Lehigh University
Lehigh Preserve

Theses and Dissertations

2019

Optimization Algorithms for Machine Learning Problems

Hiva Ghanbari
Lehigh University

Follow this and additional works at: https://preserve.lehigh.edu/etd

 Part of the Industrial Engineering Commons

Recommended Citation
Ghanbari, Hiva, "Optimization Algorithms for Machine Learning Problems" (2019). *Theses and Dissertations*. 5560.
https://preserve.lehigh.edu/etd/5560

This Dissertation is brought to you for free and open access by Lehigh Preserve. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.

Optimization Algorithms for Machine Learning Problems

by

Hiva Ghanbari

A Dissertation
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy
in
Industrial and Systems Engineering

Lehigh University
May 2019

Copyright
Hiva Ghanbari 2018

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Hiva Ghanbari
Optimization Algorithms for Machine Learning Problems

Date

Dr. Katya Scheinberg, Dissertation Director, Chair
(Must Sign with Blue Ink)

Accepted Date

Committee Members

Dr. Katya Scheinberg, Committee chair

Dr. Frank E. Curtis

Dr. Coralia Cartis

Dr. Martin Takáč

Acknowledgment

Firstly, I would like to express my sincere gratitude to my advisor Prof. Katya Scheinberg who patiently guided me through my Ph.D. studies. Besides my advisor, I would like to thank the rest of my dissertation committee: Prof. Frank E. Curtis, Prof. Coralia Cartis, and Prof. Martin Takáč for all of their kind supports and insightful suggestions to improve the quality of my dissertation.

There is another side of my life with much more depth that made the academic side of life possible. The main roles are being played by my family and friends. Expressing my gratitude of being fortunate to have this family goes beyond the strength of words. My husband, Mohammad, my best friend, who made me stronger, every single step, with his pure endless support. Every moment in my life has been made better because you are in the world. Thank you Mohammad! My parents, Mohammad-Mahdi and Fahimeh, have been supporting me throughout all of these years and they have lived my life for me. They have gone through all the moments I passed. Thank you mom and dad! My older brother, dear lifetime friend, Alireza, and my younger brother, Mohammadreza, I can't imagine the world without you. I want to thank my dear friends Khatereh, Roshanak, and Zahra. I always feel fortunate that we all shared the most part of our long studying journey together. Furthermore, I would like to thank Ms. Rita Frey, the former graduate coordinator in the ISE department and the first person I met in the department. I still remember the sound of her beautiful voice that made an incredible impact of my first day at Lehigh with her warm and thoughtful words.

I lived and studied in Bethlehem for 5 years. I love Bethlehem and its natural beauty, however, this feeling is rooted from the depth of beauty of my friends and our

relationships. I would like to thank my dear friends who made my life at Bethlehem unforgettably beautiful in my memories: Suresh Bolusani, Pelin Cay, Sertalp Cay, Choat Inthawongse, Majid Jahani, Xi He, Yuhai Hu, Xiaolong Kuang, Chenxin Ma, Mohsen Moarefdoost, Matt Menickelly, Ali Mohammad-Nezhad, Golnaz Shahidi, Shu Tu, Martin Takáč, Maria Takáčová, Martin (II) Takáč, Sahar Tahernezhad, Wei Xia, Alireza Yektamaram, Fatma and Metin Yerlikaya-Özkurt.

Last, but not least, I would like to thank Lehigh staff from different departments, who made my Ph.D. life much easier with their help and supports, Ms. Alicia Herzog, Ms. Olga Scarpero and Ms. Brie Lisk.

Contents

List of Tables

List of Figures

Abstract

In the first chapter of this thesis, we analyze the global convergence rate of a proximal quasi-Newton algorithm for solving composite optimization problems, in both exact and inexact settings, in the case when the objective function is strongly convex. We also investigate a practical variant of this method by establishing a simple stopping criterion for the subproblem optimization. Furthermore, we consider an accelerated variant, based on FISTA of Beck and Teboulle [*SIAM J. Optim.,* 2 (2009), pp. 183-202], of the proximal quasi-Newton algorithm. Jiang, Sun, and Toh [*SIAM J. Optim.,* 22 (2012), pp. 1042-1064] considered a similar accelerated method, where the convergence rate analysis relies on very strong impractical assumptions on the Hessian estimates. We present a modified analysis while relaxing these assumptions and perform a numerical comparison of the accelerated proximal quasi-Newton algorithm and the regular one. Our analysis and computational results show that acceleration may not bring any benefit in the quasi-Newton setting.

In the second chapter, we utilize a Trust Region based Derivative Free Optimization (DFO-TR) method to directly maximize the Area Under Receiver Operating Characteristic Curve (AUC), which is a nonsmooth, noisy function. The practical performance of this algorithm is compared to three prominent Bayesian optimization methods and random search. The presented numerical results show that DFO-TR surpasses Bayesian optimization and random search on various black-box optimization problems, such as maximizing AUC and hyperparameter tuning in cost-sensitive SVM.

In the third chapter, we are interested in the predictive quality of machine learning models which is typically measured in terms of their (approximate) expected

prediction error or the so-called AUC for a particular data distribution. However, when the models are constructed by means of empirical risk minimization, surrogate functions such as the logistic loss are optimized instead. This is done because the empirical approximations of the expected error and AUC functions are nonconvex and nonsmooth, and more importantly have zero derivative almost everywhere. In this work, we show that in the case of linear predictors, and under the assumption that the data is normally distributed, the expected error and the expected AUC are not only smooth, but have closed form expressions, which depend on the first and second moments of the normal distribution. However, by using some variants of the central limit theorem, we showed that under some conditions, the expected error is a smooth function in the limit. This result, similarly, can be extended to the case of the AUC function. Hence, we derive derivatives of these two functions and use these derivatives in an optimization algorithm to directly optimize the expected error and the AUC. In the case of real data sets, the derivatives can be approximated using empirical moments. We show that even when data is not normally distributed, computed derivatives are sufficiently useful to render an efficient optimization method and high quality solutions. Thus, we propose a gradient-based optimization method for direct optimization of the prediction error and AUC. Moreover, the per-iteration complexity of the proposed algorithm has no dependence on the size of the data set, unlike those for optimizing logistic regression and all other well known empirical risk minimization problems.

Chapter 1

Introduction

This thesis starts by analyzing theoretical and practical properties of some variants of proximal quasi-Newton algorithms for nonsmooth convex problems. Due to the structure of many machine learning problems such as sparse logistic regression, sparse inverse covariance selection, and unconstrained Lasso, nonsmooth convex problems arise in various applications. These problems, in these applications, are indeed unconstrained, containing a large number of convex loss functions and a possibly nonsmooth regularization part. The loss functions, however, can be strongly convex, which is the case in most of the machine learning applications.

In the first part of this thesis, we introduced an unprecedented property of a class of proximal quasi-Newton algorithms by proving linear convergence rate for the algorithm when applied to strongly convex loss functions. In our work, furthermore, we designed an accelerated variant of the proximal quasi-Newton algorithm with a guaranteed global convergence rate. This is the first concrete convergence result of combining the use of second-order information and Nesterov's accelerated scheme.

In the second part of this thesis, we addressed the challenges of learning from imbalanced data sets. Many real-world machine learning problems deal with imbalanced data sets containing rare positive data points as the minority class, but numerous negative ones as the majority class, or vice versa. Standard machine learning tools such as support vector machines and logistic regression, which aim to find a classifier optimizing accuracy, fail in such problems.

Recently, in the data mining community, instead of accuracy, another performance measure for classifiers has been getting popular, especially when dealing with imbalanced data sets: the Area Under receiver operating characteristic Curve (AUC). However, since AUC is highly nondifferentiable, gradient-based optimization methods cannot be applied directly. We prove that, despite the nondifferentiability of AUC, its expectation, which is indeed the quantity that should be optimized in the learning process, is a smooth function under some practical conditions. This novel result allows any gradient-based optimization algorithm to be applied in the training process to maximize the expectation of AUC. The resulting classifier would then have theoretical guarantees to classify data points correctly with a high probability.

Moreover, we discovered that the same approach can be applied to directly minimize the expected zero-one loss function, the corner-stone of statistical machine learning, through any smooth optimization algorithm. This method, unlike many other learning approaches such as minimizing logistic loss and pair-wise hinge loss– as approximates of zero-one loss and AUC functions, respectively–is independent of the size of the training data set, so it would be extremely fast, especially when we have an enormous training set.

Furthermore, one may use the smoothness of AUC in expectation as the theoretical base to apply Derivative-Free Optimization (DFO), as we utilized a trust region based DFO method to directly optimize AUC function. We also investigated the performance of a trust region based DFO method versus Bayesian Optimization for hyperparameter tuning in machine learning problems.

Chapter 2

Proximal Quasi-Newton Methods for Regularized Convex Optimization

2.1 Introduction

In this chapter, we address the convex optimization problem of the form

$$\min_x \{F(x) := f(x) + g(x),\ x \in \mathbb{R}^n\}, \tag{2.1}$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is a continuous convex function which is possibly nonsmooth and $f : \mathbb{R}^n \to \mathbb{R}$ is a convex continuously differentiable function with Lipschitz continuous gradient, i.e.,

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2 \quad \text{for every} \quad x, y \in \mathbb{R}^n,$$

where L is the global Lipschitz constant of the gradient ∇f. This class of problems, when $g(x) = \lambda\|x\|_1$, contains some of the most common machine learning models such as sparse logistic regression [54, 63], sparse inverse covariance selection [23, 42, 47], and unconstrained Lasso [58].

The Proximal Gradient Algorithm (PGA) is a variant of the *proximal methods* and is a well-known first-order method for solving optimization problem (2.1). Although classical *subgradient methods* can be applied to problem (2.1) when g is nonsmooth, they can only achieve the rate of convergence of $\mathcal{O}(1/\sqrt{k})$ [38], while PGA converges at a rate of $\mathcal{O}(1/k)$ in both smooth and nonsmooth cases [2, 40]. In order to improve the global sublinear rate of convergence of PGA further, the Accelerated Proximal Gradient Algorithm (APGA) has been originally proposed by Nesterov in [37], and refined by Beck and Teboulle in [2]. It has been shown that the APGA provides a significant improvement compared to PGA, both theoretically, with a rate of convergence of $\mathcal{O}(1/k^2)$, and practically [2]. This rate of convergence is known to be the best that can be obtained using only first-order information [36, 38, 50], causing APGA to be known as an *optimal first-order method*. The class of accelerated methods contains many variants that share the same convergence rates and use only first-order information [38, 39, 60]. The main known drawback of most of the variants of APGA is that the sequence of the step-size $\{\mu_k\}$ has to be nonincreasing. This theoretical restriction sometimes has a big impact on the performance of this algorithm in practice. In [50], in order to overcome this difficulty, a new version of APGA has been proposed. This variant of APGA allows to increase step-sizes from one iteration to the next, but maintain the same rate of convergence of $\mathcal{O}(1/k^2)$. In particular, the authors have shown that a full backtracking strategy can be applied in APGA and that the resulting complexity of the algorithm depends on the average value of step-size parameters, which is closely related to local Lipschitz constants, rather than the global one.

To make PGA and APGA practical, for some complicated instances of (2.1), one needs to allow for inexact computations in the algorithmic steps. In [52], inexact variants of PGA and APGA have been analyzed with two possible sources of error: one in the calculation of the gradient of the smooth term and the other in the proximal operator with respect to the nonsmooth part. The convergence rates are preserved if the sequence of errors converges to zero sufficiently fast. Moreover, it has been shown that both of these algorithms obtain a linear rate of convergence, when

the smooth term f is strongly convex[1]. Recently, in [17], the linear convergence of PGA has been shown under the *quadratic growth condition*, which is weaker than a strong convexity assumption. In particular, their analysis relies on the fact that PGA linearly bounds the distance to the solution set by the step lengths. This property, called an *error bound condition*, has been proved to be equivalent to the standard quadratic growth condition. More precisely, a strong convexity assumption is a sufficient, but not a necessary condition for this error bound property.

While PGA and APGA can be efficient in solving (2.1), it has been observed that using (partial) second-order information often significantly improves the performance of the algorithms. Hence, Newton type proximal algorithms, also known as the *proximal Newton methods*, have become popular [7, 30, 42, 48, 56] and are often the methods of choice. When accurate (or nearly accurate) second-order information is used, the method no longer falls in the first-order category and faster convergence rates are expected, at least locally. Indeed, the global convergence and the local superlinear rate of convergence of the Proximal Quasi-Newton Algorithm (PQNA) are presented in [30] and [7], respectively for the both exact and inexact settings. However, in the case of limited memory BFGS method [8, 41], the method is still essentially a first-order method. While practical performance may be by far superior, the rates of convergence are at best the same as those of the pure first-order counterparts. In [48], an inexact PQNA with global sublinear rate of $\mathcal{O}(1/k)$ is proposed. While the algorithm can use any positive definite Hessian estimates, as long as their eigenvalues are uniformly bounded above and away from zero, the practical implementation proposed in [48] used a limited memory BFGS Hessian approximation. The inexact setting of the algorithm allows for a relaxed sufficient decrease condition as well as an inexact subproblem optimization, for example via coordinate descent.

In this work, we show that PQNA, as proposed in [48], using general Hessian estimates H_k, has the linear convergence rate in the case of strongly convex smooth term f. Moreover, we consider an inexact variant, similar to the ones in [7, 48],

[1]For APGA a different variant is analyzed in the case of strong convexity.

allowing inexact subproblem solutions as well as a relaxed sufficient decrease condition. In order to control the errors in the inexact subproblem optimization, we establish a simple stopping criterion for the subproblem solver, based on the iteration count, which guarantees that the inner subproblem is solved to the required accuracy. In contrast, in related works [26, 61], it is assumed that an approximate subproblem solver yields an approximate subdifferential, which is a strong assumption on the subproblem solver which also does not clearly result in a simple stopping criterion.

Next, we apply Nesterov's acceleration scheme to PQNA as proposed in [48], with a view of developing a version of this algorithm with a faster convergence rate in the general convex case. In [26], the authors have introduced the Accelerated Proximal Quasi-Newton Algorithm (APQNA) with rate of convergence of $\mathcal{O}(1/k^2)$. However, this rate of convergence is achieved under condition $0 \prec H_k \preceq H_{k-1}$, on the Hessian estimate H_k, at each iteration k. At the same time, this sequence of matrices has to be chosen so that H_k is sufficiently positive definite to provide an overapproximation of f. Hence, these two conditions may contradict with each other unless the sequence of $\{H_k\}$ consists of unnecessarily large matrices. Moreover, in a particular case, when H_k is set to be a scalar multiple of the identity, i.e., $H_k = \frac{1}{\mu_k} I$, then assumption $0 \prec H_k \preceq H_{k-1}$ enforces $\mu_k \geq \mu_{k-1}$, implying nondecreasing step-size parameters, which contradicts the standard condition of APGA, which is $\mu_k \leq \mu_{k-1}$.

