, ϵ_1, and ϵ_2 in the generalized pinball loss function, we have tuned these hyper-parameters by using the grid search technique [35]. We have chosen from the sets $\{0.5, 0.6, 0.7, 0.75, 1\}$, $\{0.1, 0.4, 0.75\}$, $\{0.01, 0.05, 0.1, 0.25, 0.45, 0.5, 0.6\}$ and $\{0.5, 1, 1.5, 1.75\}$, respectively. In the nonlinear case, we use the RBF kernel, which is defined as $K(\mathbf{x}, \mathbf{y}) = \exp\{-\gamma\|\mathbf{x} - \mathbf{y}\|^2\}$. The results of the experiment are shown in Table 5 and Table 6. The methods with the highest performance are in bold.

To evaluate the performance of classifiers, we have used testing accuracy and reported the standard deviation of testing accuracy obtained over a 5-fold run of each algorithm. Tables 5 and 6 show that our proposed QN-SSVM algorithm performs better than other models and achieves the best accuracy on 11 of 13 datasets. Also, the QN-SSVM method performs better in generalization ability than other models. This can be partly related to the QN-SSVM model's improved smooth objective function convergence over other non-smooth functions. Further, Tables 5 and 6 also show the outcomes of the training time. We observed that the computational time depends on the amount of data. The SVM takes less computational time than our proposed method in the amount of data being small as a state in the Australian dataset, which has 690 samples, taking 0.0188s in SVM and 0.0326s in our proposed method. But if the data is large, our proposed method will take less time, as we can see in Table 5, the Ring, monk2, Titanic, and Twonorm datasets. Therefore, for most of the results, we can see that the execution time of our proposed QN-SSVM method takes less training time than the other methods. Although our method requires inverse computing of the hessian matrix on each iteration, our method converges faster than other methods, resulting in less training time for the QN-SSVM method.

In order to analyze the statistical performances of four methods on various datasets, the Friedman test is used, which has the corresponding post hoc test and is considered a simple, nonparametric, yet safe test. As seen in Tables 7 and 8, we calculate the rank of four methods on accuracy for all datasets. Suppose all the

Table 4 The details of binary datasets.

Datasets	No. of samples	Dimension
Australian	690	14
Coli2000	177	44
Diabetes	768	8
Heart	270	13
Ionosphere	351	33
Monk2	432	6
Pima Indians	768	8
Phoneme	5404	5
Seheart	462	9
Spectf Heart	267	44
Titanic	2201	41
Twonorm	7400	20
Ring	7400	20

methods are equivalent; we derive the Friedman statistic by

$$x_F^2 = \frac{12m}{k(k+1)}\left[\sum_j \left(\frac{1}{m}\sum_i r_i^j\right)^2 - \frac{k(k+1)^2}{4}\right], \tag{4.1.11}$$

where r_i^j represents the j-th of k methods on the i-th of m datasets. Friedman's x_F^2 is undesirably conservative and comes up with a better statistic

$$F_F = \frac{(m-1)x_F^2}{m(k-1)-x_F^2}, \tag{4.1.12}$$

which is distributed according to the F-distribution with $k-1$ and $(k-1)(m-1)$ degrees of freedom. Using Equations (4.1.11) and (4.1.12), we derive $x_F^2 = 11.37$ and $F_F = 5.49$ in the case of Linear kernel and $x_F^2 = 12.81$ and $F_F = 7.25$ in the case of RBF kernel. In the case of the Linear kernel, the critical value of $F(3, 27)$ is 2.96 for the level of significance 0.05. In the case of the RBF kernel, the critical value of $F(3, 24)$ is 3.01 for the level of significance 0.05. As a result, we observed that the value of F_F in both cases is larger than the corresponding critical values. Therefore, there is a significant difference between the four algorithms. Moreover,

Table 5 Performance of different algorithms using a linear kernel.

Datasets	Mean Accuracy±S.D. (%)			
	SVM	**SG-SVM**	**SG-GSVM**	**QN-SSVM**
			$\tau_1, \tau_2,$	$\tau_1, \tau_2,$
			ϵ_1, ϵ_2	$\epsilon_1, \epsilon_2, \mu$
Australian	85.51 ± 1.89	85.36 ± 2.02	85.36 ± 2.02	**87.10 ± 1.68**
Time(s)	**0.0188**	0.0730	0.1208	0.0326
			0.5,0.1,	0.7,0.4,
			0.25,0.5	0.6,1.75,0.5
Coli2000	94.03±0.50	94.00±0.45	94.03±0.50	93.84±0.44
Time(s)	5.3374	**0.1122**	0.1700	0.2021
			0.5,0.1,	0.5,0.1,
			0.25,0.5	0.25,0.5,0.01
Diabetes	76.82±2.41	76.30±2.86	76.56±2.86	**77.73 ± 4.27**
Time(s)	**0.0094**	0.0710	0.1124	0.0423
			0.5,0.1,	0.6,0.75,
			0.25,0.5	0.45,1.75,0.9
Monk2	80.54±7.36	80.54±7.36	80.54±7.36	**80.57±2.59**
Time(s)	0.0772	0.0772	0.1104	**0.0617**
			0.5,0.1,	0.5,0.1,
			0.25,0.5	0.25,0.5,0.01
Phoneme	**77.48±0.74**	76.85±0.62	77.13±0.23	77.42±0.75
Time(s)	0.3936	0.0692	0.1192	**0.0319**
			0.5,0.1,	0.75,0.75,
			0.25,0.5	0.01,1.75,0.9
Seheart	71.64±3.90	72.72±5.88	72.29±4.61	**73.80±5.08**
Time(s)	**0.0100**	0.0714	0.1082	0.0314
			0.5,0.1,	0.7,0.4,
			0.25,0.5	0.6,1.75,0.1
Spectfheart	80.52±6.80	80.16±4.50	79.42±4.82	**80.89±3.67**
Time(s)	**0.0030**	0.0782	0.1272	0.0311
			0.5,0.1,	0.7,0.4,
			0.25,0.5	0.6,1,0.9
Titanic	77.60±2.66	77.60±6.32	77.60±2.67	**77.83±1.70**
Time(s)	0.0909	0.0632	0.1011	**0.0212**
			0.5,0.1,	0.75,0.1,
			0.25,0.5	0.25,0.5,0.5
Twonorm	97.78±0.76	97.76±0.65	97.78±0.59	**97.89±0.41**
Time(s)	0.1478	0.0708	0.1151	**0.0609**
			0.75,0.75,	0.75,0.5,
			0.25,0.5	0.25,0.5,0.9
Ring	76.23±1.56	76.24±1.48	76.27±1.36	**76.32±1.59**
Time(s)	2.3166	0.0906	0.1348	**0.0434**
			0.5,0.1,	1,0.1,
			0.25,0.5	0.01,1.5,0.9

Table 6 Performance of different algorithms using a RBF kernel.

Datasets	Mean Accuracy±S.D. (%)			
	SVM	**SG-SVM**	**SG-GSVM**	**QN-SSVM**
			$\tau_1, \tau_2,$	$\tau_1, \tau_2,$
			ϵ_1, ϵ_2	$\epsilon_1, \epsilon_2, \mu$
Australian	54.20±2.69	55.36±2.85	55.51±2.88	**55.65±1.55**
$\gamma = 100$				
Time(s)	0.2731	**0.1387**	0.2026	0.4131
			0.5,0.1,	0.1,0.1,
			0.25,0.5	0.6,1.75,0.1
Coli2000	**94.03±0.50**	93.86±0.36	94.02±0.51	**94.03±0.50**
$\gamma = 0.01$				
Time(s)	1.9266	0.0534	**0.0405**	1.1969
			0.5,0.1,	1.5,0.1,
			0.25,0.5	1,0.01,0.5
Heart	82.22±4.16	82.59±4.63	81.85±4.44	**82.96±4.89**
$\gamma = 0.1$				
Time(s)	**0.0046**	0.1900	0.2377	0.0489
			0.5,0.1,	0.5,0.1,
			0.25,0.5	0.25,0.5,0.9
Ionosphere	85.20±3.82	92.02±2.94	91.17±2.46	**93.73±2.94**
$\gamma = 0.1$				
Time(s)	**0.0033**	0.1428	0.2222	0.2810
			0.5,0.1,	0.75,0.1,
			0.25,0.5	0.25,0.5,0.5
Monk2	95.36±2.66	67.12±7.56	71.55±7.53	**97.22±0.93**
$\gamma = 0.1$				
Time(s)	**0.0054**	0.0206	0.0297	0.2131
			0.5,0.1,	0.7,0.4,
			0.25,0.5	0.6,1,0.1
Pima	58.21±1.87	58.60±4.41	59.51±3.82	**60.16±1.50**
$\gamma = 0.1$				
Time(s)	0.2170	**0.1562**	0.2171	0.2692
			0.5,0.1,	0.75,0.1,
			0.25,0.5	0.01,0.5,0.01
Ring	49.76±1.44	49.92±1.26	49.76±1.61	**50.24±0.41**
$\gamma = 10$				
Time(s)	62.2964	**0.2902**	0.3992	1.9581
			0.5,0.1,	0.75,0.1,
			0.25,0.5	0.05,0.5,0.9
Spectfheart	78.64±2.31	79.39±1.77	79.39±1.77	**79.41±4.71**
$\gamma = 10$				
Time(s)	0.0021	0.1529	0.1957	**0.0200**
			0.5,0.1,	0.7,0.4,
			0.25,0.5	1,1,0.9
Titanic	**78.55±1.89**	77.78±1.54	77.78±1.74	78.33±1.69
$\gamma = 0.1$				
Time(s)	**0.1247**	0.1766	0.2719	0.7449
			0.75,0.75,	0.75,0.5,
			0.25,0.5	0.25,0.5,0.01

Table 7 Average ranks of various algorithms attained by linear kernel.

Dataset	SVM	SG-SVM	SG-GSVM	QN-SSVM
Australian	2	3.5	3.5	1
Coli2000	1.5	3	1.5	4
Diabetes	2	4	3	1
Monk2	3	3	3	1
Phoneme	1	4	3	2
Seheart	4	2	3	1
Spectfheart	2	3	4	1
Titanic	3	3	3	1
Twonorm	2.5	4	2.5	1
Ring	4	3	2	1
Average Rank	3	2.89	2.94	**1.17**

in Tables 7 and 8, the average rank of the QN-SSVM approach is lower than that of other methods. It implies that our proposed method outperforms the other four methods.

Furthermore, we contrast our results with those of other methods using the same dataset. Note that the different cross-validation procedures can not directly compare the results; however, they can still be compared in some respects. As seen in Table 9, we compare our method with the following recent classification methods in terms of accuracy.

1. The smooth pinball loss nonparallel support vector machine model [38]. We refer to this method as SpinNSVM.

2. The RES-smooth generalized pinball support vector machine model [21]. We refer to this method as RES-SGSVM.

3. The twin support vector machine with generalized pinball loss function model [39]. We refer to this method as GPin-TSVM.

Table 8 Average ranks of various algorithms attained by RBF kernel.

Dataset	SVM	SG-SVM	SG-GSVM	QN-SSVM
Australian	4	3	2	1.5
Coli2000	1.5	4	3	1.5
Heart	3	2	4	1
Ionosphere	4	2	3	1
Monk2	2	4	3	1
Pima	4	3	2	1
Ring	3.5	2	3.5	1
Spectfheart	4	2.5	2.5	1
Titanic	1	3.5	3.5	2
Average Rank	2.5	3.25	2.85	**1.4**

Table 9 shows the experimental results from five datasets, with the Australian, Spectfheart, and Twonorm datasets experimentally using the linear kernel and the rest using the RBF kernel. As a result, we can see that our QN-SSVM method has the highest accuracy in almost datasets. Note that N\A indicates that no experiment was performed on the dataset in the the associated article. However, the available comparative results demonstrate that our method outperforms other methods.

For the classification of multi-class datasets, we also use real-world multi-class datasets from the UCI machine repository. Table 10 summarizes all the multi-class datasets. In this experiment, we use the same parameter range as in the previous section. The result of the experiment is shown in Table 3.

According to all of the results, our proposed QN-SSVM method outperforms other comparable models regarding accuracy. Furthermore, we analyze the statistical performances of three methods on various datasets. As seen in Table 12, we calculate the rank of three methods on accuracy for all datasets. Using Equations (4.1.11) and (4.1.12), we derive $x_F^2 = 4.5$ and $F_F = 3.86$. The critical value of

Table 9 Comparative results.

Dataset	SpinNSVM	RES-SGSVM	GPin-TSVM	QN-SSVM
Australian	86.41±0.97	N\A	86.81±3.69	**87.10±1.68**
Spectfheart	79.85±1.63	N\A	N\A	**80.89±3.67**
Twonorm	97.84±0.15	97.85±0.49	N\A	**97.89±0.41**
Ionosphere*	92.34±0.64	93.48±2.03	**95.15±4.05**	93.73±2.94
Titanic*	78.16±0.35	N\A	N\A	**78.33±1.69**

* non-linear classifiers (RBF kernel).

$F(2,6)$ is 1.66 for the level of significance 0.05. As a result, we observed that the value of F_F is larger than the corresponding critical value. Therefore, there is a significant difference between the three algorithms. In addition, Table 12 shows that the QN-SSVM has a lower average rank than other methods, and QN-SSVM also performs best in almost all datasets. It implies that, as a result, our proposed algorithm performs better than the other three methods. Moreover, it was clear from this experiment that our method used less training time than the others.

Table 10 The details of multi-class datasets.

Datasets	No. of classes	No. of samples	Dimension
Car	4	1728	8
Glass	6	214	10
Seed	5	2310	19
Teaching eval.	3	151	5

Table 11 Multi-classification results of different algorithms using a linear kernel.

Datasets	Mean Accuracy±S.D. (%)		
	SG-SVM	**SG-GSVM**	**QN-SSVM**
	c	$c, \tau_1, \tau_2,$ ϵ_1, ϵ_2	$c, \tau_1, \tau_2,$ $\epsilon_1, \epsilon_2, \mu$
Car	78.59±2.64	76.97±3.12	**79.34±0.47**
Time(s)	4.4988	6.0891	**0.2967**
	0.1	10,1,0.75, 0.5,0.5	10,1,1, 0.25,0.5,0.9
Glass	57.06±2.27	57.60±10.96	**71.98±9.37**
Time(s)	4.0290	6.0883	**0.4007**
	10	1,1,0.75, 0.1,0.25	10,1,1, 0.25,0.5,0.9
Seed	91.90±4.29	92.38±4.86	**93.33±3.81**
Time(s)	1.2079	2.6738	**0.7554**
	100	10,1,0.75, 0.5,0.5	10,0.75,1, 0.5,0.5,0.01
Teaching eval.	50.92±6.85	**54.92±4.84**	54.28±4.62
Time(s)	1.5242	3.0526	**0.1288**
	100	1,1,0.75, 0.1,0.25	10,0.75,1, 0.5,0.5,0.9

4.1.5 Noise Insensitivity

This section highlights how our smooth generalized pinball loss function retains the noise insensitivity characteristic inherent to the generalized pinball loss function. We incrementally introduced noise levels in our experimental setup, starting from $r = 0.5$ to $r = 0.2$. Here, the term noise refers to the mean level of feature noise incorporated into the dataset. The objective was to evaluate the robustness

Table 12 Average ranks of various algorithms on multi-classification.

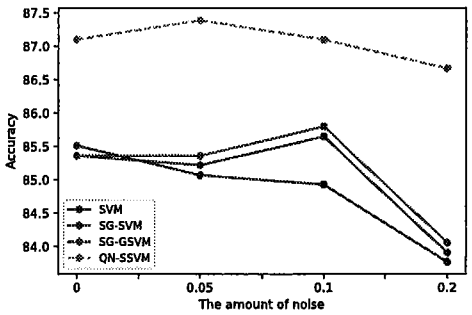
Dataset	SG-SVM	SG-GSVM	QN-SSVM
Car	2	3	1
Glass	3	2	1
Seed	3	2	1
Teaching eval.	3	1	2
Average Rank	2.75	2	**1.25**

of various SVM methodologies—namely, the standard SVM, SG-SVM, SG-GSVM, and QN-SSVM—by observing their accuracy in scenarios with increasing noise points.

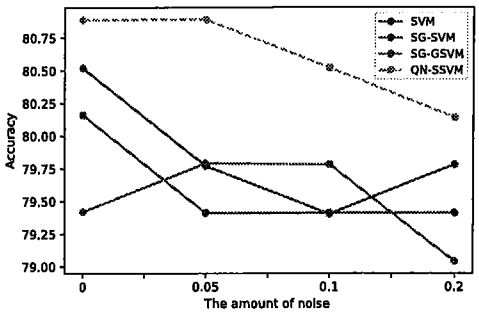
Below is the Python code used to inject feature noise into the dataset, with the results shown in Figure 23.

```
import numpy as np

def add_noise(x, r=0.05):
    """
    Adds Gaussian noise to the dataset.

    Parameters:
    - x: The dataset (features).
    - r: The standard deviation of the Gaussian noise.

    Returns:
    - The dataset with added Gaussian noise.
    """
    m, n = x.shape
    return x + np.random.normal(0, r, (m, n))

# Example usage
x = add_noise(x, r=0.05)
```

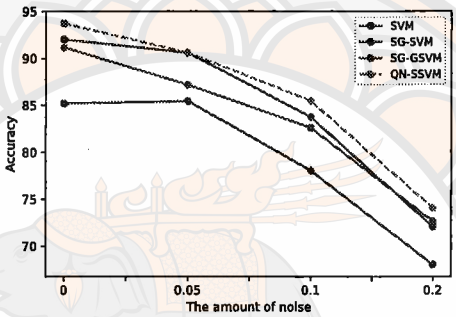
From Figure 23, we can see that with the increase in noise, the accuracy of all



(a) Australian(a)



(b) Spectheart(a)



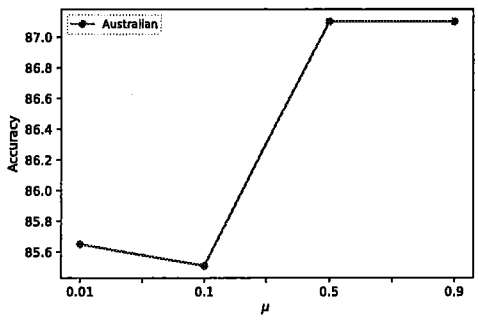
(c) Ionosphere(b)

Figure 23 Testing comparisons on the benchmark UCI dataset with a level of noise.

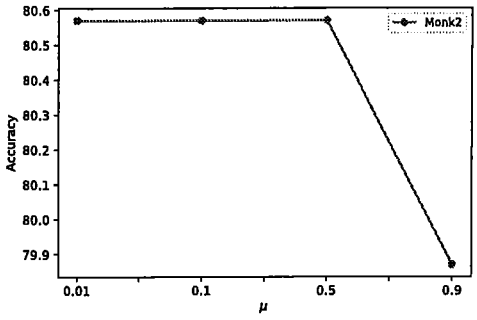
algorithms decreases. However, QN-SSVM achieves the best accuracy, followed by SG-GSVM. As a result, we can conclude that even in the presence of noise, our proposed QN-SSVM still performs accurately. Then, it implies that our smooth generalized pinball loss function successfully preserves the noise insensitivity property of the generalized pinball loss function.

4.1.6 Discussions on choice of parameter μ

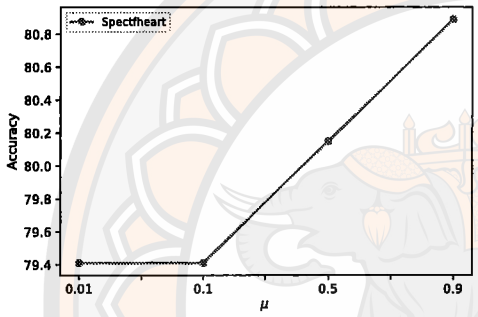
One essential parameter in introducing our smooth generalized pinball loss function is μ. This experiment will show that parameter μ has affected performance. The result of the experiment is depicted in Figure 24.



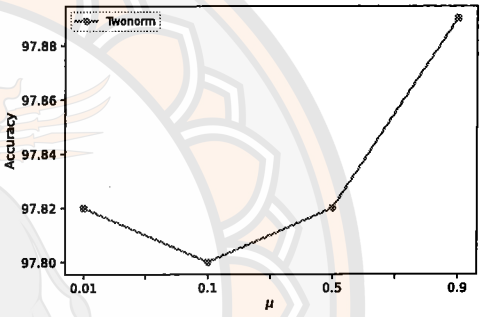
(a) Australian



(b) Monk-2



(c) Spectfheart



(d) Twonorm

Figure 24 Performance of the QN-SSVM with different values of μ.