In this work, we introduce a new variant of APQNA, where we relax the restrictive assumptions imposed in [26]. We use the scheme, originally introduced in [50], which allows for the increasing and decreasing step-size parameters. We show that our version of APQNA achieves the convergence rate of $\mathcal{O}(1/k^2)$ under some assumptions on the Hessian estimates. While we show that this assumption is rather strong and may not be satisfied by general matrices, it is not contradictory. Firstly, our result applies under the same restrictive condition from [26]. We also show that our condition on the matrices holds in the case when the approximate Hessian at each iteration is a scaled version of the same "fixed" matrix H, which is a

generalization of APGA. We investigate the performance of this algorithm in practice, and discover that this restricted version of a proximal quasi-Newton method is quite effective in practice. We also demonstrate that the general L-BFGS based PQNA does not benefit from the acceleration, which supports our analysis of the theoretical limitations.

This chapter is organized as follows. In the next section, we describe the basic definitions and existing algorithms, PGA, APGA and PQNA, that we refer to later in this chapter. In Section 2.3, we analyze PQNA in the strongly convex case. In Section 2.4, we propose, state and analyze a general APQNA and its simplified version. We present computational results in Section 2.5. Finally, we state our conclusions in Section 2.6.

2.2 Notation and Preliminaries

In this work, the Euclidean norm $\|a\|^2 := a^T a$, and the inner product $\langle a, b\rangle := a^T b$, are also defined in the scaled setting such that, $\|a\|_H^2 := a^T H a$, and $\langle a, b\rangle_H := a^T H b$. We denote the identity matrix by $I \in \mathbb{R}^{n\times n}$. The vector e_j stands for a unit vector along the j-th coordinate. We use x_k to denote the approximate minimizer (the iterate), computed at iteration k of an appropriate algorithm, and x_* to denote an exact minimizer of F. Finally, $(\partial F(x))_{\min}$ denotes the minimum norm subgradient of function F at point x.

The *proximal mapping* of a convex function g at a given point v, with parameter μ is defined as

$$prox_g^\mu(v) := \arg\min_{u\in\mathbb{R}^n}\{g(u) + \frac{1}{2\mu}\|u - v\|^2\}, \quad \text{where} \quad \mu > 0. \tag{2.2}$$

The proximal mapping is the base operation of the *proximal methods*. In order to solve the composite problem (2.1), each iteration of PGA computes the proximal

mapping of the function g at point $x_k - \mu_k \nabla f(x_k)$ as follows:

$$\begin{aligned} p_{\mu_k}(x_k) &:= prox_g^{\mu_k}(x_k - \mu_k \nabla f(x_k)) \\ &:= \arg\min_{u \in \mathbb{R}^n} \{g(u) + f(x_k) + \langle \nabla f(x_k), u - x_k \rangle + \frac{1}{2\mu_k} \|u - x_k\|^2\}. \end{aligned} \tag{2.3}$$

We will call the objective function, that is minimized in (2.3), a *composite quadratic approximation* of the convex function $F(x) := f(x) + g(x)$. This approximation at a given point v, for a given μ is defined as

$$Q_\mu(u, v) := f(v) + \nabla f(v)^T (u - v) + \frac{1}{2\mu} \|u - v\|^2 + g(u). \tag{2.4}$$

Then, the proximal operator can be written as

$$p_\mu(v) := \arg\min_{u \in \mathbb{R}^n} Q_\mu(u, v).$$

Using this notation we first present the basic PGA framework with backtracking over μ in Algorithm 1. The simple backtracking scheme enforces that the sufficient decrease condition

$$F(x_{k+1}) \leq Q_{\mu_k}(x_{k+1}, x_k) \leq Q_{\mu_k}(x_k, x_k) = F(x_k) \tag{2.5}$$

holds. This condition is essential in the convergence rate analysis of PGA and is easily satisfied when $\mu \leq 1/L$. The backtracking is used for two reasons–because the constant L may not be known and because $\mu \leq 1/L$ may be too pessimistic, i.e., condition (2.5) may be satisfied for much larger values of μ allowing for larger steps.

We now present the accelerated variant of PGA stated as APGA, where at each iteration k, instead of constructing Q_{μ_k} at the current minimizer x_k, it is constructed at a different point y_k, which is chosen as a specific linear combination of the latest two or more minimizers, e.g.,

$$y_{k+1} = x_k + \alpha_k (x_k - x_{k-1}),$$

Algorithm 1 Proximal Gradient Algorithm

Initialize $x_0 \in \mathbb{R}^n$, and choose $\beta \in (0,1)$.
for $k = 1, 2, \cdots$ **do**
  Choose μ_k^0 and define $\mu_k := \mu_k^0$.
  Compute $p_{\mu_k}(x_k) := \arg\min_{u \in \mathbb{R}^n} Q_{\mu_k}(u, x_k)$.
  while $F\left(p_{\mu_k}(x_k)\right) > Q_{\mu_k}\left(p_{\mu_k}(x_k), x_k\right)$ **do**
    Set $\mu_k \leftarrow \beta \mu_k$.
    Compute $p_{\mu_k}(x_k) := \arg\min_{u \in \mathbb{R}^n} Q_{\mu_k}(u, x_k)$.
  Set $x_{k+1} \leftarrow p_{\mu_k}(x_k)$.

where the sequence $\{\alpha_k\}$ is chosen in such a way to guarantee an accelerated convergence rate as compared to the original PGA. Algorithm 2 is a variant of APGA framework, often referred to as FISTA, presented in [2], where $\alpha_k = (t_k - 1)/(t_{k+1})$. In this work, we choose to focus on FISTA algorithm specifically. The choice of the accelerated parameter t_{k+1} in (2.6a) is dictated by the analysis of the complexity of FISTA [2] and the condition $\mu_{k+1} \leq \mu_k$ that is imposed by the initialization of the backtracking procedure with $\mu_{k+1}^0 := \mu_k$. In [50], the definition of t_{k+1} was generalized to allow $\mu_{k+1}^0 > \mu_k$, while retaining the convergence rate. We will use a similar technique in our proposed APQNA.

Algorithm 2 Accelerated Proximal Gradient Algorithm

Initialize $t_1 = 1$, $\mu_1^0 > 0$, and $y_1 = x_0 \in \mathbb{R}^n$, and choose $\beta \in (0,1)$.
for $k = 1, 2, \cdots$ **do**
  Define $\mu_k := \mu_k^0$.
  Compute $p_{\mu_k}(y_k) := \arg\min_{u \in \mathbb{R}^n} Q_{\mu_k}(u, y_k)$.
  while $F\left(p_{\mu_k}(y_k)\right) > Q_{\mu_k}\left(p_{\mu_k}(y_k), y_k\right)$ **do**
    Set $\mu_k \leftarrow \beta \mu_k$.
    Compute $p_{\mu_k}(y_k) := \arg\min_{u \in \mathbb{R}^n} Q_{\mu_k}(u, y_k)$.
  Set $x_k \leftarrow p_{\mu_k}(y_k)$.
  Define $\mu_{k+1}^0 := \mu_k$ and compute t_{k+1} and y_{k+1}, so that

$$t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right) \tag{2.6a}$$

$$\text{and} \quad y_{k+1} = x_k + \frac{t_k - 1}{t_{k+1}}\left(x_k - x_{k-1}\right). \tag{2.6b}$$

In this work, we are interested in the extensions of the above proximal methods, which utilize an approximation function Q_μ, using partial second-order information about f. These quasi-Newton type proximal algorithms use a generalized form of the proximal operator (2.2), known as the *scaled proximal mapping* of g, which are defined for a given point v as

$$prox_g^H(v) := \arg\min_{u\in\mathbb{R}^n}\{g(u) + \frac{1}{2}\|u - v\|_H^2\},$$

where matrix H is a positive definite matrix. In particular, the following operator

$$p_{H_k}(x_k) := prox_g^{H_k}\left(x_k - H_k^{-1}\nabla f(x_k)\right) \tag{2.7}$$

computes the minimizer, over u, of the following composite quadratic approximation of function F

$$Q_H(u, v) := f(v) + \langle\nabla f(v), u - v\rangle + \frac{1}{2}\|u - v\|_H^2 + g(u), \tag{2.8}$$

when $v = x_k$. Matrix H is the approximate Hessian of f and its choice plays the key role in the quality of this approximation. Clearly, when $H = \frac{1}{\mu}I$, approximation (2.8) converts to (2.4), which is used throughout PGA. If we set $H = \nabla^2 f(v)$, then (2.8) is the second-order Taylor approximation of F. At each iteration of PQNA the following optimization problem needs to be solved

$$p_H(v) := \arg\min_{u\in\mathbb{R}^n} Q_H(u, v), \tag{2.9}$$

which we assume to be computationally inexpensive relative to solving (2.1) for any $v \in \mathbb{R}^n$ and some chosen class of positive definite approximate Hessian H. Our assumption is motivated by [48], where it is shown that for L-BFGS Hessian approximation, problem (2.9) can be solved efficiently and inexactly via coordinate descent method. Specifically, the proximal operator (2.7) does not have closed form solution for most types of Hessian estimates H_k and most nonsmooth terms g, such as $g = \lambda\|x\|_1$, when (2.7) is a convex quadratic optimization problem. Hence, it may

be too expensive to seek the exact solution of subproblem (2.7) in every iteration. In [48], an efficient version of PQNA is proposed which constructs Hessian estimates based on the L-BFGS updates, resulting in H_k matrices that are sum of a diagonal and a low rank matrix. The resulting subproblem, structured as (2.7), is then solved up to some expected accuracy via randomized coordinate descent, which effectively exploits the special structure of H_k. The analysis in [48] shows that the resulting inexact PQNA converges sublinearly if the Hessian estimates remain positive definite and bounded, without assuming any other structure. In this work, all the theory is derived for arbitrary positive definite Hessian estimates, without any assumption on their structure, or how closely they are representing the true Hessian. In our implementation, however, we will construct the Hessian estimates via L-BFGS as it is done in [48]. The framework of the inexact variant of PQNA for general approximate Hessian is stated in Algorithm 3. In this algorithm, the inexact solution of (2.9) is denoted by $p_{H,\epsilon}(v)$, as an $\epsilon-$minimizer of the subproblem that satisfies

$$g(p_{H,\epsilon}(v)) + \frac{1}{2}\|p_{H,\epsilon}(v) - z\|_H^2 \leq \min_{u\in\mathbb{R}^n}\{g(u) + \frac{1}{2}\|u - z\|_H^2\} + \epsilon, \tag{2.10}$$

where $z := v - H^{-1}\nabla f(v)$. Obtaining such an inexact solution can be achieved by applying any linearly convergent algorithm, as will be discussing in detail at the end of this section.

In addition, for a given $\eta \in (0, 1]$, the typical condition (2.5), used in [2] and [51], is relaxed by using a trust region like sufficient decrease condition

$$\left(F\left(p_{H,\epsilon}(v)\right) - F(v)\right) \leq \eta\left(Q_H\left(p_{H,\epsilon}(v), v\right) - F(v)\right). \tag{2.11}$$

This relaxed condition was proposed and tested in [48] for PQNA and was shown to lead to superior numerical performance, saving multiple backtracking steps during the earlier iterations of the algorithm. Note that, one can obtain the exact version of Algorithm 3 by replacing p_{H_k,ϵ_k} with p_{H_k}, and setting $\eta = 1$.

Algorithm 3 Inexact Proximal Quasi-Newton Algorithm

Initialize $x_0 \in \mathbb{R}^n$, and choose $\beta \in (0,1)$ and $\eta \in (0,1]$.
for $k = 1, 2, \cdots$ **do**
  Choose $\mu_k > 0$ and bounded $G_k \succeq 0$.
  Define $H_k := G_k + \frac{1}{2\mu_k} I$.
  Compute $p_{H_k,\epsilon_k}(x_k)$ such that (2.10) is satisfied.
  while $\left(F\left(p_{H_k,\epsilon_k}(x_k)\right) - F(x_k)\right) > \eta\left(Q_{H_k}\left(p_{H_k,\epsilon_k}(x_k), x_k\right) - F(x_k)\right)$ **do**
    Set $\mu_k \leftarrow \beta\mu_k$.
    Update H_k via $H_k := G_k + \frac{1}{2\mu_k} I$.
    Compute $p_{H_k,\epsilon_k}(x_k)$ such that (2.10) is satisfied.
  Set $x_{k+1} \leftarrow p_{H_k,\epsilon_k}(x_k)$.

Throughout our analysis, we make the following assumptions.

Assumption 1.

- *f is convex with Lipschitz continuous gradient with constant L.*
- *g is a lower semi-continuous proper convex function.*
- *There exists an $x_* \in \mathbb{R}^n$, which is a minimizer of F.*
- *There exist positive constants m and M such that, for all $k > 0$,*

$$mI \preceq H_k \preceq MI. \tag{2.12}$$

Remark 1. *In Algorithm 3, as long as the sequence of positive definite matrices G_k has uniformly bounded eigenvalues, condition (2.12) is satisfied. In fact, since the sufficient decrease condition in Step 3 is satisfied for $H_k \succeq LI$, then it is satisfied when $\mu_k \leq 1/L$. Hence, at each iteration we have a finite and bounded number of backtracking steps and the resulting H_k has bounded eigenvalues. The lower bound on the eigenvalues of H_k is simply imposed either by choosing a positive definite G_k or bounding μ_k^0 from above.*

In the next section, we analyze the convergence properties of PQNA when f in (2.1) is strongly convex.

2.3 Proximal Quasi-Newton Algorithm under Strong Convexity

In this section, we analyze the convergence properties of PQNA to solve problem (2.1), in the case when the smooth function f is $\gamma-$strongly convex. In particular, the following assumption is made throughout this section.

Assumption 2. *For all x and y in $\mathbb{R}^n$, and any $t \in [0,1]$, the following two equivalent conditions hold.*

$$\gamma\|x-y\|^2 \leq \langle \nabla f(x) - \nabla f(y), x-y\rangle \tag{2.13a}$$

$$\text{and} \quad f\left(tx+(1-t)y\right) \leq tf(x)+(1-t)f(y) - \frac{1}{2}\gamma t(1-t)\|x-y\|^2. \tag{2.13b}$$

To establish a linear convergence rate of PQNA, we consider extending two different approaches used to show a similar result for PGA. The first approach we consider is based on the proof techniques used in [17] for PGA. The reason we chose the approach in [17] is due to the fact that the linear rate of convergence is shown under the quadratic growth condition, which is a relaxation of the strong convexity. Hence, extending this analysis to PQNA, as a subject of a future work, may allow us to relax the strong convexity assumption for this algorithm as well. However, there appears to be some limitations in the extension of this analysis, in particular in the inexact case. This observation motivates us to present the approach used in [40] to analyze convergence properties of inexact PQNA. As we see below, this analysis readily extends to our case and allows us to establish simple rules for subproblem solver termination to achieve the desired subproblem accuracy.

2.3.1 Convergence Analysis

2.3.1.1 Exact Case (First Approach)

Here, we present the first approach establishing the linear convergence rate of *PQNA*. First, we state a simple result based on the optimality of p_H.

Lemma 1. *For any $v \in \mathbb{R}^n$, there exists a subgradient of function g where $\nu_g(p_H(v)) \in \partial g(p_H(v))$ such that*

$$\nabla f(v) + H(p_H(v) - v) + \nu_g(p_H(v)) = 0. \tag{2.14}$$

Proof. The proof is followed immediatly from the optimality condition of convex optimization problem (2.9). □

We also need the following property of subgradients of a convex function.