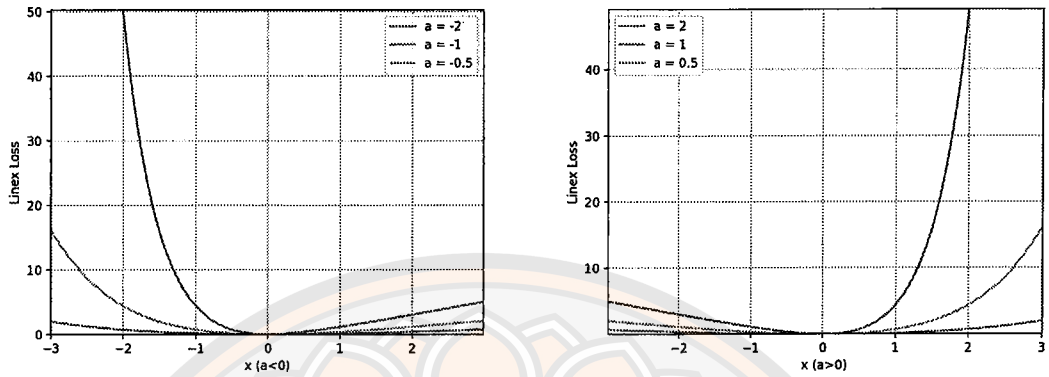
4.2 Efficient large-scale classification with Linex least square twin bounded support vector machine

To enhance the efficacy of LSTBSVM, we propose a novel LSTBSVM model utilizing an asymmetric linear-exponential loss (Linex), called LSTBSVM-linex. Specifically, the function of Linex loss [40] is defined as:

$$L(x) = \exp(ax) - ax - 1 \tag{4.2.1}$$

where $a \neq 0$ is a hyper-parameter, determining the steepness. In Figure 25, examples of Linex loss are depicted with different a values. The sign of a determines the curve's direction: for $a < 0$, the left side of the curve is steeper, resulting in heavier penalties for negative points, even if their values are the same as positive points. As $x \to -\infty$, the loss $L(x)$ increases exponentially, while it increases linearly as

$x \to +\infty$. However, for $a > 0$, the scenario is reversed.



(a) (b)

Figure 25 The illustration of Linex loss function with different value of a.

4.2.1 Methodology

In this subsection, we outline the details of the LSTBSVM-linex model. We introduce its problem statement and describe the algorithm used to solve the LSTBSVM-linex model. Furthermore, the following section will present the numerical results derived from large-scale benchmark and X-ray image datasets.

Denote $\boldsymbol{\omega}_1 = [\mathbf{w}_1^\top \ b_1]^\top$, $\boldsymbol{\omega}_2 = [\mathbf{w}_1^\top \ b_2]^\top$, $\mathbf{x}_i^+ = [\mathbf{x}_i^{+\top} \ 1]^\top$ and $\mathbf{x}_j^- = [\mathbf{x}_j^{-\top} \ 1]^\top$. The problems of LSTBSVM-linex are defined as follows:

$$\begin{aligned} \min_{\boldsymbol{\omega}_1, q_j} \ & \frac{c_3}{2}\|\boldsymbol{\omega}_1\|^2 + \frac{1}{2}\sum_{i=1}^{m_1}(\boldsymbol{\omega}_1^\top x_i^+)^2 + \frac{c_1}{2}\sum_{j=1}^{m_2}(\exp(aq_j) - aq_j - 1) \\ \text{s.t.} \ & q_j = 1 + \boldsymbol{\omega}_1^\top x_j^-, \quad j = 1, 2, \ldots, m_2 \end{aligned} \tag{4.2.2}$$

and

$$\begin{aligned} \min_{\boldsymbol{\omega}_2, b_2, p_i} \ & \frac{c_4}{2}\|\boldsymbol{\omega}_2\|^2 + \frac{1}{2}\sum_{j=1}^{m_2}(\boldsymbol{\omega}_2^\top x_j^-)^2 + \frac{c_2}{2}\sum_{i=1}^{m_1}(\exp(ap_i) - ap_i - 1) \\ \text{s.t.} \ & p_i = 1 - \boldsymbol{\omega}_2^\top x_i^+, \quad i = 1, 2, \ldots, m_1. \end{aligned} \tag{4.2.3}$$

Then, the problems (4.2.2) and (4.2.3) can be transformed to the unconstrained

problems as

$$\min_{\omega_1} \quad \frac{c_3}{2}\|\omega_1\|^2 + \frac{1}{2}\sum_{i=1}^{m_1}(\omega_1^\top x_i^+)^2 + \frac{c_1}{2}\sum_{j=1}^{m_2}(\exp\left(a(1+\omega_1^\top \mathbf{x}_j^-)\right) - a(1+\omega_1^\top \mathbf{x}_j^-) - 1) \tag{4.2.4}$$

and

$$\min_{\omega_2} \quad \frac{c_4}{2}\|\omega_2\|^2 + \frac{1}{2}\sum_{j=1}^{m_2}(\omega_2^\top x_j^-)^2 + \frac{c_2}{2}\sum_{i=1}^{m_1}(\exp\left(a(1-\omega_2^\top \mathbf{x}_i^+)\right) - a(1-\omega_2^\top \mathbf{x}_i^+) - 1). \tag{4.2.5}$$

To handle large-scale problems, we adopt the Adam optimization method [8] for model solving. Adam is a recent and popular variant of the gradient descent version. Adam uses the first and second moments to calculate the adaptive learning rates for each iteration. In the second moment, Adam acquires the exponentially decaying average of previous gradients and the exponentially decaying average of the square of previous gradients. The moment can be compared to a ball rolling down a slope, with Adam acting like a heavy ball with friction that prefers flat minima on the error surface.

For applying the Adam optimization method in the proposed LSTBSVM-linex, we again write problems (4.2.4) and (4.2.5) as follows

$$\begin{aligned} F_1(\omega_1) := \frac{c_3}{2}\|\omega_1\|^2 + \frac{1}{2m_1}\sum_{i=1}^{m_1}(\omega_1^\top x_i^+)^2 \\ + \frac{c_1}{2m_2}\sum_{j=1}^{m_2}(\exp\left(a(1+\omega_1^\top \mathbf{x}_j^-)\right) - a(1+\omega_1^\top \mathbf{x}_j^-) - 1), \end{aligned} \tag{4.2.6}$$

$$\begin{aligned} F_2(\omega_2) := \frac{c_4}{2}\|\omega_2\|^2 + \frac{1}{2m_2}\sum_{j=1}^{m_2}(\omega_2^\top x_j^-)^2 \\ + \frac{c_2}{2m_1}\sum_{i=1}^{m_1}(\exp\left(a(1-\omega_2^\top \mathbf{x}_i^+)\right) - a(1-\omega_2^\top \mathbf{x}_i^+) - 1) \end{aligned} \tag{4.2.7}$$

Then, for the t-th iteration, we randomly choose L pairs of points $\{x_i^+\}_i^L$ and $\{x_j^-\}_j^L$ and compute the gradient of objective functions $F_1(\omega_1)$ and $F_2(\omega_2)$ with respect to ω_1 and ω_2 as

$$\nabla F_1(\omega_1) = c_3\omega_1 + \frac{1}{L}\sum_{i=1}^{L}(\omega_1^\top \mathbf{x}_i^+)\cdot \mathbf{x}_i^+ \tag{4.2.8}$$

$$+\frac{c_1}{2L}\sum_{j=1}^{L}\left(\exp\left(a(1+\boldsymbol{\omega}_1^\top \mathbf{x}_j^-)\right)\cdot a\mathbf{x}_j^- - a\mathbf{x}_j^-\right),$$

$$\nabla F_2(\boldsymbol{\omega}_2) = c_4\boldsymbol{\omega}_2 + \frac{1}{L}\sum_{j=1}^{L}(\boldsymbol{\omega}_2^\top \mathbf{x}_j^-)\cdot \mathbf{x}_j^- \tag{4.2.9}$$
$$+\frac{c_2}{2L}\sum_{i=1}^{L}\left(\exp\left(a(1-\boldsymbol{\omega}_2^\top \mathbf{x}_i^+)\right)\cdot(-a\mathbf{x}_i^+) + a\mathbf{x}_i^+\right).$$

Hence, the parameters $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ can be alternately updated using the Adam optimization algorithm. At iteration $t=0$, the updates are as follows:

For $\boldsymbol{\omega}_{1,t+1}$:

$$\mathbf{m}_{1,t+1} = \beta_{1,t}\mathbf{m}_{1,t} + (1-\beta_{1,t})\nabla F_1(\boldsymbol{\omega}_{1,t}) \tag{4.2.10}$$
$$\mathbf{v}_{1,t+1} = \beta_2\mathbf{v}_{1,t} + (1-\beta_2)(\nabla F_1(\boldsymbol{\omega}_{1,t}))^2 \tag{4.2.11}$$
$$\hat{\mathbf{m}}_{1,t+1} = \frac{\mathbf{m}_{1,t+1}}{1-(\beta_1)^{t+1}} \tag{4.2.12}$$
$$\hat{\mathbf{v}}_{1,t+1} = \frac{\mathbf{v}_{1,t+1}}{1-(\beta_2)^{t+1}} \tag{4.2.13}$$
$$\boldsymbol{\omega}_{1,t+1} = \boldsymbol{\omega}_{1,t} - \frac{\eta_{1,t+1}\hat{\mathbf{m}}_{1,t+1}}{\sqrt{\hat{\mathbf{v}}_{1,t+1}}+\epsilon}, \tag{4.2.14}$$

and for $\boldsymbol{\omega}_{2,t+1}$:

$$\mathbf{m}_{2,t+1} = \beta_{1,t}\mathbf{m}_{2,t} + (1-\beta_{1,t})\nabla F_2(\boldsymbol{\omega}_{2,t}) \tag{4.2.15}$$
$$\mathbf{v}_{2,t+1} = \beta_2\mathbf{v}_{2,t} + (1-\beta_2)(\nabla F_2(\boldsymbol{\omega}_{2,t}))^2 \tag{4.2.16}$$
$$\hat{\mathbf{m}}_{2,t+1} = \frac{\mathbf{m}_{2,t+1}}{1-(\beta_1)^{t+1}} \tag{4.2.17}$$
$$\hat{\mathbf{v}}_{2,t+1} = \frac{\mathbf{v}_{2,t+1}}{1-(\beta_2)^{t+1}} \tag{4.2.18}$$
$$\boldsymbol{\omega}_{2,t+1} = \boldsymbol{\omega}_{2,t} - \frac{\eta_{2,t+1}\hat{\mathbf{m}}_{2,t+1}}{\sqrt{\hat{\mathbf{v}}_{2,t+1}}+\epsilon}, \tag{4.2.19}$$

where the initial values for $\mathbf{m}_0$ and $\mathbf{v}_0$ are set to $\mathbf{0}$. This method utilizes moving averages of both the gradients ($\mathbf{m}$) and the squared gradients ($\mathbf{v}$) to adjust the learning rates for each parameter, with $\beta_{1,t}$, β_2, $\eta_{1,t+1}$, and $\eta_{2,t+1}$ controlling the decay rates and learning rates respectively. The use of $\hat{\mathbf{m}}$ and $\hat{\mathbf{v}}$ ensures that the estimates are unbiased even at the initial iterations. The ϵ term is included for

numerical stability, preventing division by zero. The proposed method, LSTBSVM-linex optimized by Adam, is outlined in detail in Algorithm 4. This algorithm represents the core of our method, detailing step-by-step how the LSTBSVM-linex operates, from initialization to the iterative optimization process that enhances its performance over traditional methods.