Lemma 2. *Subgradients of a convex function g are monotone operators such that*

$$(\nu_g(x) - \nu_g(y))^T(x - y) \geq 0, \qquad \forall x, y \in \operatorname{dom} g,$$

where $\nu_g \in \partial g$.

Proof. The proof is implied by the definition of subgradients. □

Now, we can show the following lemma, which bounds the change in the objective function F and is a simple extension of a similar theorem in [2].

Lemma 3. *Let $v \in \mathbb{R}^n$ and $H \succ 0$ be such*

$$F(p_H(v)) \leq Q_H(p_H(v), v), \tag{2.15}$$

holds for a given v, then for any $x \in \mathbb{R}^n$

$$F(x) - F(p_H(v)) \geq \frac{1}{2}\|p_H(v) - v\|_H^2 + \langle v - x, p_H(v) - v\rangle_H. \tag{2.16}$$

Proof. From (2.57) we have

$$F(x) - F(p_H(v)) \geq F(x) - Q_H(p_H(v), v). \tag{2.17}$$

Now, based on the convexity of functions f and g, we have

$$\begin{aligned} f(x) &\geq f(v) + \langle \nabla f(v), x - v \rangle \quad \text{and} \\ g(x) &\geq g(p_H(v)) + \langle \nu_g(p_H(v)), x - p_H(v) \rangle, \end{aligned}$$

where $\nu_g(p_H(v))$ is defined in Lemma 5. Summing the above inequalities yields

$$F(x) \geq f(v) + \langle \nabla f(v), x - v \rangle + g(p_H(v)) + \langle \nu_g(p_H(v)), x - p_H(v) \rangle. \tag{2.18}$$

Using (2.8) and (2.59) in (2.58) yields

$$\begin{aligned} F(x) - F(p_H(v)) \geq\ & f(v) + \langle \nabla f(v), x - v \rangle + g(p_H(v)) + \langle \nu_g(p_H(v)), x - p_H(v) \rangle \\ & - f(v) - \langle \nabla f(v), p_H(v) - v \rangle - \frac{1}{2}\|p_H(v) - v\|_H^2 - g(p_H(v)) \\ =& -\frac{1}{2}\|p_H(v) - v\|_H^2 + \langle x - p_H(v), \nabla f(v) + \nu_g(p_H(v)) \rangle \\ =& -\frac{1}{2}\|p_H(v) - v\|_H^2 + \langle x - p_H(v), H(v - p_H(v)) \rangle \\ =& -\frac{1}{2}\|p_H(v) - v\|_H^2 + \langle x - p_H(v), H(v - p_H(v)) \rangle \\ & + \langle v - p_H(v), v - p_H(v) \rangle_H - \langle v - p_H(v), v - p_H(v) \rangle_H \\ =& \frac{1}{2}\|p_H(v) - v\|_H^2 + \langle v - x, p_H(v) - v \rangle_H. \end{aligned}$$

□

By using Lemma 6, we can conclude the following results.

Corollary 1. *By setting $x = x_*$, $v = x_k$, $p_H(x_k) = x_{k+1}$, and $H = H_k$ in the result of Lemma 6, we have*

$$F(x_{k+1}) - F(x_*) \leq -\frac{1}{2}\|x_k - x_{k+1}\|_{H_k}^2 + \langle x_k - x_{k+1}, x_k - x_* \rangle_{H_k}. \tag{2.19}$$

With $x = v = x_k$, $p_H(x_k) = x_{k+1}$, and $H = H_k$ we have

$$F(x_k) - F(x_{k+1}) \geq \frac{1}{2}\|x_{k+1} - x_k\|_{H_k}^2. \tag{2.20}$$

Now, under Assumptions 1 and 2, the following theorem shows geometric decrease in function values.

Theorem 1. *At each step of the PQNA the following improvement in the objective function value is guaranteed*

$$F(x_{k+1}) - F(x_*) \leq (1 - \frac{m}{2M\gamma_k})(F(x_k) - F(x_*)), \tag{2.21}$$

where $\gamma_k = \|x_k - x_*\|/\|x_k - x_{k+1}\|$.

Proof. By using inequality (2.19), we have

$$\begin{aligned} F(x_{k+1}) - F(x_*) &\leq M\|x_k - x_{k+1}\|\|x_k - x_*\| - \frac{m}{2}\|x_k - x_{k+1}\|^2 \\ &= \|x_k - x_{k+1}\|^2(M\frac{\|x_k - x_*\|}{\|x_k - x_{k+1}\|} - \frac{m}{2}) \\ &= \|x_k - x_{k+1}\|^2(M\gamma_k - \frac{m}{2}), \end{aligned}$$

where $\gamma_k = \|x_k - x_*\|/\|x_k - x_{k+1}\|$. Now, by using inequality (2.20), we will have

$$\begin{aligned} F(x_{k+1}) - F(x_*) &\leq (F(x_k) - F(x_{k+1}))(\frac{2M}{m}\gamma_k - 1) \\ &\leq ((F(x_k) - F(x_*)) - (F(x_{k+1}) - F(x_*)))\,(\frac{2M}{m}\gamma_k - 1), \end{aligned}$$

which implies

$$F(x_{k+1}) - F(x_*) \leq (1 - \frac{m}{2M\gamma_k})(F(x_k) - F(x_*)). \tag{2.22}$$

□

Based on Theorem 2.48, if the quantities γ_k are bounded for all large k, the Q-linear convergence in the function values follows automatically, which motivates the following theorem.

Theorem 2. *Let Assumptions 1 and 2 hold, then during the application of PQNA,*

for all k we have

$$\gamma_k \leq \frac{M(L+m)}{\mu m} \quad \textit{and} \tag{2.23a}$$

$$\gamma_k \geq \frac{m^2}{M(L+m)}, \tag{2.23b}$$

where $\gamma_k = \|x_k - x_*\|/\|x_k - x_{k+1}\|$.

Proof. In (2.2), by defining function $h(u) := g(u) + \frac{1}{2}\|u - v\|_H^2$, the $prox_g^H(v)$ will be the minimizer of function h, which implies that for any $x \in \mathbb{R}^n$, there exists a $\nu_h(prox_g^H(v)) \in \partial h(prox_g^H(v))$ such that

$$\nu_h(prox_g^H(v))^T(x - prox_g^H(v)) = 0. \tag{2.24}$$

Using $\nu_h(prox_g^H(v)) = \nu_g(prox_g^H(v)) + H(prox_g^H(v) - v)$, where $\nu_g(prox_g^H(v)) \in \partial g(prox_g^H(v))$ in the above inequality, implies

$$(\nu_g(prox_g^H(v)) + H(prox_g^H(v) - v))^T(x - prox_g^H(v)) = 0,$$

which states the following inequality by substituting $x = x_*$, such that

$$\nu_g(prox_g^H(v))^T(x_* - prox_g^H(v)) + (prox_g^H(v) - v)^T H(x_* - prox_g^H(v)) = 0. \tag{2.25}$$

By introducing $\tau := H(x - prox_g^H(x - H^{-1}\nabla f(x)))$, we have $prox_g^H(x - H^{-1}\nabla f(x)) = x - H^{-1}\tau$, and consequently, by setting $v = x - H^{-1}\nabla f(x)$ in (2.25), we will have

$$\nu_g(x - H^{-1}\tau)^T(x_* - x + H^{-1}\tau) + (H^{-1}\nabla f(x) - H^{-1}\tau)^T H(x_* - x + H^{-1}\tau) = 0. \tag{2.26}$$

Now, since x_* is the minimum of the composite function F, we have

$$(x - H^{-1}\tau - x_*)^T(\nabla f(x_*) + \nu_g(x_*)) = 0. \tag{2.27}$$

Summing up two inequalities (2.26) and (2.27) implies,

$$-[(\nu_g(x-H^{-1}\tau)-\nu_g(x_*))^T((x-H^{-1}\tau)-x_*)]+(x-H^{-1}\tau-x_*)^T(\nabla f(x_*)-(\nabla f(x)-\tau))=0.$$

Now, by using the monotonicity of the subgradient, as described in Lemma 2, in above equality, we will have

$$(x-H^{-1}\tau-x_*)^T(\nabla f(x_*)-(\nabla f(x)-\tau))\geq 0,$$

or equivalently,

$$(x-x_*)^T(\nabla f(x)-\nabla f(x_*))\leq \tau^T(H^{-1}(\nabla f(x)-\nabla f(x_*))+x-x_*)-\tau^T H^{-1}\tau. \quad (2.28)$$

By using (2.13a) and (2.12), we have

$$\mu\|x-x_*\|^2\leq\|\tau\|(\frac{1}{m}\|\nabla f(x)-\nabla f(x_*)\|+\|x-x_*\|).$$

By using the Lipschitz continuity of the gradient ∇f, we will have

$$\mu\|x-x_*\|^2\leq\|\tau\|(\frac{L}{m}+1)\|x-x_*\|.$$

Finally, by substituting the definition of τ and defining $x=x_k$, we obtain (2.23a)

$$\frac{\|x_k-x_*\|}{\|x_k-x_{k+1}\|}\leq\frac{M(L+m)}{\mu m}.$$

On the other hand, since function f is a convex function, we have $(x-x_*)^T(\nabla f(x)-\nabla f(x_*))\geq 0$, and consequently, we can rewrite equality (2.28) as

$$\begin{aligned}\tau^T(H^{-1}(\nabla f(x)-\nabla f(x_*))+x-x_*)-\tau^T H^{-1}\tau&\geq 0\\ \|\tau\|(\frac{1}{m}\|\nabla f(x)-\nabla f(x_*)\|+\|x-x_*\|)&\geq\frac{1}{M}\|\tau\|^2\\ \|\tau\|(\frac{L}{m}+1)\|x-x_*\|&\geq\frac{1}{M}\|\tau\|^2,\end{aligned}$$

which implies (2.23b) by substituting the definition of τ and setting $x = x_k$. □

Finally, we can establish the linear convergence of *PQNA*.

Corollary 2. *By using bound* (2.23a) *in the result of Theorem 2.48, we obtain the following final global linear rate of convergence for PQNA such that*

$$F(x_{k+1}) - F(x_*) \leq \left(1 - \frac{\mu m^2}{2M^2(L+M)}\right)(F(x_k) - F(x_*)). \tag{2.29}$$

In the following, we analyze the convergence result of an inexact variant of *PQNA*.

2.3.1.2 Inexact Case (First Approach)

First, in terms of inexact subproblem solvers, stated in (2.10), since point $p_{H,\varepsilon}$ is an $\varepsilon-$minmizer of the convex function $g(u) + \frac{1}{2}\|u - z\|_H^2$, there exists vector ζ with small norm $\|\zeta\|$, so that

$$H(z - p_{H,\varepsilon}(v)) - \zeta \in \partial g(p_{H,\varepsilon}(v)), \tag{2.30}$$

where $z := v - H^{-1}\nabla f(v)$.

Equivalently we have

$$\nabla f(v) + \nu_g(p_{H,\varepsilon}(v)) + H(p_{H,\varepsilon}(v) - v) = \zeta,$$

where $\nu_g(p_{H,\varepsilon}(v)) \in \partial g(p_{H,\varepsilon}(v))$.

Lemma 4. *Let $v \in \mathbb{R}^n$ and $H \succ 0$ be such that*

$$F(p_{H,\varepsilon}(v)) \leq Q_H(p_{H,\varepsilon}(v), v) + \xi, \tag{2.31}$$

then, for any $x \in \mathbb{R}^n$ we have

$$F(x) - F(p_{H,\varepsilon}(v)) \geq \frac{1}{2}\|p_{H,\varepsilon}(v) - v\|_H^2 + \langle v - x, p_{H,\varepsilon}(v) - v\rangle_H + \langle x - p_{H,\varepsilon}(v), \zeta\rangle - \xi. \tag{2.32}$$

Proof. From (2.31) we have

$$F(x) - F(p_{H,\varepsilon}(v)) \geq F(x) - Q_H(p_{H,\varepsilon}(v), v) - \xi. \tag{2.33}$$

Now, based on the convexity of function f and g we have

$$\begin{aligned} f(x) &\geq f(v) + \langle \nabla f(v), x - v \rangle \quad \text{and} \\ g(x) &\geq g(p_{H,\varepsilon}(v)) + \langle \nu_g(p_{H,\varepsilon}(v)), x - p_{H,\varepsilon}(v) \rangle, \end{aligned}$$

where $\nu_g(p_{H,\varepsilon}(v)) \in \partial g(p_{H,\varepsilon}(v))$. Summing the above inequalities yields

$$F(x) \geq f(v) + \langle \nabla f(v), x - v \rangle + g(p_H(v)) + \langle \nu_g(p_H(v)), x - p_H(v) \rangle. \tag{2.34}$$

Using (2.8) and (2.34) in (2.33) yields

$$\begin{aligned} & F(x) - F(p_{H,\varepsilon}(v)) \\ & \geq f(v) + \langle \nabla f(v), x - v \rangle + g(p_{H,\varepsilon}(v)) + \langle \nu_g(p_{H,\varepsilon}(v)), x - p_{H,\varepsilon}(v) \rangle \\ & \quad - f(v) - \langle \nabla f(v), p_{H,\varepsilon}(v) - v \rangle - \frac{1}{2}\|p_{H,\varepsilon}(v) - v\|_H^2 - g(p_{H,\varepsilon}(v)) - \xi \\ & = -\frac{1}{2}\|p_{H,\varepsilon}(v) - v\|_H^2 + \langle x - p_{H,\varepsilon}(v), \nabla f(v) + \nu_g(p_{H,\varepsilon}(v)) \rangle - \xi \\ & = -\frac{1}{2}\|p_{H,\varepsilon}(v) - v\|_H^2 + \langle x - p_{H,\varepsilon}(v), H(v - p_{H,\varepsilon}(v)) + \zeta \rangle - \xi \\ & = -\frac{1}{2}\|p_{H,\varepsilon}(v) - v\|_H^2 + \langle x - p_{H,\varepsilon}(v), H(v - p_{H,\varepsilon}(v)) \rangle \\ & \quad + \langle v - p_{H,\varepsilon}(v), v - p_{H,\varepsilon}(v) \rangle_H - \langle v - p_{H,\varepsilon}(v), v - p_{H,\varepsilon}(v) \rangle_H + \langle x - p_{H,\varepsilon}(v), \zeta \rangle - \xi \\ & = \frac{1}{2}\|p_{H,\varepsilon}(v) - v\|_H^2 + \langle v - x, p_{H,\varepsilon}(v) - v \rangle_H + \langle x - p_{H,\varepsilon}(v), \zeta \rangle - \xi \end{aligned}$$

□

Corollary 3. *By setting $x = x_*$, $v = x_k$, $p_{H,\varepsilon}(x_k) = x_{k+1}$, $H = H_k$, $\xi = \xi_k$, and $\zeta = \zeta_k$ in the result of Lemma 4, we have*

$$F(x_{k+1}) - F(x_*) \leq -\frac{1}{2}\|x_k - x_{k+1}\|_{H_k}^2 + \langle x_k - x_{k+1}, x_k - x_* \rangle_{H_k} + \langle x_* - x_{k+1}, \zeta_k \rangle - \xi_k. \tag{2.35}$$

With $x = v = x_k$, $p_{H,\varepsilon}(x_k) = x_{k+1}$, *and* $H = H_k$, *we have*

$$F(x_k) - F(x_{k+1}) \geq \frac{1}{2}\|x_{k+1} - x_k\|^2_{H_k} + \langle x_k - x_{k+1}, \zeta_k\rangle - \xi_k. \tag{2.36}$$

Now, under Assumptions 1 and 2, the following theorem shows geometric decrease in function values.

Theorem 3. *At each step of PQNA, the following improvement in the objective function value is guaranteed*

$$\begin{aligned} F(x_{k+1}) - F(x_*) \leq (1 - \frac{1}{2M\gamma_k})(F(x_k) - F(x_*)) \\ + \frac{1}{\gamma_k}\left(\frac{\xi_k}{M} + \left(\frac{\|x_k - x_{k+1}\|}{M} + \frac{mD_*}{2M}\right)\|\zeta_k\|\right), \end{aligned} \tag{2.37}$$

where $\gamma_k = \|x_k - x_*\|/\|x_k - x_{k+1}\|$ *and* $\|x_k - x_*\| \leq D_*$.

Proof. Using inequality (2.35) we have

$$\begin{aligned} F(x_{k+1}) - F(x_*) &\leq M\|x_k - x_{k+1}\|\|x_k - x_*\| - \frac{m}{2}\|x_k - x_{k+1}\|^2 + D_*\|\zeta_k\| \\ &= \|x_k - x_{k+1}\|^2(M\frac{\|x_k - x_*\|}{\|x_k - x_{k+1}\|} - \frac{m}{2}) + D_*\|\zeta_k\| \\ &= \|x_k - x_{k+1}\|^2(M\gamma_k - \frac{m}{2}) + D_*\|\zeta_k\|, \end{aligned}$$

where $\gamma_k = \|x_k - x_*\|/\|x_k - x_{k+1}\|$. Using inequality (2.36)

$$\begin{aligned} F(x_{k+1}) - F(x_*) &\leq (F(x_k) - F(x_{k+1}))(2\frac{M}{m}\gamma_k - 1) + \frac{2}{m}\|x_k - x_{k+1}\|\|\zeta_k\| \\ &\quad + \frac{2}{m}\xi_k + D_*\|\zeta_k\| \\ &\leq [(F(x_k) - F(x_*)) - (F(x_{k+1}) - F(x_*))](2\frac{M}{m}\gamma_k - 1) \\ &\quad + \frac{2}{m}\|x_k - x_{k+1}\|\|\zeta_k\| + \frac{2}{m}\xi_k + D_*\|\zeta_k\|, \end{aligned}$$

which implies

$$\begin{aligned} F(x_{k+1}) - F(x_*) &\leq (1 - \frac{m}{2M\gamma_k})(F(x_k) - F(x_*)) \\ &\quad + \frac{1}{M\gamma_k}\|x_k - x_{k+1}\|\|\zeta_k\| + \frac{\xi_k}{M\gamma_k} + \frac{mD_*}{2M}\frac{\|\zeta_k\|}{\gamma_k} \\ &= (1 - \frac{1}{2M\gamma_k})(F(x_k) - F(x_*)) \\ &\quad + \frac{1}{\gamma_k}\left(\frac{\xi_k}{M} + \left(\frac{\|x_k - x_{k+1}\|}{M} + \frac{mD_*}{2M}\right)\|\zeta_k\|\right). \end{aligned} \tag{2.38}$$

□

Based on Theorem 3, if the quantities γ_k is bounded for all large k, asymptotic Q-linear convergence in function values will be assured, this fact motivates the following theorem.

Theorem 4. *Let Assumptions 1 and 2 hold, then during the application of PQNA, for all k, we have*

$$\gamma_k \leq \frac{M(L+m)}{\mu m} + \frac{M}{m}\frac{\|\zeta_k\|}{\|x_k - x_{k+1}\|} + \frac{M}{m}\frac{\|\zeta_k\|}{\|x_k - x_*\|} \quad \text{and} \tag{2.39a}$$

$$\gamma_k \geq \left(\frac{M(L+m)}{m^2} + \frac{M}{m^2}\frac{\|\zeta_k\|}{\|x_k - x_{k+1}\|} + \frac{M}{m^2}\frac{\|\zeta_k\|}{\|x_k - x_*\|}\right)^{-1}, \tag{2.39b}$$

where $\gamma_k = \|x_k - x_*\|/\|x_k - x_{k+1}\|$.

Proof. Define $prox_g^H(v)$ to be the $\varepsilon-$minimizer of function $h(u) := g(u) + \frac{1}{2}\|u - v\|_H^2$, which implies that for any $x \in \mathbb{R}^n$, there exists a $\nu_h(prox_g^H(v)) \in \partial h(prox_g^H(v))$, such that

$$\nu_h(prox_g^H(v))^T(x - prox_g^H(v)) = \zeta^T(x - prox_g^H(v)). \tag{2.40}$$

Using $\nu_h(prox_g^H(v)) = \nu_g(prox_g^H(v)) + H(prox_g^H(v) - v)$ where $\nu_g(prox_g^H(v)) \in \partial g(prox_g^H(v))$ in the above inequality implies

$$(\nu_g(prox_g^H(v)) + H(prox_g^H(v) - v))^T(x - prox_g^H(v)) = \zeta^T(x - prox_g^H(v)),$$

which implies the following inequality by substituting $x = x_*$, such that

$$\begin{aligned}&\nu_g(prox_g^H(v))^T(x_* - prox_g^H(v)) + (prox_g^H(v) - v)^T H(x_* - prox_g^H(v))\\&= \zeta^T(x_* - prox_g^H(v)).\end{aligned} \tag{2.41}$$

By introducing $\tau = H(x - prox_g^H(x - H^{-1}\nabla f(x)))$, we have $prox_g^H(x - H^{-1}\nabla f(x)) = x - H^{-1}\tau$, and consequently, be setting $v = x - H^{-1}\nabla f(x)$ in (2.41), we will have

$$\begin{aligned}&\nu_g(x - H^{-1}\tau)^T(x_* - x + H^{-1}\tau) + (H^{-1}\nabla f(x) - H^{-1}\tau)^T H(x_* - x + H^{-1}\tau)\\&= \zeta^T(x_* - x + H^{-1}\tau).\end{aligned} \tag{2.42}$$

Now, since x_* is the minimizer of the composite function F, we have

$$(x - H^{-1}\tau - x_*)^T(\nabla f(x_*) + \nu_g(x_*)) = 0. \tag{2.43}$$

Summing up two inequalities (2.42) and (2.43) implies

$$\begin{aligned}&- [(\nu_g(x - H^{-1}\tau) - \nu_g(x_*))^T((x - H^{-1}\tau) - x_*)]\\&+ (x - H^{-1}\tau - x_*)^T(\nabla f(x_*) - (\nabla f(x) - \tau))\\&= \zeta^T(x_* - x + H^{-1}\tau).\end{aligned}$$

Now, by using the monotonicity of the subgradient as described in Lemma 2, in the above equality, we will have

$$(x - H^{-1}\tau - x_*)^T(\nabla f(x_*) - (\nabla f(x) - \tau)) \geq \zeta^T(x_* - x + H^{-1}\tau),$$

or equivalently

$$\begin{aligned}&(x - x_*)^T(\nabla f(x) - \nabla f(x_*))\\&\leq \tau^T(H^{-1}(\nabla f(x) - \nabla f(x_*)) + x - x_*) - \tau^T H^{-1}\tau + \zeta^T(x - x_*) - \zeta^T H^{-1}\tau.\end{aligned} \tag{2.44}$$

By using (2.13a) and (2.12), we have

$$\mu\|x-x_*\|^2 \leq \|\tau\|(\frac{1}{m}\|\nabla f(x)-\nabla f(x_*)\|+\|x-x_*\|)+\|\zeta\|\|x-x_*\|+\frac{1}{m}\|\zeta\|\|\tau\|.$$

By using Lipschitz continuity of gradient ∇f, we will have

$$\mu\|x-x_*\|^2 \leq \|\tau\|(\frac{L}{m}+1)\|x-x_*\|+\|\zeta\|\|x-x_*\|+\frac{1}{m}\|\zeta\|\|\tau\|.$$

Finally, by substituting the definition of τ and defining $x = x_k$ and $\zeta = \zeta_k$, we obtain (2.39a)

$$\gamma_k \leq \frac{M(L+m)}{\mu m}+\frac{M}{m}\frac{\|\zeta_k\|}{\|x_k-x_{k+1}\|}+\frac{M}{m}\frac{\|\zeta_k\|}{\|x_k-x_*\|}.$$

On the other hand, since function f is a convex function, we have $(x-x_*)^T(\nabla f(x)-\nabla f(x_*)) \geq 0$, and consequently, we can rewrite inequality (2.44) as

$$\begin{aligned}
\tau^T(H^{-1}(\nabla f(x)-\nabla f(x_*))+x-x_*)-\tau^T H^{-1}\tau+\zeta^T(x-x_*)-\zeta^T H^{-1}\tau \geq 0\\
\tau^T(H^{-1}(\nabla f(x)-\nabla f(x_*))+x-x_*)+\|\zeta\|\|x-x_*\|+\frac{1}{m}\|\zeta\|\|\tau\| \geq \tau^T H^{-1}\tau\\
\|\tau\|(\frac{1}{m}\|\nabla f(x)-\nabla f(x_*)\|+\|x-x_*\|)+\|\zeta\|\|x-x_*\|+\frac{1}{m}\|\zeta\|\|\tau\| \geq \frac{1}{M}\|\tau\|^2\\
\|\tau\|(\frac{L}{m}+1)\|x-x_*\|+\|\zeta\|\|x-x_*\|+\frac{1}{m}\|\zeta\|\|\tau\| \geq \frac{1}{M}\|\tau\|^2,
\end{aligned}$$

which implies (2.39b) by substituting the definition of τ, and setting $x = x_k$ and $\zeta = \zeta_k$

$$\gamma_k \geq \left(\frac{M(L+m)}{m^2}+\frac{M}{m^2}\frac{\|\zeta_k\|}{\|x_k-x_{k+1}\|}+\frac{M}{m^2}\frac{\|\zeta_k\|}{\|x_k-x_*\|}\right)^{-1}.$$

□

Based on the result of Theorem 4, we can bound γ_k, if we guarantee that $\|\zeta_k\|$ converges to zero faster than the step length $\|x_k - x_{k+1}\|$ as well as distance to optimality $\|x_k - x_*\|$. In this case, by using Theorem 3, one can prove the desired linear convergence rate of inexact *PQNA*. However, the rigorous analysis of this

inexact setting is a subject of a future work.

In what follows, we present our second approach to present the linear convergence rates for both exact and inexact variants of *PQNA*.

2.3.1.3 Inexact Case (Second Approach)

Let us consider Algorithm 3 for which (2.10) holds for some sequence of errors $\epsilon_k \geq 0$. The relaxed sufficient decrease condition

$$F(x_{k+1}) - F(x_k) \leq \eta(Q_{H_k}(x_{k+1}, x_k) - F(x_k)),$$

for a given $\eta \in (0, 1]$, can be written as

$$\begin{aligned} F(x_{k+1}) &\leq Q_{H_k}(x_{k+1}, x_k) - (1-\eta)\left(Q_{H_k}(x_{k+1}, x_k) - F(x_k)\right) \\ &\leq Q_{H_k}(x_{k+1}, x_k) - \frac{1-\eta}{\eta}\left(F(x_{k+1}) - F(x_k)\right). \end{aligned}$$

Thus, at each iteration we have

$$F(x_{k+1}) \leq Q_{H_k}(x_{k+1}, x_k) + \xi_k, \tag{2.45}$$

where the sequence of the errors ξ_k is defined as

$$\xi_k \leq (1 - \frac{1}{\eta})\left(F(x_{k+1}) - F(x_k)\right). \tag{2.46}$$

In particular, setting $\eta = 1$ results in $\xi_k = 0$, for all k and enforces the algorithm to accept only those steps that achieve full (predicted) reduction. However, using $\eta < 1$ allows the algorithm to take steps satisfying only a fraction of the predicted reduction, which may lead to larger steps and faster progress.

Under the above inexact condition, we can show the following result.

Theorem 5. *Suppose that Assumptions 1 and 2 hold. At each iteration of the*

inexact PQNA, stated in Algorithm 3, we have

$$F(x_k) - F(x_*) \leq \rho^k \left(F(x_0) - F(x_*) + A_k\right), \tag{2.47}$$

when $\rho = 1 - (\eta\gamma)/(\gamma + M)$, *and*

$$A_k := \eta \sum_{i=1}^{k} \left(\epsilon_i/\rho^i\right).$$

Proof. Applying (2.45), with $v = x_k$ and consequently $p_{H_k,\epsilon_k}(x_k) = x_{k+1}$, we have

$$\begin{aligned}
F(x_{k+1}) &\leq Q_{H_k}(x_{k+1}, x_k) + \xi_k \\
&= f(x_k) + \langle \nabla f(x_k), x_{k+1} - x_k \rangle + \frac{1}{2}\|x_{k+1} - x_k\|_{H_k}^2 + g(x_{k+1}) + \xi_k \\
&= \min_{u \in \mathbb{R}^n} \; f(x_k) + \langle \nabla f(x_k), u - x_k \rangle + \frac{1}{2}\|u - x_k\|_{H_k}^2 + g(u) + \epsilon_k + \xi_k \\
&\leq \min_{u \in \mathbb{R}^n} \; f(u) + \frac{1}{2}\|u - x_k\|_{H_k}^2 + g(u) + (\epsilon_k + \xi_k) \qquad \text{(convexity of } f\text{)} \\
&= \min_{u \in \mathbb{R}^n} \; F(u) + \frac{1}{2}\|x_k - u\|_{H_k}^2 + (\epsilon_k + \xi_k) \\
&\leq \min_{t \in [0,1]} \; F(tx_* + (1-t)x_k) + \frac{1}{2}\|x_k - tx_* - (1-t)x_k\|_{H_k}^2 \\
&\quad + (\epsilon_k + \xi_k) \\
&\leq \min_{t \in [0,1]} \; tF(x_*) + (1-t)F(x_k) - \frac{1}{2}\gamma t(1-t)\|x_* - x_k\|^2 \\
&\quad + \frac{1}{2}t^2\|x_* - x_k\|_{H_k}^2 + (\epsilon_k + \xi_k) \qquad \text{(using (2.13b))} \\
&\leq \min_{t \in [0,1]} \; tF(x_*) + (1-t)F(x_k) - \frac{1}{2}\gamma t(1-t)\|x_* - x_k\|^2 \\
&\quad + \frac{1}{2}Mt^2\|x_* - x_k\|^2 + (\epsilon_k + \xi_k) \\
&\leq t'F(x_*) + (1-t')F(x_k) + (\epsilon_k + \xi_k). \qquad \left(\text{where } t' = \frac{\gamma}{\gamma + M}\right)
\end{aligned}$$