Following the algorithmic description, we provide a Python code snippet that implements the LSTBSVM-linex method. The Python implementation includes initializing parameters, the main optimization loop where updates are made according to the Adam, and finally, the conditions under which the algorithm concludes, returning the optimized parameters.

Algorithm 4 Adam for the LSTBSVM-linex

Data: $\eta_t := \frac{\eta}{\sqrt{t}}$, $\beta_1, \beta_2 \in (0,1)$ as decay rate for the moment estimates, $\beta_{1,t} := \beta_1 \lambda^{t-1}$ with $\lambda \in (0,1)$, $\epsilon > 0$, $\boldsymbol{\omega}_{(1,0)}, \boldsymbol{\omega}_{(2,0)}$ as the initial weight vector and T as max iteration.

Set : $\mathbf{m}_0 = \mathbf{0}$ as initial 1^{st} moment vector

Set : $\mathbf{v}_0 = \mathbf{0}$ as initial 2^{nd} moment vector

Set : $t = 0$ as initial time stamp

while $(\|\nabla F_1(\boldsymbol{\omega}_1^{(t)})\| > tol \,\&\, \|\nabla F_2(\boldsymbol{\omega}_2^{(t)})\| > tol)$ *and* $t > T$ **do**

Compute $\nabla F_1(\boldsymbol{\omega}_{(1,t)})$ and $\nabla F_2(\boldsymbol{\omega}_{(2,t)})$ using Eqs. (4.2.8) and (4.2.9), respectively.

Compute $\mathbf{m}_{(1,t+1)}$ and $\mathbf{m}_{(2,t+1)}$ using Eqs. (4.2.10) and (4.2.15), respectively.

Compute $\mathbf{v}_{(1,t+1)}$ and $\mathbf{v}_{(2,t+1)}$ using Eqs. (4.2.11) and (4.2.16), respectively.

Compute $\hat{\mathbf{m}}_{(1,t+1)}$ and $\hat{\mathbf{m}}_{(2,t+1)}$ using Eqs. (4.2.12) and (4.2.17), respectively.

Compute $\hat{\mathbf{v}}_{(1,t+1)}$ and $\hat{\mathbf{v}}_{(2,t+1)}$ using Eqs. (4.2.13) and (4.2.18), respectively.

Update $\boldsymbol{\omega}_{(1,t+1)}$ and $\boldsymbol{\omega}_{(2,t+1)}$ using Eqs. (4.2.14) and (4.2.19), respectively.

$t = t + 1$

end

Return: $\boldsymbol{\omega}_{(1,t)}$ and $\boldsymbol{\omega}_{(2,t)}$

```
class adamTBSVMwithLinex(BaseEstimator):

    def __init__(self, C_1 = 1, C_3 = 1, a = 1,
                 B1 = 0.9, B2 = 0.999, epsilon = 0.000001,
                 alpha = 0.001,m = 64, max_iter = 1000,
```

```
                tol = 0.0001):
        assert C_1 > 0
        self.C_1 = C_1
        assert C_3 > 0
        self.C_3 = C_3
        assert B1 > 0
        self.B1 = B1
        assert B2 > 0
        self.B2 = B2
        self.epsilon = epsilon
        self.alpha = alpha
        self.m = m
        self.max_iter = max_iter
        self.tol = tol

    def pos_gradient_term2(self, w, x):
        gradient2__ = []
        for i in range(len(x)):
          wTxx = 2 * np.dot(w,x[i,:]) * x[i,:]
          #print(wTxx)
          gradient2__.append(wTxx)

        gradient2_ = np.vstack(gradient2__)
        gradient2 = np.mean(gradient2_, axis = 0)

        return gradient2

    def pos_gradient_term3(self, w, x):
        gradient3__ = []
        for i in range(len(x)):
          c  = 1 + np.dot(w,x[i,:])
          e_ = math.exp(self.a*c) *
                (self.a*x[1,:]) - (self.a*x[1,:])
          gradient3__.append(e_)

        gradient3_ = np.vstack(gradient3__)
        gradient3 = np.mean(gradient3_, axis = 0)

        return gradient3

```

```
46      def neg_gradient_term2(self, w, x):
47          gradient2__ = []
48          for i in range(len(x)):
49            wTxx = 2 * np.dot(w,x[i,:]) * x[i,:]
50            #print(wTxx)
51            gradient2__.append(wTxx)
52
53          gradient2_ = np.vstack(gradient2__)
54          gradient2 = np.mean(gradient2_, axis = 0)
55
56          return gradient2
57
58      def neg_gradient_term3(self, w, x):
59          gradient3__ = []
60          for i in range(len(x)):
61            c  = 1 - np.dot(w,x[i,:])
62            e_ = math.exp(self.a*c) * -(self.a*x[1,:])
63                 +(self.a*x[1,:])
64            gradient3__.append(e_)
65
66          gradient3_ = np.vstack(gradient3__)
67          gradient3 = np.mean(gradient3_, axis = 0)
68
69          return gradient3
70
71      def fit(self, x, t):
72          self.classes_ = np.unique(t)
73          t[t==0] = -1
74          k_pos = 1
75          k_neg = 1
76          update_pos = True
77          update_neg = True
78          w_pos = np.zeros(len(x[0])+1)
79          w_neg = np.zeros(len(x[0])+1)
80
81          m_adam = np.zeros(len(x[0])+1)
82          v_adam = np.zeros(len(x[0])+1)
83
84          X = np.ones((len(x),len(x[0])+1))
85          X[:,:-1] = x
```

```
86          X_pos = X[t==1]
87          X_neg = X[t==-1]
88
89          #solve the positive class problem
90          while k_pos <= (self.max_iter) and update_pos:
91
92            #stochastic random a data
93            r_pos = np.random.randint(len(X_pos), size = self.m)
94            r_neg = np.random.randint(len(X_neg), size = self.m)
95            X_pos_ = X_pos[r_pos,:]
96            X_neg_ = X_neg[r_neg,:]
97
98            #compute gradient
99            g_2 = self.pos_gradient_term2(w_pos, X_pos_)
100           g_3 = self.pos_gradient_term3(w_pos, X_neg_)
101           gradient_pos = self.C_3 * w_pos + g_2 +
102                          (self.C_1/2)*g_3
103
104           #check stopping criterion
105           if np.linalg.norm(gradient_pos) <= self.tol:
106               update_pos = False
107
108           #update step
109           else:
110               #step adam algorithm
111               m_adam = self.B1 * m_adam + (1-self.B1) *
112                        gradient_pos
113               v_adam = self.B2 * v_adam + (1-self.B2) *
114                        (gradient_pos**2)
115
116               m_adam_ = m_adam / (1-(self.B1**k_pos))
117               v_adam_ = v_adam / (1-(self.B2**k_pos))
118
119               w_pos = w_pos - self.alpha * m_adam_ /
120                       (geek.sqrt(v_adam_) + self.epsilon)
121
122               k_pos += 1
123
124         #solve the negative class problem
125         while k_neg <= (self.max_iter) and update_neg:
```

```
126
127         #stochastic random a data
128         r_pos = np.random.randint(len(X_pos), size = self.m)
129         r_neg = np.random.randint(len(X_neg), size = self.m)
130         X_pos_ = X_pos[r_pos,:]
131         X_neg_ = X_neg[r_neg,:]
132
133         #compute gradient
134         g_2 = self.neg_gradient_term2(w_neg, X_neg_)
135         g_3 = self.neg_gradient_term3(w_neg, X_pos_)
136         gradient_neg = self.C_3 * w_neg + g_2 +
137                        (self.C_1/2)*g_3
138
139         #check stopping criterion
140         if np.linalg.norm(gradient_neg) <= self.tol:
141             update_neg = False
142
143         #update step
144         else:
145             #step adam algorithm
146             m_adam = self.B1 * m_adam + (1-self.B1) *
147                      gradient_neg
148             v_adam = self.B2 * v_adam + (1-self.B2) *
149                      (gradient_neg**2)
150             m_adam_ = m_adam / (1-(self.B1**k_neg))
151             v_adam_ = v_adam / (1-(self.B2**k_neg))
152             w_neg = w_neg - self.alpha * m_adam_ /
153                     (geek.sqrt(v_adam_) + self.epsilon)
154
155             k_neg += 1
156
157
158     self.final_iteration_pos = k_pos
159     self.omega_pos = w_pos
160     self._coef_pos = w_pos[:-1]
161     self._intercept_pos = w_pos[-1]
162     self.final_iteration_neg = k_neg
163     self.omega_neg = w_neg
164     self._coef_neg = w_neg[:-1]
165     self._intercept_neg = w_neg[-1]
```

```
166
167         return self
168
169     def predict(self, x):
170         pos_dis = abs(np.matmul(x ,self._coef_pos)
171                       + self._intercept_pos )
172         neg_dis = abs(np.matmul(x ,self._coef_neg)
173                       + self._intercept_neg )
174         p = np.sign(abs(neg_dis) - abs(pos_dis))
175         p[p==0] = 1
176         return p
```

In the following, we delve into the convexity analysis of the proposed LSTBSVM-linex model. The incorporation of the Linex loss function into the LSTBSVM engenders a combination of convex functions, implying that LSTBSVM-linex represents convex optimization problems. This implication, as supported by [8], suggests that the estimated convergence speed of the LSTBSVM-linex, when solved using Adam, is approximately $O(\frac{1}{\sqrt{T}})$. This confirmation further reinforces the expected convergence behavior under the Adam optimization method. Regarding the algorithm's hyperparameters, Adam involves five variables: $\alpha, \beta_1, \beta_2, \lambda$ and ϵ. We assign specific values to these hyperparameters to ensure consistency and optimize experimentation: $\alpha = 1, \beta_1 = 0.9, \beta_2 = 0.999, \lambda = 0.5$, and $\epsilon = 0.0001$.

4.2.2 Numerical experiments

This section comprehensively evaluates the proposed approach's classification performance in relation to other relevant methods. The evaluations are conducted on synthetic datasets, the UCI machine learning repository, and knee osteoarthritis X-ray image classification. We employ a 5-fold cross-validation technique to ensure reliable results across all experiments. The tables display the average accuracy and standard deviation for each experiment, with emphasis on highlighting the best-performing approach.

Table 13 Hyperparameters tuned.

Algorithm	Parameter	Range
TBSVM	c_1	10^{-3} to 10^3
	c_3	10^{-3} to 10^3
SmoLSTBSVM	c_1	10^{-3} to 10^3
	c_3	10^{-3} to 10^3
LSTBSVM-linex	c_1	10^{-3} to 10^3
	c_3	10^{-3} to 10^3
	a	-3,-2,-1,1,2,3

Throughout these experiments, we focused on comparing the performance of our proposed LSTBSVM-linex method with TBSVM [41] and SmoLSTBSVM [24]. Our evaluation process employed the grid search method over the hyperparameters within the ranges specified in Table 13. It is important to note that we maintained consistency in hyperparameters for all models by setting $c_1 = c_2$ and $c_3 = c_4$. The dataset details are presented in Table 14.