Therefore, we have

$$F(x_{k+1}) \leq t'F(x_*) + (1-t')F(x_k) + (\epsilon_k + \xi_k),$$

which implies

$$F(x_{k+1}) - F(x_*) \leq (1 - t')(F(x_k) - F(x_*)) + (\epsilon_k + \xi_k).$$

Now, by substituting the expression for ξ_k, as stated in (2.46), we will have

$$F(x_{k+1}) - F(x_*) \leq \rho(F(x_k) - F(x_*)) + \eta\epsilon_k,$$

where $\rho = 1 - \eta t'$. Now, we can conclude the final result as

$$\begin{aligned} F(x_k) - F(x_*) &\leq \rho^k(F(x_0) - F(x_*)) + \sum_{i=1}^{k} \eta\rho^{k-i}\epsilon_i \\ &= \rho^k \left(F(x_0) - F(x_*) + \eta \sum_{i=1}^{k} \left(\epsilon_i / \rho^i \right) \right), \end{aligned}$$

where $\rho = 1 - (\eta\gamma)/(\gamma + M)$. □

Remark 2. *In Theorem 5, by setting $\epsilon_k = 0$ and $\eta = 1$, which implies $\xi_k = 0$, we achieve the linear convergence rate of the exact variant of PQNA.*

Remark 3. *We have shown that in the linear convergence rate of PQNA, the constant is $\rho = 1 - (\eta\gamma)/(\gamma + M)$. As argued in Remark 1, M is of the same order as L in the worst case, hence in that case the linear rate of PQNA is the same as that of the simple PGA. However, it is easy to see that in the proof of Theorem 5, the linear rate is derived using the upper bound on $\|x_* - x_k\|^2_{H_k}$, where H_k is the approximate Hessian on step k. Clearly, the idea of using the partial second-order information is to reduce the worst case bound of H_k in general and consequently on $\|x_* - x_k\|^2_{H_k}$. In particular, obtaining a smaller bound M_k on each iteration yields a larger convergence coefficient $\rho_k = 1 - (\eta\gamma)/(\gamma + M_k)$. While for general H_k, we do not expect to improve upon the regular PGA in theory, this remark serves to explain the better performance of PQNA in practice.*

Based on the result of Theorem 5, it follows that the boundedness of the sequence $\{A_k\}$ is a sufficient condition to achieve the linear convergence rate. Hence, the

required condition on the sequence of errors is $\sum_{i=1}^{k}(\epsilon_i/\rho^i) < \infty$. For all $i \leq k$, suppose that $\epsilon_i \leq C\rho^{i\cdot\delta}$, for some $\delta > 1$ and some $C > 0$. Then, we have $\sum_{i=1}^{k}(\epsilon_i/\rho^i) \leq C\sum_{i=1}^{k}\rho^{i(\delta-1)}$, which is uniformly bounded for all k. Recall that the k-th subproblem $Q_k^* := \min_{u\in\mathbb{R}^n} Q_{H_k}(u, x_k)$ is a strongly convex function with strong convexity parameter at least m–the lower bound on the eigenvalues of H_k. Now, suppose each subproblem is solved via an algorithm with a linear convergence rate for strongly convex problems. In particular, if the subproblem solver is applied for $r(k)$ iterations to the k-th subproblem, we have

$$\left(Q_{H_k}(u_{r(k)}, x_k) - Q_k^*\right) \leq \alpha\left(Q_{H_k}(u_{r(k)-1}, x_k) - Q_k^*\right), \tag{2.48}$$

where $\alpha \in (0, 1)$. Our goal is to ensure that $\epsilon_k \leq C\rho^{k\cdot\delta}$, which can be achieved by applying sufficient number of iterations of the subproblem algorithm. To be specific, the following theorem characterizes this required bound on the number of inner iterations.

Theorem 6. *Suppose that at the k-th iteration of Algorithm 3, after applying the subproblem solver satisfying (2.48) for $r(k)$ iterations, starting with $u_0 = x_k$, we obtain solution $x_{k+1} = u_{r(k)}$.*

Let $r(k)$ satisfy

$$r(k) \geq k\log_{1/\alpha}(1/\rho^{\delta}), \tag{2.49}$$

for some $\delta > 1$, and ρ defined in Theorem 5. Then

$$Q_{H_k}(x_{k+1}, x_k) - Q_k^* \leq C\rho^{k\cdot\delta}$$

holds for all $k \geq 1$, with C being the uniform bound on $Q_{H_k}(x_k, x_k) - Q_k^*$, and the linear convergence of Algorithm 3 is achieved.

Proof. First, assume that at the $k-$th iteration we have applied the subproblem solver for $r(k)$ iterations to minimize strongly convex function Q_{H_k}. Now, by combining $Q_{H_k}(u_0, u_0) - Q_k^* \leq C$ and (2.48), we can conclude the following upper bound,

so that

$$Q_{H_k}(u_{r(k)}, u_0) - Q_k^* \leq \alpha^{r(k)} C.$$

Now, if $\alpha^{r(k)} C \leq \epsilon_k$, we can guarantee that $u_{r(k)}$ is an ϵ_k-solution of the k-th subproblem, so that $Q_{H_k}(u_{r(k)}, u_0) \leq Q_k^* + \epsilon_k$. Now, assuming that ρ is known, we can set the error rate of the k-th iteration as $\epsilon_k \leq C\rho^{k \cdot \delta}$, for a fixed $\delta > 1$. In this case, the number of inner iterations which guarantees the ϵ_k-minimizer will be

$$r(k) \geq k \log_{1/\alpha}(1/\rho^{\delta}).$$

□

Remark 4. *Since subproblems are strongly convex, the required linear convergence rate for the subproblem solver, stated in (2.48), can be guaranteed via some basic first-order algorithms or their accelerated variants. However, one difficulty in obtaining lower bound (2.49) is that it depends on the prior knowledge of ρ and α. Consider the following simple modification of Theorem 6; instead of condition (2.49), consider $r(k)$ satisfying*

$$r(k) \geq k \log_{1/\alpha'}(k/\ell), \tag{2.50}$$

for any given $\ell > 0$ and $\alpha' \in (0, 1)$. Then $\epsilon_k \leq C\,(\ell/k)^k$ implies $\epsilon_k \leq C\rho^{k \cdot \delta}$, for sufficiently large k.

In the next subsection, we extend our analysis to the case of solving subproblems via the randomized coordinate descent, where at each iteration the desired error bound related to ϵ_k is only satisfied in expectation.

2.3.2 Solving Subproblems via Randomized Coordinate Descent

As we mentioned before, in order to achieve linear convergence rate of the inexact PQNA, any simple first-order method (such as PGA) can be applied. However, as

discussed in [48], in the case when $g(x) = \lambda\|x\|_1$ and H_k is sum of a diagonal and a low rank matrix, as in the case of L-BFGS approximations, the *coordinate descent* method is the most efficient approach to solve the strongly convex quadratic subproblems. In this case, each iteration of coordinate descent has complexity of $\mathcal{O}(m)$, where m is the memory size of L-BFGS, which is usually chosen to be less than 20, while each iteration of a proximal gradient method has complexity of $\mathcal{O}(nm)$ and each iteration of a Newton type proximal method has complexity of $\mathcal{O}(nm^2)$. While more iterations of coordinate descent may be required to achieve the same accuracy, it tends to be the most efficient approach. To extend our theory of the previous section and to establish the bound on the number of coordinate descent steps needed to solve each subproblem, we utilize convergence results for the *randomized coordinate descent* [45], as is done in [48].

Algorithm 4 shows the framework of the randomized coordinate descent method, which can be used as a subproblem solver of Algorithm 3 and is identical to the method used in [48]. In Algorithm 4, function Q_H is iteratively minimized over a randomly chosen coordinate, while the other coordinates remain fixed.

Algorithm 4 Randomized Coordinate Descent Algorithm

1: Initialize point $v \in \mathbb{R}^n$ and required number of iterations $r > 0$.
2: Set $u_0 \leftarrow v$.
3: **for** $l = 1, 2, \cdots, r-1$ **do**
4: Choose j uniformly from $\{1, 2, \cdots, n\}$.
5: Compute $z^* := \arg\min_{z \in \mathbb{R}^n} Q_H(u_l + ze_j, v)$.
6: Set $u_{l+1} \leftarrow u_l + z^* e_j$.
7: Return u_r.

In what follows, we restate *Theorem 6* in [45], which establishes linear convergence rate of the randomized coordinate descent algorithm, in expectation, to solve strongly convex problems.

Theorem 7. *Suppose we apply randomized coordinate descent for r iterations, to minimize the m-strongly convex function Q with M-Lipschitz gradient, to obtain the random point u_r.*

When u_0 is the initial point and $Q^* := \min_{u\in\mathbb{R}^n} Q_H(u, u_0)$, for any r, we have

$$\mathbb{E}\left(Q_H(u_r, u_0) - Q^*\right) \leq \left(1 - \frac{1-\phi_{m,M}}{n}\right)^r \left(Q_H(u_0, u_0) - Q^*\right), \tag{2.51}$$

where ϕ_m is defined as

$$\phi_{m,M} = \begin{cases} 1 - m/4M & \text{if } m \leq 2M, \\ M/m & \text{otherwise.} \end{cases} \tag{2.52}$$

Proof. The proof can be found in [45]. □

Now, we want to analyze how we can utilize the result of Theorem 7 to achieve the linear convergence rate of inexact PQNA, in expectation. Toward this end, first we need the following theorem as the probabilistic extension of Theorem 5.

Theorem 8. *Suppose that Assumptions 1 and 2 hold. At each iteration k of the inexact PQNA, stated in Algorithm 3, assume that the error ϵ_k is a nonnegaitve random variable defined on some probability space with an arbitrary distribution. Then, we have*

$$\mathbb{E}\left(F(x_k) - F(x_*)\right) \leq \rho^k \left(F(x_0) - F(x_*) + B_k\right), \tag{2.53}$$

when $\rho = 1 - (\eta\gamma)/(\gamma + M)$, and

$$B_k := \eta \sum_{i=1}^{k} \left(\mathbb{E}(\epsilon_i)/\rho^i\right).$$

Proof. The proof is a trivial modification of that of Theorem 5. □

In what follows, we describe how the randomized coordinate descent method ensures the required accuracy of subproblems and consequently guarantees linear convergence of the inexact PQNA.

Theorem 9. *Suppose that at the k-th iteration of Algorithm 3, after applying Algorithm 4 for $r(k)$ iterations, starting with $u_0 = x_k$, we obtain solution $x_{k+1} = u_{r(k)}$.*

If

$$r(k) \geq k \log_{1/\alpha_n}(k/\ell),$$

where ℓ *is any positive constant,* $\alpha_n = \left(1 - \frac{1-\phi_{m,M}}{n}\right)$ *with* $\phi_{m,M}$ *defined in* (2.52)*, and* C *is the uniform bound on* $Q_{H_k}(x_k, x_k) - Q_k^*$*, then Algorithm 3, converges linearly with constant* $\rho = 1 - (\eta\gamma)/(\gamma + M)$*, in expectation.*

Proof. Suppose that at the k-th iteration of Algorithm 3, we apply $r(k)$ steps of Algorithm 4 to minimize the strongly convex function Q_{H_k}. If $u_{r(k)}$ denotes the resulting random point, when u_0 is the initial point and $Q_k^* := \min_{u \in \mathbb{R}^n} Q_{H_k}(u, u_0)$, then based on Theorem 7 we have

$$\mathbb{E}\left(Q_{H_k}(u_{r(k)}, u_0) - Q_k^*\right) \leq \alpha_n^{r(k)} C, \tag{2.54}$$

where $\alpha_n = \left(1 - \frac{1-\phi_{m,M}}{n}\right)$, with $\phi_{m,M}$ defined in (2.52), $Q_{H_k}(u_0, u_0) - Q_k^*$ is bounded from above by C. Now, based on the result of Theorem 8, if $\mathbb{E}(\epsilon_k) \leq C(\ell/k)^k$ for some given positive constant ℓ, then for sufficiently large k, we can guarantee that B_k is uniformly bounded for all k, and consequently the linear convergence rate of Algorithm 3, in expectation is established. Now, by using (2.54), $E(\epsilon_k) \leq C(\ell/k)^k$ simply follows from

$$r(k) \geq k \log_{1/\alpha_n}(k/\ell).$$

□

Remark 5. *The bound on the number of steps* $r(k) \geq k \log_{1/\alpha_n}(k/\ell)$ *for randomized coordinate descent differs from the bound* $r(k) \geq k \log_{1/\alpha}(k/\ell)$ *on the number of steps of a deterministic linear convergence method, such as PGA by the difference in constants* α *and* α_n*. It can be easily shown that in the worst case* $\alpha_n \approx \alpha/n$*, and hence, the number of coordinate descent steps is around* n *times larger than that of a proximal gradient method. On the other hand, each coordinate descent step is* n *times less expensive and in many practical cases a modest number of iterations of randomized coordinate descent is sufficient. Discussions on this can be found in [45] and [48] as well as in Section 2.5.*

2.4 Accelerated Proximal Quasi-Newton Algorithm

We now turn to an accelerated variant of PQNA. As we described in the introduction section, the algorithm proposed in [26] is a proximal quasi-Newton variant of FISTA, described in Algorithm 2. In [26], the convergence rate of $\mathcal{O}(1/k^2)$ is shown under the condition that the Hessian estimates satisfy $0 \prec H_k \preceq H_{k-1}$, at each iteration. On the other hand, the sequence $\{H_k\}$ is chosen so that the quadratic approximation of f is an over approximation. This leads to an unrealistic setting where two possible contradictory conditions need to be satisfied and as mentioned earlier, this condition contradicts the assumptions of the original APGA, stated in Algorithm 2. We propose a more general version, henceforth referred to as APQNA, which allows a more general sequence of H_k and is based on the relaxed version of FISTA, proposed in [50], which does not impose monotonicity of the step-size parameters. Moreover, our algorithm allows more general Hessian estimates as we explain below.

2.4.1 Algorithm Description

The main framework of APQNA as stated in Algorithm 5 is similar to that of Algorithm 2, where the simple composite quadratic approximation Q_μ was replaced by the scaled version Q_H, as is done in Algorithm 3, using (partial) Hessian information. As in the case of Algorithm 3, we assume that the approximate Hessian H_k is a positive definite matrix such that $mI \preceq H_k \preceq MI$, for some positive constants m and M. As discussed in Remark 1, it is simple to show that this condition can be satisfied for any positive m and for any large enough M. Here, however, we will need additional much stronger assumptions on the sequence $\{H_k\}$. The algorithm, thus, has some additional steps compared to Algorithm 2 and the standard FISTA type proximal quasi-Newton algorithm proposed in [26]. Below, we present Algorithm 5 and discuss the steps of each iteration in detail.