Table 14 Characteristics of UCI datasets.

	Dataset	Size	Attribute
Small-size	Banknote	1372	4
	Coli2000	9822	85
	Heart	270	13
	Ionosphere	351	33
	Pima	768	8
	Phoneme	5404	5
	Sonar	208	60
	Titanic	2201	3
	Twonorm	7400	20
	Wdbc	569	30

To quantitatively measure the performance of algorithms, we computed several key metrics:

1. The Accuracy assesses the ratio of correctly predicted instances to the total instances in the dataset, giving an overall measure of how often the model's predictions are correct.

2. The F1-score represents the harmonic mean of precision and recall, providing a balanced measure of model accuracy, particularly useful when dealing with imbalanced classes.

3. The Matthews Correlation Coefficient (MCC) measures the quality of binary classifications, considering true and false positives and negatives. It remains robust even when handling imbalanced datasets, providing an insightful evaluation metric for classification models.

The corresponding formulas for these metrics are represented in Table 15, while all experimental results are showcased in Tables 16 to 21.

Table 15 Performance measure.

Performance measure	Formula used
Accuracy	$\frac{T.N.+T.P.}{T.N.+F.T.+T.P.+F.P.}$
F1-score	$\frac{2T.P.}{2T.P.+F.P.+F.N.}$
Matthews correlation coefficient (MCC)	$\frac{T.P.\times T.N.-F.P.\times F.N.}{\sqrt{(T.P.+F.P.)(T.P.+F.N.)(T.N.+F.P.)(T.N.+F.N.)}}$

Our extensive experimentation delivered compelling results, underscoring our model's effectiveness across various metrics. Consistently ranking at the bottom of each table, our model surpassed all others in Accuracy, F1 score, and Matthews Correlation Coefficient (MCC) measurements. The notably high MCC values were particularly noteworthy, highlighting our model's robust adaptability in handling imbalanced datasets. A key contributing factor to our model's superiority is the integration of the Linex loss function, setting it apart from other models.

We conducted additional tests on ten binary classification datasets to assess our model's performance in handling feature noise. These experiments involved gradually introducing noise levels ranging from $r = 0.5$ to $r = 0.1$, where noise signifies the mean of feature noise. Tables 17, 18, 20, and 21 present the outcomes for LSTBSVM-line and related models in linear and nonlinear cases, respectively. Notably, our method consistently exhibited superior performance. Despite the Linex loss lacking a theoretical guarantee for optimal performance in noisy environments, our experiments demonstrated its remarkable efficacy in such scenarios. These findings confirm our initial hypothesis, showcasing our model's exceptional performance even in datasets affected by feature noise.

Furthermore, the meticulous assessment conducted via the Friedman test significantly bolsters this robust statistical validation. It encompasses the ranking analysis of accuracy (ACC), F1 score, and Matthews Correlation Coefficient (MCC)

Table 16 Classification result for the testing set on UCI datasets attained by the linear kernel.

Dataset	Performance measure	Algorithms		
		LSTBSVM-linex	**SmoLSTBSVM**	**TBSVM**
Banknote	Accuracy	**98.25±0.42**	97.89±0.63	96.65±1.01
	F1-score	**0.9825**	0.9788	0.9666
	MCC	**0.9652**	0.9578	0.9349
Coli2000	Accuracy	**93.99±0.35**	92.99±0.76	93.81±0.32
	F1-score	**0.9112**	0.9084	0.9111
	MCC	**0.3300**	0.2510	0.1750
Ionosphere	Accuracy	88.32±1.03	**89.75± 1.61**	**89.75±1.61**
	F1-score	0.8882	**0.8974**	0.8957
	MCC	0.7632	0.7733	**0.7776**
Haberman	Accuracy	73.84±5.23	73.53±3.36	**73.86±3.74**
	F1-score	**0.7141**	0.6650	0.6642
	MCC	**0.2419**	0.1389	0.1429
Phoneme	Accuracy	**75.22±1.49**	70.65±0.53	72.85±0.35
	F1-score	**0.7598**	0.5850	0.6872
	MCC	**0.4697**	0.0000	0.2445
Sonar	Accuracy	74.07±4.48	76.49±5.36	**77.96±6.47**
	F1-score	0.7337	0.7646	**0.7804**
	MCC	0.4614	0.5207	**0.5539**
Australian	Accuracy	**85.94±3.51**	84.35±2.36	85.51±3.83
	F1-score	**0.8596**	0.8423	0.8552
	MCC	**0.7276**	0.6867	0.7199
Twonorm	Accuracy	**97.81±0.27**	93.39±2.19	93.99±2.77
	F1-score	**0.9772**	0.9339	0.9395
	MCC	**0.9544**	0.8683	0.8863
Appendicitis	Accuracy	81.04±8.07	**85.80±8.57**	**85.80±3.13**
	F1-score	0.7609	0.8373	**0.8634**
	MCC	0.2561	0.4740	**0.5848**
Wdbc	Accuracy	**96.13±0.89**	95.78±1.02	95.61±1.57
	F1-score	**0.9594**	0.9575	0.9555
	MCC	**0.9149**	0.9102	0.9075

Table 17 Classification result for the testing set on UCI datasets attained by the linear kernel with 0.05% feature noise.

Dataset	Performance measure	Algorithm		
		LSTBSVM-linex	SmoLSTBSVM	TBSVM
Banknote	Accuracy	98.40±0.44	**98.69±0.49**	96.58±1.04
	F1-score	0.9840	**0.9869**	0.9659
	MCC	0.9680	**0.9734**	0.9336
Coli2000	Accuracy	**93.99±0.35**	92.99±0.76	93.81±0.32
	F1-score	**0.9112**	0.9084	0.9111
	MCC	**0.3300**	0.2510	0.1750
Ionosphere	Accuracy	88.90±2.23	**90.04.90±2.67**	90.04±2.67
	F1-score	0.8847	**0.8999**	0.8833
	MCC	0.7586	**0.7800**	0.7510
Haberman	Accuracy	**74.83±5.93**	73.19± 4.04	73.19±4.04
	F1-score	**0.7298**	0.6589	0.6629
	MCC	**0.2795**	0.1042	0.1148
Phoneme	Accuracy	**75.06±1.29**	70.65±0.53	72.95±0.79
	F1-score	**0.7592**	0.5850	0.6885
	MCC	**0.4716**	0.0000	0.2476
Sonar	Accuracy	74.08±7.84	76.485±5.20	**77.46±5.17**
	F1-score	0.7378	0.7637	**0.7754**
	MCC	0.4779	0.5198	**0.5430**
Australian	Accuracy	**85.80±3.45**	83.77±6.09	85.51±3.81
	F1-score	**0.8581**	0.8383	0.8551
	MCC	0.4614	0.5207	**0.5539**
Twonorm	Accuracy	**97.66±0.32**	91.59±2.30	92.00±8.70
	F1-score	**0.9766**	0.9159	0.9163
	MCC	**0.9533**	0.8332	0.8567
Appendicitis	Accuracy	**85.80±3.13**	85.80±8.51	81.99±7.22
	F1-score	**0.8634**	0.8373	0.7661
	MCC	**0.5848**	0.4740	0.2644
Wdbc	Accuracy	**96.31±0.28**	95.78±0.85	95.26±2.04
	F1-score	**0.9627**	0.9574	0.9521
	MCC	**0.9219**	0.9102	0.8987

Table 18 Classification result for the testing set on UCI datasets attained by the linear kernel with 0.1% feature noise.

Dataset	Performance measure	Algorithms		
		LSTBSVM-linex	**SmoLSTBSVM**	**TBSVM**
Banknote	Accuracy	**98.47±0.27**	97.52±1.01	97.30±0.68
	F1-score	**0.9854**	0.9832	0.9731
	MCC	**0.9652**	0.9578	0.9349
Coli2000	Accuracy	**93.99±0.35**	92.99±0.76	93.81±0.32
	F1-score	**0.9112**	0.9084	0.9111
	MCC	**0.3300**	0.2510	0.1750
Ionosphere	Accuracy	**88.61±1.24**	87.75±3.05	84.62±3.96
	F1-score	**0.8820**	0.8711	0.8331
	MCC	**0.7520**	0.7308	0.6715
Haberman	Accuracy	**76.41±5.56**	73.19± 4.27	73.84±4.82
	F1-score	0.7411	**0.6561**	0.6638
	MCC	0.3081	0.1035	**0.1319**
Phoneme	Accuracy	**75.37±1.43**	70.65±0.53	72.76±0.68
	F1-score	**0.7622**	0.5850	0.6857
	MCC	**0.4725**	0.0000	2401
Sonar	Accuracy	75.53±7.22	76.47±5.47	**77.47±6.34**
	F1-score	0.7525	0.7639	**0.7753**
	MCC	0.4947	0.5219	**0.5451**
Australian	Accuracy	**86.09±4.11**	84.06±3.67	85.65±3.71
	F1-score	**0.8609**	0.8407	0.8566
	MCC	**0.7308**	0.6802	0.7232
Twonorm	Accuracy	97.51±0.23	89.42±3.55	**90.92±3.78**
	F1-score	0.9751	0.8941	**0.9080**
	MCC	0.9503	0.7901	**0.8330**
Appendicitis	Accuracy	85.80±3.15	**86.75±8.23**	81.04±9.13
	F1-score	**0.8634**	0.8462	0.7463
	MCC	**0.5848**	0.5005	0.2011
Wdbc	Accuracy	**95.78±1.02**	**95.78±0.65**	95.61±1.75
	F1-score	0.9574	**0.9575**	0.9558
	MCC	**0.9111**	0.9103	0.9073

Table 19 Classification result for the testing set on UCI datasets attained by RBF kernel.

Dataset	Performance measure	Algorithms		
		LSTBSVM-linex	**SmoLSTBSVM**	**TBSVM**
Banknote	Accuracy	98.25±0.53	**99.71±0.27**	97.3-±0.68
	F1-score	0.9847	**0.9978**	0.9731
	MCC	0.9696	**0.9956**	0.9472
Coli2000	Accuracy	68.53±0.35	63.75±0.73	**94.03±0.31**
	F1-score	0.6883	0.6375	0.9108
	MCC	0.1112	0.0205	0.2001
Ionosphere	Accuracy	**89.18± 1.41**	88.90± 3.51	87.76±3.39
	F1-score	**0.8869**	0.8831	0.8712
	MCC	0.7559	**0.7598**	0.7346
Haberman	Accuracy	73.53±4.54	73.53±4.42	**73.87±3.94**
	F1-score	**0.7345**	0.6930	0.6696
	MCC	**0.3275**	0.1882	0.1479
Phoneme	Accuracy	71.43±1.59	70.65±0.91	75.41±0.28
	F1-score	0.7260	0.5850	0.7404
	MCC	0.4119	0.0015	0.3624
Sonar	Accuracy	**80.79±1.95**	78.40±2.81	77.48±7.50
	F1-score	**0.8080**	0.7836	0.7753
	MCC	**0.6119**	0.5564	0.5416
Australian	Accuracy	**53.77±5.37**	50.87±2.31	53.19±2.54
	F1-score	**0.5354**	0.5075	0.5270
	MCC	**0.0684**	0.0023	0.0356
Twonorm	Accuracy	**97.64±0.25**	96.77±0.36	97.46±0.49
	F1-score	**0.9741**	0.9677	0.9746
	MCC	0.9481	0.9354	**0.9492**
Appendicitis	Accuracy	**84.85±5.64**	83.94±3.88	80.17±5.59
	F1-score	**0.8474**	0.8244	0.7849
	MCC	**0.5350**	0.4021	0.2734
Wdbc	Accuracy	90.68±2.40	91.56±2.64	**92.26±4.26**
	F1-score	0.9070	0.9156	**0.9212**
	MCC	0.8044	0.8220	**0.8371**

Table 20 Classification result for the testing set on UCI datasets attained by RBF kernel with 0.05% feature noise.

Dataset	Performance measure	Algorithms		
		LSTBSVM-linex	**SmoLSTBSVM**	**TBSVM**
Banknote	Accuracy	**98.54±0.46**	98.32±1.43	97.30±0.08
	F1-score	**0.9854**	0.9832	0.9731
	MCC	**0.9710**	0.9661	0.9472
Coli2000	Accuracy	67.13±1.01	62.15±0.03	**91.03±0.15**
	F1-score	0.6110	0.6345	0.8908
	MCC	0.1012	0.0201	0.1901
Ionosphere	Accuracy	**89.18± 1.08**	88.90± 2.41	86.91±4.23
	F1-score	**0.8873**	0.8832	0.8612
	MCC	**0.7649**	0.7592	0.7229
Haberman	Accuracy	73.20±3.35	72.88±4.89	**73.54±3.87**
	F1-score	**0.7276**	0.6754	0.6637
	MCC	**0.2998**	0.1516	0.1315
Phoneme	Accuracy	72.39±3.01	70.65±0.91	75.46±0.35
	F1-score	0.7334	0.5850	0.7410
	MCC	0.4112	0.0000	0.3641
Sonar	Accuracy	**77.41±1.85**	74.10±5.91	75.52±7.49
	F1-score	**0.7739**	0.7415	0.7558
	MCC	**0.5387**	0.4808	0.5072
Australian	Accuracy	51.45±3.01	51.30±0.85	**52.90±2.55**
	F1-score	0.5155	0.5114	**0.5241**
	MCC	0.0214	0.0076	**0.0300**
Twonorm	Accuracy	**97.62±0.23**	96.35±0.29	97.57±0.41
	F1-score	**0.9762**	0.9635	0.9757
	MCC	**0.9525**	0.9271	0.9514
Appendicitis	Accuracy	**83.03±7.70**	82.03±4.78	82.94±4.99
	F1-score	**0.8299**	0.8170	0.8258
	MCC	**0.4389**	0.3864	0.4080
Wdbc	Accuracy	90.86±3.07	89.63±0.85	**92.08±4.06**
	F1-score	0.9089	0.8964	**0.9194**
	MCC	0.8075	0.7802	**0.8331**

Table 21 Classification result for the testing set on UCI datasets attained by RBF kernel with 0.1% feature noise.

Dataset	Performance measure	Algorithms		
		LSTBSVM-linex	**SmoLSTBSVM**	**TBSVM**
Banknote	Accuracy	**98.54±0.69**	99.85±0.18	97.23±0.75
	F1-score	**0.9854**	0.9985	0.9724
	MCC	**0.9711**	0.9971	0.9458
Coli2000	Accuracy	64.91±1.14	60.12±0.51	**90.57±0.22**
	F1-score	0.5110	0.5050	0.8997
	MCC	0.1009	0.0193	0.1101
Ionosphere	Accuracy	**88.04± 1.89**	84.89± 2.52	86.61±4.58
	F1-score	**0.8750**	0.8466	0.8562
	MCC	**0.7401**	0.6706	0.7206
Haberman	Accuracy	71.58±3.90	72.89±4.96	**74.19±4.12**
	F1-score	**0.7138**	0.6828	0.6721
	MCC	**0.2998**	0.1516	0.1315
Phoneme	Accuracy	72.35±2.06	70.65±0.91	**75.00±0.31**
	F1-score	0.7331	0.5850	**0.7357**
	MCC	**0.4034**	0.0000	0.3506
Sonar	Accuracy	**70.20±1.82**	69.71±1.17	63.90±7.91
	F1-score	**0.7020**	0.6972	0.6375
	MCC	**0.5989**	0.4002	0.2861
Australian	Accuracy	51.74±3.16	51.16±1.98	**52.90±2.55**
	F1-score	0.5184	0.5110	**0.5241**
	MCC	0.0215	0.0079	**0.0298**
Twonorm	Accuracy	**97.43±0.29**	96.64±0.52	97.32±0.39
	F1-score	**0.9743**	0.9663	0.9732
	MCC	**0.9487**	0.9328	0.9465
Appendicitis	Accuracy	**83.07±6.25**	82.12±0.23	80.26±4.19
	F1-score	**0.8192**	0.8206	0.8006
	MCC	**0.3854**	0.4185	0.3754
Wdbc	Accuracy	91.74±1.54	89.80±2.06	**91.91±4.27**
	F1-score	0.9176	0.8983	**0.9177**
	MCC	0.8250	0.7846	**0.8287**

across 30 diverse datasets, employing six distinct models: LSTBSVM-linex, SmoLSTBSVM, TBSVM, LSTBSVM-linex(b), SmoLSTBSVM(b), and TBSVM(b). Detailed findings are systematically presented in Tables 16 to 21. Throughout this evaluation, the Friedman test consistently attributes the lowest rank value to the algorithm exhibiting the highest performance within this comprehensive assessment framework. The rankings illustrated in Figure 26 corroborate these findings, showcasing that the proposed method consistently outperforms others across various metrics. These results solidify the conclusion that the proposed method emerges as the top-performing algorithm, as indicated by its consistently superior rank across accuracy (ACC), F1 score, and Matthews Correlation Coefficient (MCC).

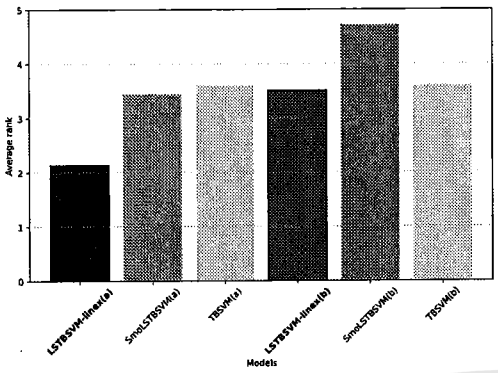
Additionally, in scenarios where all methods exhibit equivalence, the Friedman statistic assumes a pivotal role, derived by

$$x_F^2 = \frac{12m}{k(k+1)}\left[\sum_j \left(\frac{1}{m}\sum_i r_i^j\right)^2 - \frac{k(k+1)^2}{4}\right], \tag{4.2.20}$$

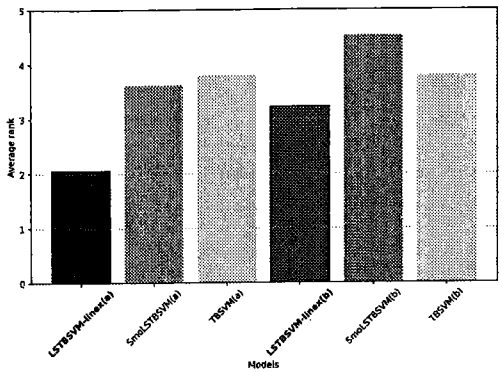
where r_i^j represents the j-th of k methods on the i-th of m datasets. Friedman's x_F^2 is undesirably conservative and comes up with a better statistic

$$F_F = \frac{(m-1)x_F^2}{m(k-1)-x_F^2}, \tag{4.2.21}$$

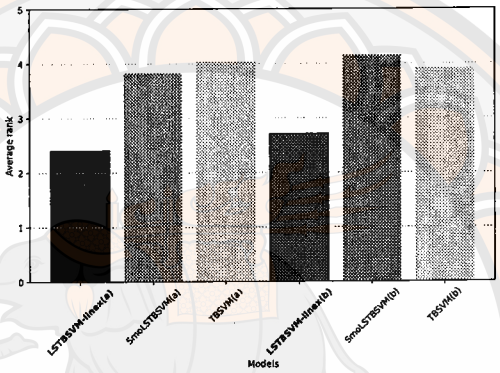
which is distributed according to the F-distribution with $k-1$ and $(k-1)(m-1)$ degrees of freedom. This statistical metric, derived specifically in cases where all methods demonstrate equivalency, allows for nuanced differentiation among algorithms despite their similar performances. Table 22 illustrates the Friedman statistics, where F_F is distributed with degrees of freedom $(k-1) = 6-1 = 5$ and $(k-1)(m-1) = (6-1)(30-1) = 145$. The values of F_F in Table 22 indicate the rejection of the null hypothesis for all performance metrics, utilizing the F distribution table with $F(5, 145)$ at $\alpha = 0.05$ significance level. This implies that there are significant differences among the six algorithms.



(a) Average accuracy ranking.



(b) Average F1-score ranking.



(c) Average MCC ranking.

Figure 26 The Average rank of the six models was calculated based on their performance metrics, indicating their relative standings in terms of accuracy, F1 score, and MCC.

Table 22 Friedman test.

Performance Metric	x_F^2	F_F	Critical value
Accuracy	28.89	6.92	
F1-score	29.23	7.02	2.27
Mcc	24.15	5.56	

In this sub-section, we aim to assess the performance of LSTBSVM-linex on large-scale datasets by comparing it with SVM and SmoLSTBSVM. The compar-

ison involves four large-scale UCI datasets, outlined in Table 23, to evaluate the efficacy of these models across varying data scales. Notably, the parameters employed for our model remain consistent with those from the previous section. In contrast, the SVM model is fine-tuned within the range of 10^{-3} to 10^{3}, adding granularity to our evaluation. It is important to note that the computation time is the sum of the training and testing time, averaged over 5 times 5-fold cross-validation. The results of this comparative analysis are presented in Table 24.

In light of the experimental findings, it is evident that our proposed model demonstrates superiority over others, particularly in terms of computational time efficiency. When examining timing, our model notably outperformed the SVM model. For instance, in the Miniboone dataset, although our proposed method achieved an F1-score of 0.8265, slightly lower than the SVM's 0.066, its computational time was significantly lower at only 1.4995 seconds. Thus, it is essential to highlight that despite our model's outstanding speed, it maintains accuracy levels performance, outperforming two out of four datasets, with the Skin and Magic datasets being a noteworthy example. The dual strengths of our model are its efficiency and reliability, making it an appealing option for real-world applications that demand both speed and accuracy.