The key requirement imposed by Algorithm 5 on the sequence $\{H_k\}$ is that $\sigma_{k+1}H_{k+1} \preceq \sigma_k H_k$, while $\theta_k := \sigma_k/\sigma_{k+1}$ is used to evaluate the accelerated parameter

Algorithm 5 Accelerated Proximal Quasi-Newton Algorithm

1: Initialize $t_1 = 1, \theta_0 = 1, \sigma_1^0 > 0$, $y_1 = x_{-1} = x_0 \in \mathbb{R}^n$, and positive definite matrix $H_0 \in \mathbb{R}^{n\times n}$, and choose $\beta \in (0,1)$.
2: **for** $k = 1, 2, \cdots$ **do**
3: Define $\sigma_k := \sigma_k^0$.
4: Compute $p_{H_k}(y_k) := \arg\min_{u\in\mathbb{R}^n} Q_{H_k}(u, y_k)$.
5: **while** $F\left(p_{H_k}(y_k)\right) > Q_{H_k}\left(p_{H_k}(y_k), y_k\right)$ **do**
6: Set $H_k \leftarrow \frac{1}{\beta} H_k$.
7: Modify σ_k so that $\sigma_k H_k \preceq \sigma_{k-1} H_{k-1}$.
8: Update $\theta_{k-1} = \sigma_{k-1}/\sigma_k$ and recompute t_k and y_k using (2.55a)-(2.55b).
9: Compute $p_{H_k}(y_k) := \arg\min_{u\in\mathbb{R}^n} Q_{H_k}(u, y_k)$.
10: Set $x_k \leftarrow p_{H_k}(y_k)$.
11: Choose $\sigma_{k+1}^0 > 0$ and H_{k+1} so that $\sigma_{k+1}^0 H_{k+1} \preceq \sigma_k H_k$.
12: Define $\theta_k := \sigma_k/\sigma_{k+1}^0$ and compute t_{k+1} and y_{k+1}, so that

$$t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4\theta_k t_k^2}\right) \tag{2.55a}$$

$$\text{and} \quad y_{k+1} = x_k + \frac{t_k - 1}{t_{k+1}}\left(x_k - x_{k-1}\right). \tag{2.55b}$$

t_{k+1} through (2.55a). During Steps 4 and 5 of iteration k, initial guesses for σ_{k+1}^0 and H_{k+1} are computed and used to define θ_k, which is then used to compute t_{k+1} and y_{k+1}. Since the approximate Hessian H_{k+1} may change during Step 2 of iteration $k+1$, σ_{k+1} may need to change as well in order to satisfy condition $\sigma_{k+1}H_{k+1} \preceq \sigma_k H_k$. In particular, we may shrink the value of σ_{k+1} and consequently will need to recompute θ_k and, thus, t_{k+1} and y_{k+1}. Therefore, the backtracking process in Step 2 of Algorithm 5 involves a loop which may require repeated computations of y_k and hence $\nabla f(y_k)$.

Remark 6. *We do not specify how to compute H_k in Algorithm 5, as long as it satisfies (2.12) and condition $\sigma_{k+1}H_{k+1} \preceq \sigma_k H_k$. Note that Algorithm 5 does not allow the use of exact Hessian information at y_{k+1}, i.e., $H_{k+1} = \nabla^2 f(y_{k+1})$, because it is assumed that H_{k+1} is computed before y_{k+1} (since y_{k+1} uses the value of σ_{k+1}, whose value may have to be dependent on H_{k+1}). However, it is possible to use $H_{k+1} = \nabla^2 f(x_k)$ in Algorithm 5. To use $H_{k+1} = \nabla^2 f(y_{k+1})$, one would need to be*

able to compute σ_{k+1} before H_{k+1} and somehow ensure that condition $\sigma_{k+1}H_{k+1} \preceq \sigma_k H_k$ is satisfied. This condition can eventually be satisfied by applying similar technique to Step 2, but in that case H_{k+1} will not be equal to the Hessian, but to some multiple of the Hessian, i.e., $\frac{1}{\beta^i}\nabla^2 f(y_{k+1})$, for some i.

In our numerical results, we construct H_k via L-BFGS and ignore condition $\sigma_{k+1}H_{k+1} \preceq \sigma_k H_k$, since enforcing it in this case causes a very rapid decrease in σ. It is unclear, however, if a practical version of Algorithm 5, based on L-BFGS Hessian approximation can be derived, which may explain why the accelerated version of our algorithm does not represent any significant advantage.

One trivial choice of the matrix sequence is $H_k = \frac{1}{\mu_k} I$. In this case, the sequence of scalars $\sigma_k = \mu_k$, satisfies $\sigma_{k+1}H_{k+1} \preceq \sigma_k H_k$, for all k. This choice of Hessian reduces Algorithm 5 to the version of APGA with full backtracking of the step-size parameters, proposed in [50], hence Algorithm 5 is the generalization of that algorithm. Another choice for the matrix sequence is $H_k = \frac{1}{\sigma_k} H$, where the matrix H is any fixed positive definite matrix. This setting of H_k automatically satisfies condition $\sigma_{k+1}H_{k+1} \preceq \sigma_k H_k$, and Algorithm 5 reduces to the simplified version stated below in Algorithm 6.

Note that, by the same logic that was used in Remark 1, the number of backtracking steps at each iteration of Algorithm 6 is uniformly bounded. Thus, as long as the fixed approximate Hessian H is positive definite, a Hessian estimate $H_k = \frac{1}{\sigma_k} H$ has positive eigenvalues bounded from above and below. In our implementation, we compute a fixed matrix H by applying L-BFGS for a fixed number of iterations and then apply Algorithm 6.

In the next section, we analyze the convergence properties of Algorithm 5, where the approximate Hessian H_k is produced by some generic unspecified scheme. The motivation is to be able to apply the analysis to popular and efficient Hessian approximation methods, such as L-BFGS. However, in the worst case for general H_k, a positive lower bound for $\{\sigma_k\}$ can not be guaranteed for such a generic scheme. This observation motivates the analysis of Algorithm 6, as a simplified version of Algorithm 5. It remains to be seen if some bound on $\{\sigma_k\}$ may be derived for

Algorithm 6 Accelerated Proximal Quasi-Newton Algorithm with Fixed Hessian

Initialize $t_1 = 1, \theta_0 = 1, \sigma_1^0 > 0$, and $y_1 = x_{-1} = x_0 \in \mathbb{R}^n$, and choose positive definite matrix $H \in \mathbb{R}^{n\times n}$, and $\beta \in (0, 1)$.
for $k = 1, 2, \cdots$ **do**
  Define $\sigma_k := \sigma_k^0$.
  Compute $H_k = (1/\sigma_k)H$ and $p_{H_k}(y_k) := \arg\min_{u\in\mathbb{R}^n} Q_{H_k}(u, y_k)$.
  while $F\left(p_{H_k}(y_k)\right) > Q_{H_k}\left(p_{H_k}(y_k), y_k\right)$ **do**
    Set $\sigma_k \leftarrow \beta\sigma_k$.
    Update θ_{k-1} and recompute t_k and y_k using (2.56a)-(2.56b).
    Update $H_k = (1/\sigma_k)H$.
    Compute $p_{H_k}(y_k) := \arg\min_{u\in\mathbb{R}^n} Q_{H_k}(u, y_k)$.
  Set $x_k \leftarrow p_{H_k}(y_k)$.
  Choose $\sigma_{k+1}^0 > 0$, define $\theta_k := \sigma_k/\sigma_{k+1}^0$, and compute t_{k+1} and y_{k+1}, so that

$$t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4\theta_k t_k^2}\right) \tag{2.56a}$$

$$\text{and} \quad y_{k+1} = x_k + \frac{t_k - 1}{t_{k+1}}\left(x_k - x_{k-1}\right). \tag{2.56b}$$

matrices arising specifically via L-BFGS updates.

2.4.2 Convergence Analysis

In this section, we prove that if the sequence $\{\sigma_k\}$ is bounded away from zero, Algorithm 5 achieves the same rate of convergence as APGA, i.e., $\mathcal{O}(1/k^2)$. First, we state a simple result based on the optimality of p_H.

Lemma 5. *For any $v \in \mathbb{R}^n$, there exists a subgradient of function g where $\nu_g(p_H(v)) \in \partial g(p_H(v))$, such that*

$$\nabla f(v) + H(p_H(v) - v) + \nu_g(p_H(v)) = 0.$$

Proof. The proof is followed immediately from the optimality condition of the convex optimization problem (2.9). □

Now, we can show the following lemma, which bounds the change in the objective

function F and is a simple extension of *Lemma 2.3* in [2].

Lemma 6. *Suppose that for given $v \in \mathbb{R}^n$ and $H \succ 0$, the following condition*

$$F\left(p_H(v)\right) \leq Q_H\left(p_H(v), v\right) \tag{2.57}$$

holds. Then for any $x \in \mathbb{R}^n$

$$F(x) - F(p_H(v)) \geq \frac{1}{2}\|p_H(v) - v\|_H^2 + \langle v - x, p_H(v) - v\rangle_H.$$

Proof. From (2.57), we have

$$F(x) - F(p_H(v)) \geq F(x) - Q_H(p_H(v), v). \tag{2.58}$$

Now, based on the convexity of functions f and g, we have

$$\begin{aligned} f(x) &\geq f(v) + \langle \nabla f(v), x - v\rangle \\ \text{and} \quad g(x) &\geq g(p_H(v)) + \langle \nu_g(p_H(v)), x - p_H(v)\rangle, \end{aligned}$$

where $\nu_g(p_H(v))$ is defined in Lemma 5. Summing the above inequalities yields

$$F(x) \geq f(v) + \langle \nabla f(v), x - v\rangle + g(p_H(v)) + \langle \nu_g(p_H(v)), x - p_H(v)\rangle. \tag{2.59}$$

Using (2.8) and (2.59) in (2.58) yields

$$\begin{aligned}
F(x) - F(p_H(v)) \geq\ & f(v) + \langle \nabla f(v), x - v\rangle + g(p_H(v)) + \langle \nu_g(p_H(v)), x - p_H(v)\rangle \\
& - f(v) - \langle \nabla f(v), p_H(v) - v\rangle - \frac{1}{2}\|p_H(v) - v\|_H^2 - g(p_H(v)) \\
=& -\frac{1}{2}\|p_H(v) - v\|_H^2 + \langle x - p_H(v), \nabla f(v) + \nu_g(p_H(v))\rangle \\
=& -\frac{1}{2}\|p_H(v) - v\|_H^2 + \langle x - p_H(v), H(v - p_H(v))\rangle \\
=& -\frac{1}{2}\|p_H(v) - v\|_H^2 + \langle x - p_H(v), H(v - p_H(v))\rangle \\
& + \langle v - p_H(v), v - p_H(v)\rangle_H - \langle v - p_H(v), v - p_H(v)\rangle_H \\
=& \frac{1}{2}\|p_H(v) - v\|_H^2 + \langle v - x, p_H(v) - v\rangle_H.
\end{aligned}$$

□

The following result is a simple corollary of Lemma 6.

Corollary 4. *Let $v \in \mathbb{R}^n$ and $H \succ 0$ be such that*

$$F(p_H(v)) \leq Q_H(p_H(v), v),$$

then for any $x \in \mathbb{R}^n$

$$\begin{aligned}
2(F(x) - F(p_H(v))) &\geq \|p_H(v) - v\|_H^2 + 2\langle p_H(v) - v, v - x\rangle_H, \\
&= \|p_H(v) - x\|_H^2 - \|v - x\|_H^2.
\end{aligned} \tag{2.60}$$

Proof. The result immediately follows by applying the following equality

$$\|b - a\|^2 + 2(b - a)^T(a - c) = \|b - c\|^2 - \|a - c\|^2 \tag{2.61}$$

to Lemma 6 with

$$a := H^{\frac{1}{2}}v, \quad b := H^{\frac{1}{2}}p_H(v), \quad c := H^{\frac{1}{2}}x.$$

□

The next lemma states the key properties which are used in the convergence analysis.

Lemma 7. *At each iteration of Algorithm 5, the following relations hold*

$$\sigma_k H_k \succeq \sigma_{k+1} H_{k+1} \tag{2.62a}$$

$$\text{and} \quad \sigma_k t_k^2 \geq \sigma_{k+1} t_{k+1}(t_{k+1} - 1). \tag{2.62b}$$

Proof. The proof follows trivially from the conditions in Algorithm 5 and the fact that $\theta_k \leq \sigma_k / \sigma_{k+1}$. □

Now, using this lemma and previous results we derive the key property of the iterations of APQNA.

Lemma 8. *In Algorithm 5, for all $k \geq 1$, we have*

$$2\sigma_k t_k^2 v_k + \sigma_k u_k^T H_k u_k \geq 2\sigma_{k+1} t_{k+1}^2 v_{k+1} + \sigma_{k+1} u_{k+1}^T H_{k+1} u_{k+1},$$

where $v_k = F(x_k) - F(x_*)$ and $u_k = t_k x_k - (t_k - 1)x_{k-1} - x_*$.

Proof. In (2.60), by setting $v = y_{k+1}$, $p_H(v) = x_{k+1}$, $H = H_{k+1}$, and $x = x_k$ and then by multiplying the resulting inequality by $\sigma_{k+1}(t_{k+1} - 1)$, we will have

$$\begin{aligned} & 2\sigma_{k+1}(t_{k+1} - 1)(v_k - v_{k+1}) \\ \geq\ & (t_{k+1} - 1)(x_{k+1} - y_{k+1})^T \sigma_{k+1} H_{k+1}(x_{k+1} - y_{k+1}) \\ & + 2(t_{k+1} - 1)(x_{k+1} - y_{k+1})^T \sigma_{k+1} H_{k+1}(y_{k+1} - x_k). \end{aligned}$$

On the other hand, in (2.60), by setting $x = x_*$ and multiplying it by σ_{k+1}, we have

$$\begin{aligned} -2\sigma_{k+1} v_{k+1} \geq\ & (x_{k+1} - y_{k+1})^T \sigma_{k+1} H_{k+1}(x_{k+1} - y_{k+1}) \\ & + 2(x_{k+1} - y_{k+1})^T \sigma_{k+1} H_{k+1}(y_{k+1} - x_*). \end{aligned}$$

By adding these two inequalities, we have

$$\begin{aligned}
&2\sigma_{k+1}((t_{k+1}-1)v_k - t_{k+1}v_{k+1}) \\
&\geq t_{k+1}(x_{k+1}-y_{k+1})^T\sigma_{k+1}H_{k+1}(x_{k+1}-y_{k+1}) \\
&\quad + 2(x_{k+1}-y_{k+1})^T\sigma_{k+1}H_{k+1}(t_{k+1}y_{k+1} - (t_{k+1}-1)x_k - x_*).
\end{aligned}$$

Multiplying the last inequality by t_{k+1} and applying inequality (2.62b) give

$$\begin{aligned}
&2(\sigma_k t_k^2 v_k - \sigma_{k+1}t_{k+1}^2 v_{k+1}) \\
&\geq t_{k+1}^2(x_{k+1}-y_{k+1})^T\sigma_{k+1}H_{k+1}(x_{k+1}-y_{k+1}) \\
&\quad + 2t_{k+1}(x_{k+1}-y_{k+1})^T\sigma_{k+1}H_{k+1}(t_{k+1}y_{k+1} - (t_{k+1}-1)x_k - x_*).
\end{aligned}$$

By utilizing (2.61) with

$$\begin{aligned}
&a := \sqrt{\sigma_{k+1}}H_{k+1}^{\frac{1}{2}}t_{k+1}y_{k+1}, \quad b := \sqrt{\sigma_{k+1}}H_{k+1}^{\frac{1}{2}}t_{k+1}x_{k+1}, \\
&c := \sqrt{\sigma_{k+1}}H_{k+1}^{\frac{1}{2}}((t_{k+1}-1)x_k + x_*),
\end{aligned}$$

the last inequality can be written as

$$\begin{aligned}
&2(\sigma_k t_k^2 v_k - \sigma_{k+1}t_{k+1}^2 v_{k+1}) \\
&\geq \|\sqrt{\sigma_{k+1}}H_{k+1}^{\frac{1}{2}}t_{k+1}x_{k+1} - \sqrt{\sigma_{k+1}}H_{k+1}^{\frac{1}{2}}((t_{k+1}-1)x_k + x_*)\|^2 \\
&- \|\sqrt{\sigma_{k+1}}H_{k+1}^{\frac{1}{2}}t_{k+1}y_{k+1} - \sqrt{\sigma_{k+1}}H_{k+1}^{\frac{1}{2}}((t_{k+1}-1)x_k + x_*)\|^2.
\end{aligned}$$

Hence, by using the definition of y_{k+1} and u_k, we have

$$2(\sigma_k t_k^2 v_k - \sigma_{k+1}t_{k+1}^2 v_{k+1}) \geq u_{k+1}^T\sigma_{k+1}H_{k+1}u_{k+1} - u_k^T\sigma_{k+1}H_{k+1}u_k.$$

Now, based on (2.62a), we have

$$u_k^T\sigma_k H_k u_k \geq u_k^T\sigma_{k+1}H_{k+1}u_k,$$

which implies

$$2(\sigma_k t_k^2 v_k - \sigma_{k+1} t_{k+1}^2 v_{k+1}) \geq u_{k+1}^T \sigma_{k+1} H_{k+1} u_{k+1} - u_k^T \sigma_k H_k u_k.$$

□

Now, we are ready to state and prove the convergence rate result.

Theorem 10. *The sequence of iterates x_k, generated by Algorithm 5, satisfies*

$$F(x_k) - F(x_*) \leq \frac{\|x_0 - x_*\|^2}{2\sigma_k t_k^2}.$$

Proof. By setting $t_1 = 1$, using the definition of u_k at $k = 1$, which is $u_1 = x_1 - x_*$, and also considering the positive definiteness of H_k for all $k \geq 1$, it follows from Lemma 8 that

$$2\sigma_k t_k^2 v_k \leq 2\sigma_k t_k^2 v_k + \sigma_k u_k^T H_k u_k \leq 2\sigma_1 t_1^2 v_1 + (x_1 - x_*)^T \sigma_1 H_1 (x_1 - x_*). \tag{2.63}$$

Setting $x = x_*$, $v = y_1 = x_0$, $p_H(v) = x_1$, $t_1 = 1$, and $H = H_1$ in (2.60) implies

$$-2v_1 \geq (x_1 - x_*)^T H_1 (x_1 - x_*) - (x_0 - x_*)^T H_1 (x_0 - x_*).$$

Multiplying the above inequality by σ_1 gives

$$2\sigma_1 v_1 + (x_1 - x_*)^T \sigma_1 H_1 (x_1 - x_*) \leq (x_0 - x_*)^T \sigma_1 H_1 (x_0 - x_*).$$

By using inequality (2.63), we have

$$2\sigma_k t_k^2 v_k \leq (x_0 - x_*)^T \sigma_1 H_1 (x_0 - x_*).$$

Finally, by setting $\sigma_1 = 1$ and $H_1 = I$, we obtain

$$v_k \leq \frac{\|x_0 - x_*\|^2}{2\sigma_k t_k^2},$$

which completes the proof. □

Now, based on the result of Theorem 10, in order to obtain the rate of convergence of $\mathcal{O}(1/k^2)$ for Algorithm 5, it is sufficient to show that

$$\sigma_k t_k^2 \geq \psi k^2,$$

for some constant $\psi > 0$. The next result is a simple consequence of the relation (2.62b), or equivalently (2.55a).

Lemma 9. *The sequence $\{\sigma_k\}$ generated by Algorithm 5 satisfies*

$$\sigma_k t_k^2 \geq \left(\frac{\sum_{i=1}^{k} \sqrt{\sigma_i}}{2} \right)^2.$$

Proof. We can prove this lemma by using induction. Trivially, for $k = 1$, since $t_1 = 1$, the inequality holds. As the induction assumption, assume that for $k > 1$, we have $\sigma_k t_k^2 \geq \left(\frac{\sum_{i=1}^{k} \sqrt{\sigma_i}}{2} \right)^2$. Since (2.55a) holds for all k, it follows that

$$t_{k+1} = \frac{1}{2} + \sqrt{\frac{1}{4} + (\frac{\sigma_k}{\sigma_{k+1}}) t_k^2} \geq \frac{1}{2} + \sqrt{\frac{\sigma_k}{\sigma_{k+1}}} t_k.$$

Multiplying by $\sqrt{\sigma_{k+1}}$ implies

$$\sqrt{\sigma_{k+1}} t_{k+1} \geq \frac{\sqrt{\sigma_{k+1}}}{2} + \sqrt{\sigma_k} t_k.$$

Finally, by using induction assumption, we will have

$$\sqrt{\sigma_{k+1}} t_{k+1} \geq \frac{\sqrt{\sigma_{k+1}}}{2} + \frac{\sum_{i=1}^{k} \sqrt{\sigma_i}}{2} = \frac{\sum_{i=1}^{k+1} \sqrt{\sigma_i}}{2}.$$

□

Hence, if we assume that the sequence $\{\sigma_k\}$ is bounded below by a positive constant $\underline{\sigma}$, i.e., $\sigma_k \geq \underline{\sigma}$, we can establish the desired bound on $\sigma_k t_k^2$, as stated in the following theorem.

Theorem 11. *If for all iterations of Algorithm 5 we have $\sigma_k \geq \underline{\sigma}$, then for all $k \geq 1$,*

$$F(x_k) - F(x_*) \leq \frac{2\|x_0 - x_*\|^2}{\underline{\sigma} k^2}. \tag{2.64}$$

Proof. Under the assumption $\sigma_k \geq \underline{\sigma}$, we will have

$$\left(\frac{\sum_{i=1}^{k} \sqrt{\sigma_i}}{2} \right)^2 \geq \frac{k^2 \underline{\sigma}}{4}$$

and consequently, by using Lemma 9, we obtain

$$\sigma_k t_k^2 \geq \frac{k^2 \underline{\sigma}}{4}.$$

Then, by using Theorem 10, we have the desired rate of convergence of $\mathcal{O}(1/k^2)$ as stated in (2.64). □

The assumption of the existence of a bounded sequence $\{\sigma_k\}$ such that $\sigma_k \geq \underline{\sigma}$ and (2.62b) holds may not be satisfied when we use a general approximate Hessian. To illustrate this, consider the following simple sequence of matrices:

$$H_{2k} = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad H_{2k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix}.$$

Clearly, $\sigma_{2k+1} \leq \sigma_{2k}/10$ and $\sigma_{2k} \leq \sigma_{2k-1}/10$, and hence $\sigma_k \leq 10^{-k}$. In this case, based on the result of Theorem 10, we cannot guarantee any convergence result. Some convergence result can still be attained, when $\sigma_k \to 0$, for example, if $\sigma_k \geq \underline{\sigma}/k$, as we show in the following relaxed version of Theorem 11.

Theorem 12. *If for all iterations of Algorithm 5 we have $\sigma_k \geq \underline{\sigma}/k$, then for all $k \geq 1$,*

$$F(x_k) - F(x_*) \leq \frac{2\|x_0 - x_*\|^2}{\underline{\sigma} k}. \tag{2.65}$$

Proof. From $\sigma_k \geq \underline{\sigma}/k$, we will have

$$\left(\frac{\sum_{i=1}^{k} \sqrt{\sigma_i}}{2} \right)^2 \geq \frac{k \underline{\sigma}}{4}$$

and consequently, by using Lemma 9, we obtain

$$\sigma_k t_k^2 \geq \frac{k\sigma}{4}.$$

Then, by using Theorem 10, we have (2.65). □

The above theorem shows that if σ_k converges to zero, but not faster than $1/k$, then our APQNA method may loose its accelerated rate of convergence, but still converges at least at the same rate as PQNA. Establishing lower bounds of σ_k for different choices of Hessian estimates is a nontrivial task and is a path for the future research. As we will demonstrate in our computational section, APQNA with L-BFGS Hessian approximation does not seem to have any practical advantage over its nonaccelerated counterpart, however it is clearly convergent.

We can establish the accelerated rate of Algorithm 6, since in this case we can guarantee a lower bound on σ_k, due to the restricted nature of H_k matrices.

Lemma 10. *In Algorithm 6, let $mI \preceq H$, then $\sigma_k \geq \beta m/L$ and hence the convergence rate of $\mathcal{O}(1/k^2)$ is achieved.*

Proof. In Algorithm 6, we define $H_k = \frac{1}{\sigma_k} H$. The sufficient decrease condition $F(p_{H_k}(y_k)) \leq Q_{H_k}(p_{H_k}(y_k), y_k)$, is satisfied for any $H_k \succeq LI$, hence it is satisfied for any $H_k = \frac{1}{\sigma_k} H$ with $\sigma_k \leq m/L$. By the mechanism of Step 3 in Algorithm 6, we observe that for all k, we have $\sigma_k \geq \beta m/L$. Let us note now that Algorithm 6 is a special case of Algorithm 5, hence all the above results, in particular Theorem 10 and Lemma 9 hold. Consequently, based on Theorem 11, the desired convergence rate of $\mathcal{O}(1/k^2)$ for Algorithm 6 is obtained. □

Remark 7. *We have studied only the exact variant of APQNA in this section. Incorporating inexact subproblem solutions, as was done for APQNA in the previous section, is relatively straightforward following the techniques for inexact APGA, [52]. It is easy to show that if the exact algorithm has the accelerated convergence rate, then the inexact counterpart, with subproblems solved by a linearly convergent*

method, such as randomized coordinate descent, inherits this convergence rate. However, using the relaxed sufficient decrease condition does not apply here as it does not preserve the accelerated convergence rate.

In the next section, we present the numerical results comparing the performance of Algorithm 5 and Algorithm 6 to their nonaccelerated counterparts, to see how much practical acceleration is achieved.

2.5 Numerical Experiments

In this section, we investigate the practical performance of several algorithms discussed in this work, applied to the sparse logistic regression problem

$$\min_w \{F(w) := \frac{1}{N}\sum_{i=1}^{N} \log(1+\exp(-y_i \cdot w^T x_i)) + \lambda\|w\|_1,\ w \in \mathbb{R}^n\},$$

where $f(w) = \frac{1}{N}\sum_{i=1}^{N} \log(1+\exp(-y_i \cdot w^T x_i))$ is the average logistic loss function and $g(w) = \lambda\|w\|_1$, with $\lambda > 0$, is the ℓ_1-regularization function. The input data for this problem is a set of m training data points, $x_i \in \mathbb{R}^n$, and corresponding labels $y_i \in \{-1, +1\}$, for $i = 1, 2, \ldots, N$.

The algorithms that we compare here are as follows:

- Accelerated Proximal Gradient Algorithm (APGA), proposed in [2], (also known as FISTA),
- Proximal Quasi-Newton Algorithm with Fixed Hessian approximation (PQNA-FH),
- Accelerated Proximal Quasi-Newton Algorithm with Fixed Hessian approximation (APQNA-FH),
- Proximal Quasi-Newton Algorithm with L-BFGS Hessian approximation (PQNA-LBFGS), proposed in [48], and

- Accelerated Proximal Quasi-Newton Algorithm with L-BFGS Hessian approximation (APQNA-LBFGS).

In PQNA-FH and APQNA-FH, we set $H_k = \frac{1}{\sigma_k} H$, where H is a positive definite matrix computed via applying L-BFGS updates over the first few iterations of the algorithm which then is fixed for all remaining iterations. On the other hand, PQNA-LBFGS and APQNA-LBFGS employ the L-BFGS updates to compute Hessian estimates throughout the algorithm. In all of the above algorithms, we use the coordinate descent scheme, as described in [48], to solve the subproblems inexactly. According to the theory in [48], PQNA-FH and PQNA-LBFGS converge at the rate of $\mathcal{O}(1/k)$. If f is strongly convex (which depends on the problem data), then according to Theorem 5, PQNA-FH and PQNA-LBFGS converge at a linear rate. By Lemma 10, in APQNA-FH, condition $\sigma_k H_k \preceq \sigma_{k-1} H_{k-1}$ holds automatically and the algorithm converges at the rate of $\mathcal{O}(1/k^2)$. On the other hand, for APQNA-LBFGS, condition $\sigma_k H_k \preceq \sigma_{k-1} H_{k-1}$ has to be enforced. We have tested various implementations that ensure this condition and none have produced a practical approach. We then chose to set $\theta_k = 1$ and relax the condition $\sigma_k H_k \preceq \sigma_{k-1} H_{k-1}$. The resulting algorithm is practical and is empirically convergent but, as we will see, does not provide an improvement over PQNA-LBFGS.

Throughout all of our experiments, we initialize the algorithms with $w_0 = 0$ and we set the regularization parameter $\lambda = 10^{-3}$. Each algorithm terminates whenever $\|(\partial F(x_k))_{\min}\|_\infty \leq 10^{-5} \|\, (\partial F(x_0))_{\min} \,\|_\infty$. In terms of the stopping criteria of subproblems solver at i-th iteration, we performed the coordinate descent method for $r(i)$ steps, so that $r(i) > \min(10^3, i/3)$, as long as the generated step is longer than 10^{-16}. In APQNA-FH and PQNA-FH, in order to construct the fixed matrix H, we apply the L-BFGS scheme by using the information from the first $\bar{k}$ (with $\bar{k}$ chosen between 1 and 10) iterations and then use that fixed matrix through the rest of the algorithm. Finally, to construct the sequence $\{\sigma_k\}$, we set $\sigma_0 = 1$ and $\sigma_{k+1}^0 = 1.015\sigma_k$. The information on the data sets used in our tests is summarized in Table 2.1. These data sets are available through UCI machine learning repository

[2].

Table 2.1: Data information, dimension (d) and number of data points (N).

Instance	d	N	Description
a9a	123	32561	census income dataset
mnist	782	100000	handwritten digit recognition
connect-4	126	10000	win versus loss recognition
HAPT	561	7767	human activities and postural transitions recognition

The algorithms are implemented in MATLAB R2014b and computations were performed on the COR@L computational cluster of the ISE department at Lehigh University, consisting of 16-cores AMD Operation, 2.0 GHz nodes with 32 Gb of memory.

First, in order to demonstrate the effect of using even limited Hessian information within an accelerated method, we compared the performance of APQNA-FH and APGA, both in terms of the number of iterations and the total solution time, see the results in Table 2.2.