Table 23 Characteristics of large-scale datasets.

	Dataset	**Size**	**Attribute**
Large-size	Magic	19020	10
	Sepsis	11041	3
	Miniboone	130064	50
	Skin	245057	3

Table 24 Classification result for large-scale datasets.

Dataset		Algorithms		
		LSTBSVM-linex	**SmoLSTBSVM**	**SVM**
Magic	Accuracy	**78.88**	76.00	78.52
	F1-score	**0.7796**	0.7387	0.7786
	MCC	**0.5074**	0.4367	0.5061
	Time	**1.0079**	2273.7902	112.1515
Sepsis	Accuracy	**92.63**	*	92.58
	F1-score	0.8900	*	**0.8901**
	MCC	**0.0000**	*	-0.3333
	Time	**2.0907**	*	37.7943
Miniboone	Accuracy	82.47	*	**89.39**
	F1-score	0.8265	*	**0.8925**
	MCC	0.5806	*	**0.7347**
	Time	**1.4995**	*	391.3255
Skin	Accuracy	**96.99**	*	92.84
	F1-score	**0.9698**	*	0.9310
	MCC	**0.9088**	*	0.8083
	Time	**2.0860**	*	517.2883

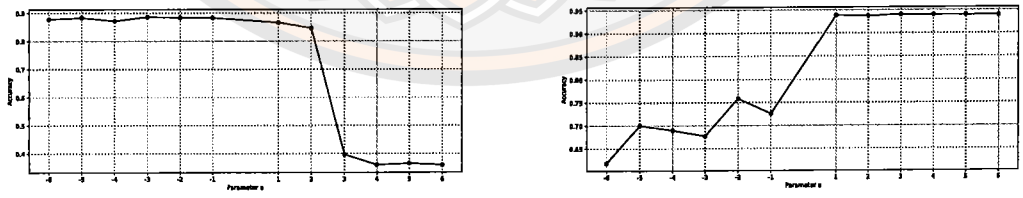
To comprehend the impact of specific parameters, we initiated a study focusing on critical variables within LSTBSVM-linex. This model incorporates two crucial parameters: a, governing the steepness of the Linex loss function curve, and m, determining the mini-batch size in the SGD algorithm. For simplicity, we chose two small-sized UCI datasets, German and Pima, as our subjects for examining these parameters. For simplicity, we decided on two small-sized UCI datasets,

Ionosphere and Coil2000, as the subjects for studying these parameters.

In our initial study on parameter a, we are presented in Figure 27. For the Ionosphere dataset, maintaining negative values of a in consistently high accuracy, while positive leads to a substantial drop in accuracy. However, in the Coil2000 dataset, negative a values gradually decrease accuracy, while positive a values consistently achieve high accuracy levels. These findings align with established theories, validating our hypothesis that an influences penalty weight: negative a values imply heavier penalties between the two centre hyperplanes, while positive a values reverse this effect. Consequently, in LSTBSVM-linex, the value of parameter a is not restricted to positive or negative values; we can set a parameter range and utilize the Grid search method to pinpoint optimal values.

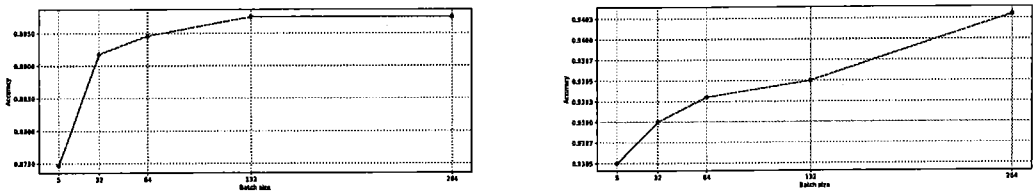
Furthermore, we experimented on the mini-batch size, with results depicted in Figure 28. These results demonstrate how accuracy fluctuates concerning batch size. It's evident that an increase in the mini-batch size initially corresponds to an increase in accuracy, which eventually stabilizes.

Moreover, we analyzed the relationship between running time and batch size, as displayed in Figure 29. Notably, the running time demonstrates an approximately linear increase with batch size. Consequently, combining insights from Figs.28 and 29 suggest that scenarios prioritizing accuracy benefit from a relatively medium batch size (not necessarily large), while scenarios emphasizing efficiency favour a relatively smaller batch size.



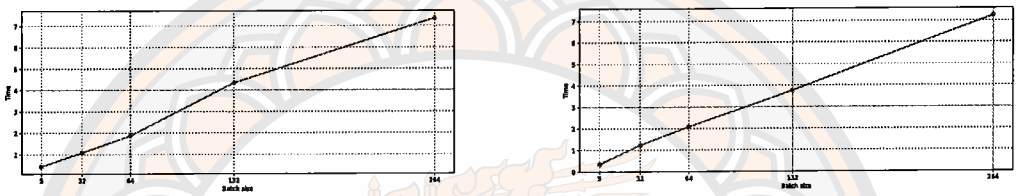
(a) Ionosphere dataset **(b)** Coil2000 dataset

Figure 27 Parameter study on the parameter a.



(a) Ionosphere dataset (b) Coil2000 dataset

Figure 28 Parameter study on the size of mini-batch.



(a) Ionosphere dataset (b) Coil2000 dataset

Figure 29 Parameter study on the relation between the mini-batch size and running time.

In the following, we will explore the application of the proposed LSTBSVM-linex method in image recognition tasks. The experiments used the widely recognized Knee Osteoarthritis dataset sourced from Kaggle [42]. This dataset comprises 3836 X-ray images divided into two classes: Osteoarthritis and normal, as illustrated in Figure 30. A meticulous preprocessing pipeline was implemented to prepare the dataset for analysis to ensure standardization and quality of the images before model application.

(a) Normal (b) Osteoarthritis

Figure 30 Representative X-ray images of normal and osteoarthritis.

In the stage of image preparation, we implemented image enhancement techniques as utilized in [42] to improve the clarity and contrast of X-ray images. Subsequently, we employed ResizedCrop to standardize dimensions and conducted image normalization to ensure uniformity in pixel intensities and dimensions across the dataset. Following this, feature extraction was performed using the pre-trained VGG16 model, generating a comprehensive set of 512 deep features per image crucial for subsequent analysis. Instead of employing traditional feature extraction methods standard in machine learning-based studies, this paper leverages the capabilities of the pre-trained VGG16 model. These extracted features served as the primary input for a SVM classifier, facilitating efficient classification between osteoarthritis and normal conditions. Subsequently, we conducted post-classification analysis and result interpretation to evaluate the model's efficacy in accurately discerning the two conditions within the X-ray images.

The upcoming code segment will demonstrate a method that utilizes the OpenCV library to enhance and sharpen images. This technique involves a two-step process: first, it applies contrast-limited Adaptive Histogram Equalization (CLAHE) to enhance image contrast, and then it performs unsharp masking to enhance edges and make the image appear more defined.

```
import cv2
import numpy as np

# Function to apply CLAHE and unsharp masking to an image
def enhance_and_sharpen_image(input_image):
    # Create a CLAHE object with specified parameters
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

    # Apply CLAHE to the image
    equalized_image = clahe.apply(input_image)

    # Calculate the 1st and 99th percentiles of the original image
    percentile1 = np.percentile(input_image, 1)
    percentile99 = np.percentile(input_image, 99)

    # Perform contrast stretching
```

```
    equalized_image = np.where(equalized_image < percentile1,
                               0, equalized_image)
    equalized_image = np.where(equalized_image > percentile99,
                               255, equalized_image)

    # Unsharp masking parameters
    radius = 2
    amount = 1.5  # Adjust as needed

    # Apply Gaussian blur to create a blurred image
    blurred_image = cv2.GaussianBlur(equalized_image, (0, 0),
                                     radius)

    # Calculate the sharpened image by subtracting the blurred
    image from the original
    sharpened_image = cv2.addWeighted(equalized_image, 1 +
                                      amount, blurred_image,
                                      - amount, 0)

    return sharpened_image
```

In the code, we create a CLAHE object with specific parameters, apply this object to the input image to improve contrast, and execute unsharp masking to finalize the image enhancement. These steps aim to significantly improve the visual quality of images, making it particularly useful for areas requiring detailed image analysis. Figure 31 will display the results obtained from the enhance and sharpen image process.

To evaluate the performance of the proposed LSTBSVM-linex classifier, we conducted a comparative analysis against the TBSVM and SmoLSTBSVM methods. The experimental results, shown in Table 25, were derived using the same preprocessing techniques. The findings indicate that the LSTBSVM-linex method outperforms the others in accuracy, F1 score, and MCC. This highlights the effectiveness of combining the LSTBSVM-linex approach with features extracted from the pre-trained VGG16 model, significantly improving the model's ability to differentiate between conditions in X-ray images.

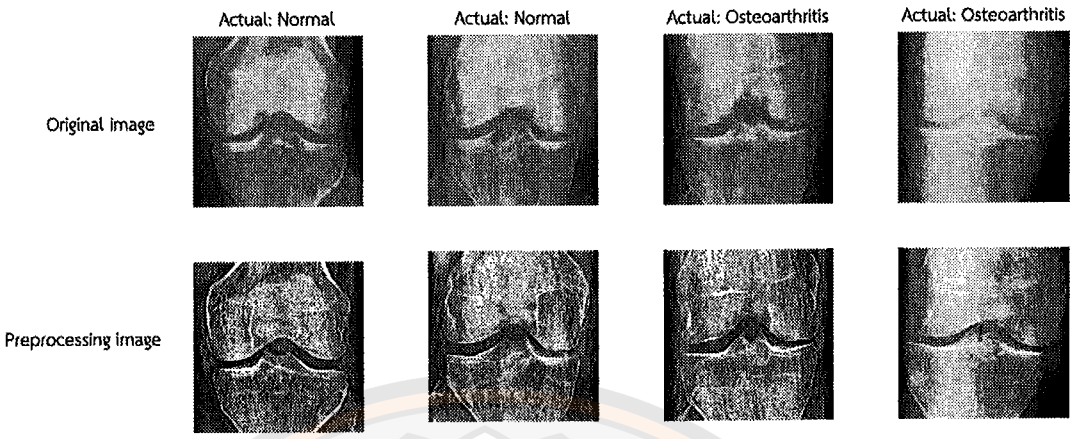


Figure 31 Comparison of the original image with the Preprocessing image.

Table 25 Performance comparisons on knee osteoarthritis X-ray dataset.

Algorithm	Accuracy	F1-score	MCC
TBSVM	75.65	0.7557	0.4950
SmoLSTBSVM	60.94	0.5873	0.1561
LSTBSVM-linex	**76.44**	**0.7644**	**0.5030**

CHAPTER VI

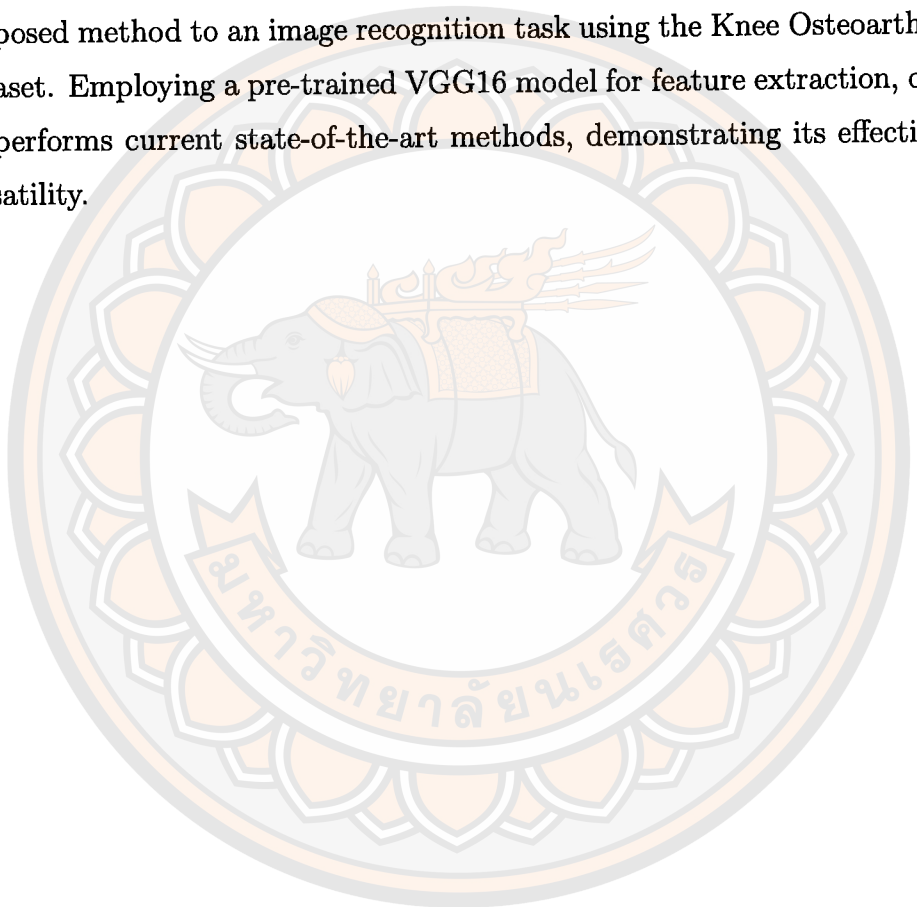
CONCLUSION

In this chapter, we conclude all results again to see an overview of this thesis. In part to improve the efficiency of the optimization method, we propose a novel regularized stochastic Nesterov's accelerated BFGS method (RES-NAQ). It was developed to solve strongly convex optimization problems. For stable and faster convergence, a new momentum coefficient was proposed and applied to our proposed RES-NAQ. Consequently, we have proven that a generated iteration subsequence almost surely converges to an optimal solution and have shown that the rate of convergence of the RES-NAQ is $\mathcal{O}(\frac{1}{t})$. Furthermore, the effectiveness of the proposed method is evaluated via a support vector machine problem. The experimental results show that our method is more efficient at accelerating convergence speed and reducing oscillations. Finally, the experimental results have shown that the performance of our method is better than the existing method in terms of classification accuracy (ACC).

Next, in part of the SVM-based model, we constructed a smooth approximation of the generalized pinball loss function. As a result, a new smooth support vector machine with generalized pinball loss is obtained. Theoretically, we demonstrate that the generalized pinball loss function can be estimated using our smooth approximation function and that our model's solutions converge to the exact model's solutions. We perform a comprehensive experimental study using many machine learning benchmark datasets and hundreds of training examples. The experiment results demonstrate that the proposed method remains noise-insensitivity, reduces the computational time, and works excellently in accuracy on binary datasets, achieving up to 1.59% better than the original SVM method in the linear and 1.45% in the nonlinear case. Furthermore, our method surpasses a related model on multi-class datasets by up to 1.43%. We also employ the Friedman test to assess the statistical significance of the performance improvements across various models, finding that our method's average rank surpasses the competing models.

Furthermore, we incorporate the Linex loss function into the LSTBSVM framework to enhance robust classification. By leveraging the Adam algorithm, our

method proves highly effective in handling large-scale classification challenges. We evaluate our method using various small and large benchmark datasets, particularly those with feature noise, by evaluating performance metrics such as accuracy, MCC value, F1 score, and training time. Our method shows competitive robustness compared to other baseline methods. The statistical significance of our performance improvements is confirmed through the Friedman test, which indicates that our method's average rank outperforms competing models. Additionally, we apply our proposed method to an image recognition task using the Knee Osteoarthritis image dataset. Employing a pre-trained VGG16 model for feature extraction, our method outperforms current state-of-the-art methods, demonstrating its effectiveness and versatility.



REFERENCES

REFERENCES

1. Jorge N, Stephen J. W. Numerical Optimization. New York: Springer; 2006.

2. Schraudolph N.N, Yu J, Günter S. A Stochastic quasi-Newton Method for Online Convex Optimization. Proceedings of The 11th International Conference on Artificial Intelligence and Statistics (AISTATS 2007). 2017;2:436–43.

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

4. Byrd R.H. and Khalfan H.F, and Schnabel R.B. Analysis of a Symmetric Rank-One Trust Region Method. SIAM Journal on Optimization. 1996;6(4).

5. Indrapriyadarsini S, Mahboubi S, Ninomiya H, Asai H. A Stochastic Quasi-Newton Method with Nesterov's accelerated gradient, Machine Learning and Knowledge Discovery in Databases. 2020.

6. Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. Proceedings of the 30th International Conference on International Conference on Machine Learning. 2013;28:1139–147.

7. Robbins H, Monro S, A Stochastic Approximation Method. The Annals of Mathematical Statistics. 1915;22(3):400–07.

8. Bock S, Goppold J, Weiß M.G. An improvement of the convergence proof of the ADAM-Optimizer. ArXiv, abs/1804.10587. 2018.

9. Vapnik VN. The Nature of Statistical Learning Theory. New York: Springer; 1995.

10. Shai S, Singer Y, Srebro N, Cotter A. PEGASOS: primal estimated subgradient solver for svm. Math. Program. 2011;127:3–30.

11. Suykens J, Vandewalle J. Least Squares Support Vector Machine Classifiers. Neural Processing Letters. 1999;293–300.

12. John P. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Advances in Kernel Methods-Support Vector Learning. 1998;208.

13. Joachims T. Text categorization with Support Vector Machines: Learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds) Machine Learning: ECML-98. ECML 1998. Lecture Notes in Computer Science, vol 1398. Springer, Berlin, Heidelberg. 1998.

14. Devendran V, Hemalatha T, A.K.Santra S, Amitabh W. (2008). Feature Selection for Scene Categorization Using Support Vector Machines. Proceedings of the IEEE. 2008;588—92.

15. Guodong G, Stan L, Kapluk C. Face Recognition by Support Vector Machines. Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition; 2000 Mar. 28-30; Grenoble, France: IEEE; 2000. p. 196–201.

16. Al-Jumaili S, Al-Azzawi A, Duru A.D, Ibrahim A.A. Covid-19 X-ray image classification using SVM based on Local Binary Pattern. 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey. 2021;383-87.

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

18. Reshma R, Khemchandani R, Pal A, Chandra S. Generalized pinball loss svms. Neurocomputing. 2018;36(5):322.

19. Lee YJ, Mangasarian OL. SSVM: A smooth support vector machine forclassification. Computational Optimization and Applications. 2001;22(1):5–21.

20. Li K, Lv Z. Smooth twin bounded support vector machine with pinball loss. Applied Intelligence. 2021;51,5489–505.

21. Ratiphaphongthon W, Panup W, Wangkeeree R. (2022). A new technique for Pneumonia infected patients image recognition based on combination algorithm of smooth generalized pinball SVM and VAEs. IEEE Access. 2022.

22. Jayadeva, R. Khemchandani, S. Chandra. Twin support vector machines for pattern classification. IEEE Trans. Pattern Anal. Mach. Intell. 2007;29(5):905–10.

23. Kumar MA, Gopal M. Least squares twin support vector machines for pattern classification. Expert Systems with Applications. 2009;36(4):7535–43.

24. Tanveer M, Sharma S, Muhammad K. Large-Scale Least Squares Twin SVMs. ACM Transactions on Internet Technology. 2021;21(2):1–19.

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

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

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

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

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

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

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

32. Deisenroth M, Faisal A, Ong C. Mathematics for Machine Learning. Cambridge: Cambridge University Press; 2020.

33. Solo V, Kong K. Adaptive Signal Processing Algorithms:Stability and Performance. Englewood Cliffs, NJ, USA: Prentice-Hall; 1995.

34. Bache K, Lichman M. UCI Machine Learning Repository [Online]. 1990;92.

35. Hsu C.W, Chang C.C, Lin C.J. A Practical Guide to Support Vector Classification. Nat. Taiwan Univ., Taipei, Taiwan. 2003:1–12.

36. Moreau J.J. Proximité et dualité dans un espace hilbertien. Bulletin de la Société Mathématique de France 93. 1965:273-99.

37. Panup W, Wangkeeree, R. Stochastic Subgradient for Large-Scale Support Vector Machine Using the Generalized Pinball Loss Function. Symmetry. 2021;13:1652.

38. Liu M.Z., Shao Y.H, Li C.N, Chen W.J. Smooth pinball loss nonparallel support vector machine for robust classification. Applied Soft Computing. 2021;98:106840.

39. Panup W, Ratipapongton W, Wangkeeree R. A novel twin support vector machine with generalized pinball loss function for pattern classification. Symmetry. 2022;14(2).

40. Zellner, Arnold. Bayesian Estimation and Prediction Using Asymmetric Loss Functions. Journal of the American Statistical Association. 1986;394:446–51.

41. Shao Y.H, Zhang C.H, Wang X.B, Deng N.Y. Improvements on twin support vector machines. IEEE Transactions on Neural Networks. 2011;22(6):962–68.

42. Osteoarthritis Dataset[Internet]. [updated 2022 Sep 19; cited 2022 Dec 8]. Available from: https://www.kaggle.com/datasets/samantabeginner/osteoarthritis-dataset/.

43. Alshamrani HA, Rashid M, Alshamrani SS, Alshehri AHD. Osteo-NeT: An Automated System for Predicting Knee Osteoarthritis from X-ray Images Using Transfer-Learning-Based Neural Networks Approach. Healthcare. 2023; 11(9):1206.

44. He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA. 2016:770–78.