[2] http://archive.ics.uci.edu/ml/

Table 2.2: APQNA-FH vs. APGA in terms of function value (Fval), number of iterations (iter) and total solution time (time) in seconds.

a9a

Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APGA	40	3.4891e-01	80	3.4730e-01	862	3.4703e-01	1.95e+01
APQNA-FH	40	3.4706e-01	80	3.4703e-01	121	3.4703e-01	5.52e+00

mnist

Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APGA	48	9.1506e-02	96	9.0206e-02	1202	8.9695e-02	5.13e+02
APQNA-FH	48	8.9754e-02	96	8.9699e-02	144	8.9695e-02	9.91e+01

connect-4

Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APGA	92	3.8284e-01	184	3.7777e-01	3045	3.7682e-01	4.65e+01
APQNA-FH	92	3.7701e-01	184	3.7683e-01	278	3.7682e-01	2.05e+01

HAPT

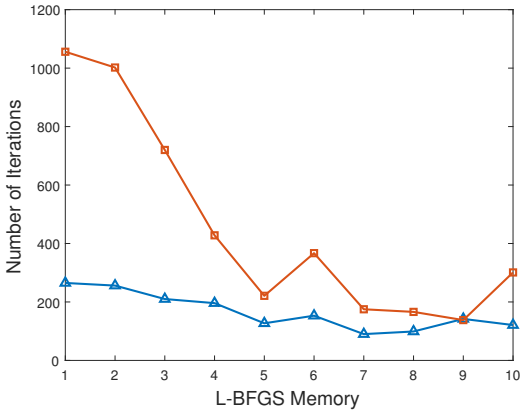
Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APGA	222	8.5415e-02	444	7.7179e-02	13293	7.1511e-02	1.38e+03
APQNA-FH	222	7.2208e-02	444	7.1524e-02	677	7.1511e-02	1.53e+02

Based on the results shown in Table 2.2, we conclude that APQNA-FH consistently dominates the APGA, both in terms of the number of function evaluations and also in terms of the total solution time.

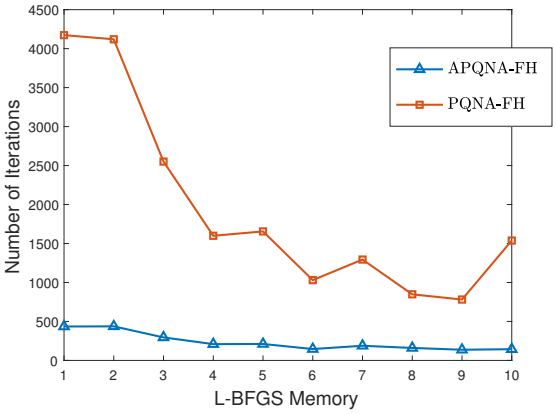
It is worth mentioning that although in terms of computational effort, each iteration of APGA is cheaper than each iteration of APQNA-FH, the total solution time of APQNA-FH is significantly less than APGA, due to the smaller number of iterations of APQNA-FH compared to APGA.

The next experiment is to compare the performance of APQNA-FH and PQNA-FH to observe the effect of acceleration in the fixed matrix setting. This comparison is done in terms of the number of iterations and the number of function evaluations,

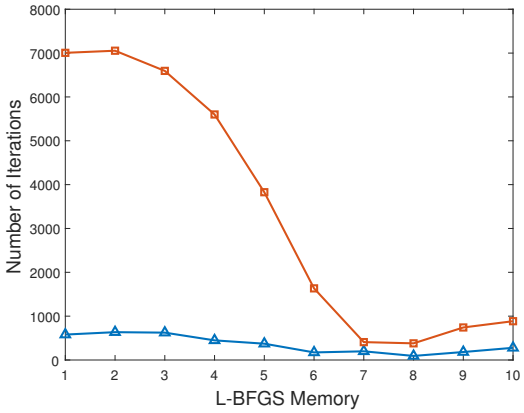
and is shown in Figure 2.1 and Figure 2.2, respectively. The subproblem solution time is the same for both algorithms. As we can see in Figure 2.1, in terms of the number of iterations, APQNA-FH dominates PQNA-FH, for a range of memory sizes of L-BFGS which have been used to compute matrix H. Moreover, as is seen in Figure 2.2, APQNA-FH dominates PQNA-FH, in terms of the number of function evaluations, even though each iteration of APQNA-FH requires two function evaluations, because of the nature of the accelerated scheme. This shows that APQNA-FH achieves practical acceleration compared to PQNA-FH, as supported by the theory in the previous section.



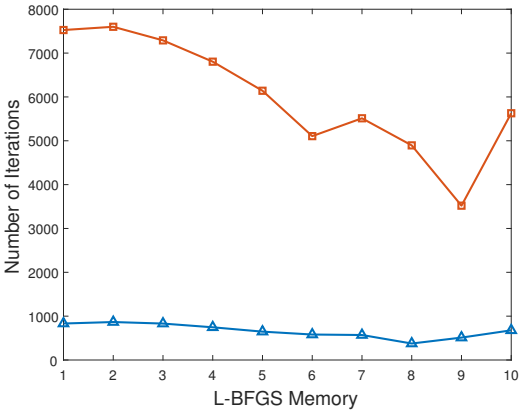
(a) a9a (d=123, m=32561)



(b) mnist (d=782, m=100000)

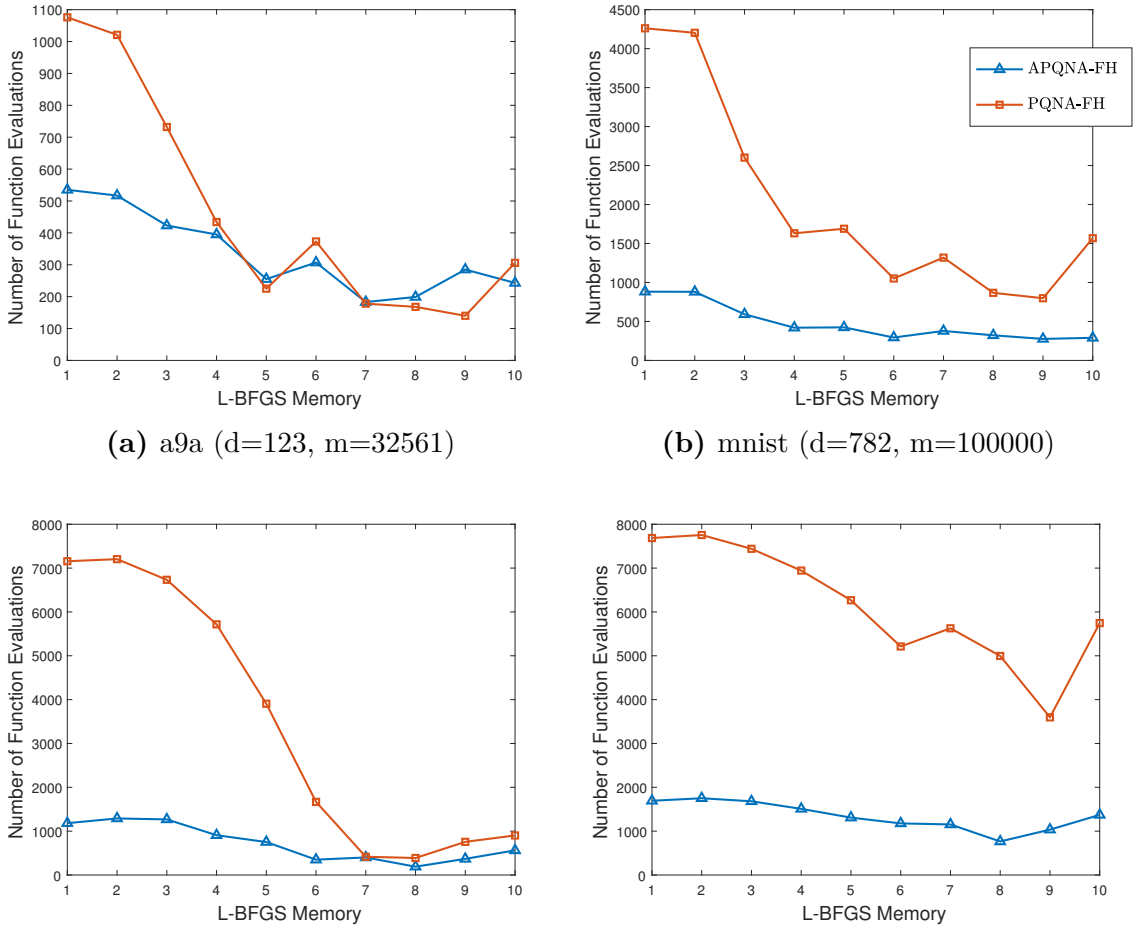


(c) connect-4 (d=126, m=10000)



(d) HAPT (d=561, m=7767)

Figure 2.1: APQNA-FH vs. PQNA-FH in terms of number of iterations.



(a) a9a (d=123, m=32561)

(b) mnist (d=782, m=100000)

(c) connect-4 (d=126, m=10000)

(d) HAPT (d=561, m=7767)

Figure 2.2: APQNA-FH vs. PQNA-FH in terms of number of function evaluations.

Next, we compare the performance of APQNA-FH versus APQNA-LBFGS to compare the effect of using the fixed approximate Hessian $H_k = \frac{1}{\sigma_k} H$, which satisfies condition $\sigma_k H_k \preceq \sigma_{k-1} H_{k-1}$ versus using variable Hessian estimates computed via L-BFGS method at each iteration, while relaxing condition $\sigma_k H_k \preceq \sigma_{k-1} H_{k-1}$. Table 2.3 shows the results of this comparison, obtained based on the best choices of memory size for L-BFGS, in particular $\bar{k} = 8$ and $\bar{k} = 9$, respectively. As we can see, these two algorithms are competitive both in terms of the number of iterations and

also the total solution time. Since APQNA-FH does not use the local information of function f to approximate H_k, it often takes more iterations than APQNA-LBFGS, which constantly updates H_k matrices. On the other hand, since APQNA-FH does not require additional computational effort to evaluate H_k, hence one iteration of this algorithm is cheaper than one iteration of APQNA-LBFGS, which causes the competitive total solution time.

Table 2.3: APQNA-FH vs. APQNA-LBFGS in terms of function value (Fval), number of iterations (iter) and total solution time (time) in seconds.

a9a

Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APQNA-LBFGS	20	3.4760e-01	40	3.4703e-01	64	3.4703e-01	2.83e+00
APQNA-FH	20	3.4763e-01	40	3.4704e-01	99	3.4703e-01	4.33e+00

mnist

Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APQNA-LBFGS	50	8.9713e-02	100	8.9695e-02	148	8.9695e-02	1.04e+02
APQNA-FH	50	8.9797e-02	100	8.9698e-02	160	8.9695e-02	1.15e+02

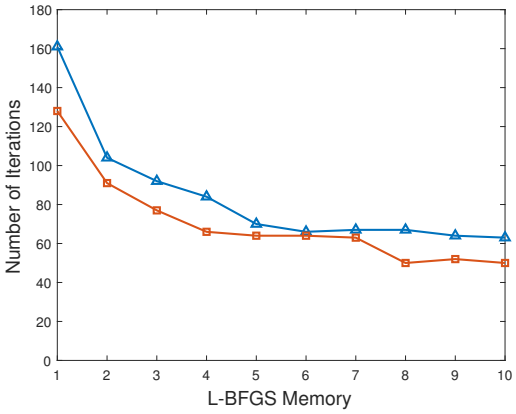
connect-4

Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APQNA-LBFGS	30	3.7769e-01	60	3.7688e-01	144	3.7682e-01	8.35e+00
APQNA-FH	30	3.7689e-01	60	3.7682e-01	93	3.7682e-01	3.95e+00

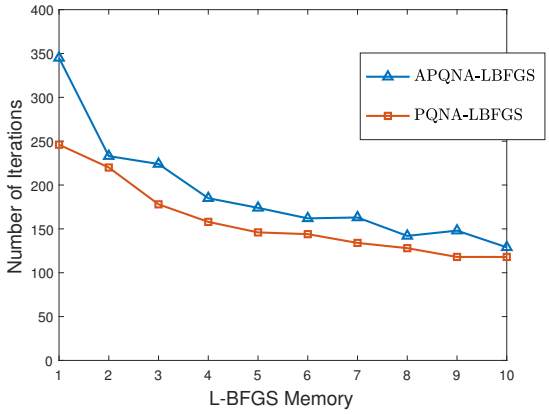
HAPT

Algorithm	iter	Fval	iter	Fval	iter	Fval	time
APQNA-LBFGS	120	7.1860e-02	240	7.1519e-02	356	7.1511e-02	1.04e+02
APQNA-FH	120	7.2134e-02	240	7.1523e-02	376	7.1511e-02	6.89e+01

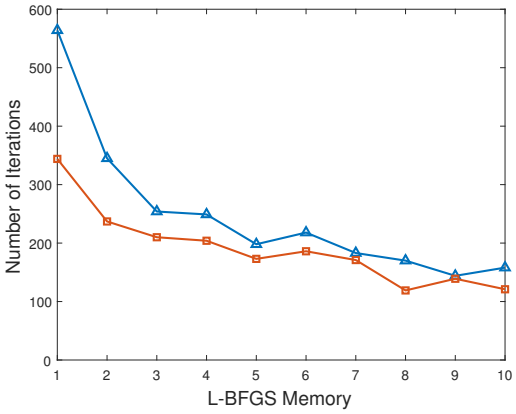
Finally, we compare APQNA-LBFGS and PQNA-LBFGS, to demonstrate the effect of using an accelerated scheme in the quasi-Newton type proximal algorithms. The results of this comparison are shown in Figure 2.3 and Figure 2.4 in terms of the number of iterations and the number of function evaluations, respectively, for different memory sizes of L-BFGS Hessian approximation.



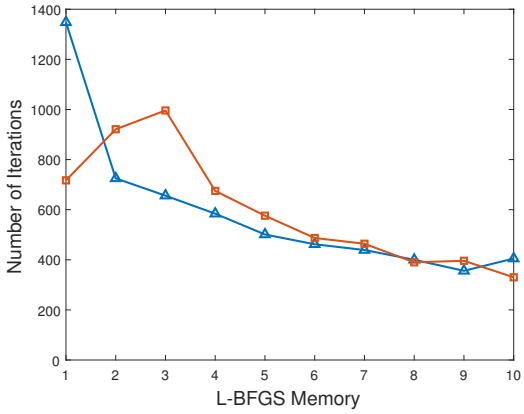
(a) a9a (d=123, m=32561)



(b) mnist (d=782, m=100000)

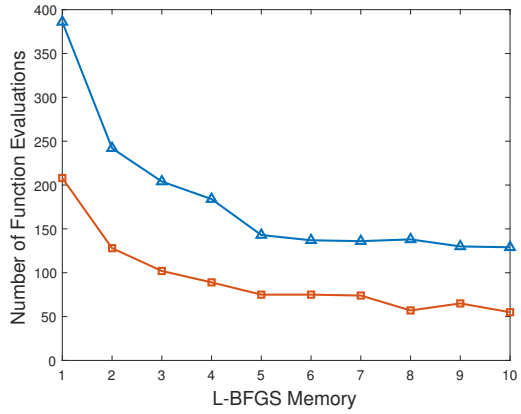


(c) connect-4 (d=126, m=10000)

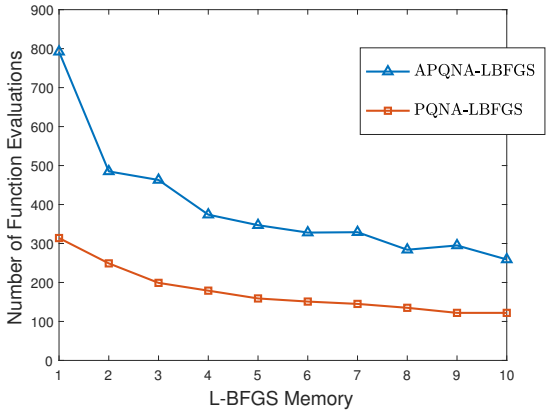


(d) HAPT (d=561, m=7767)

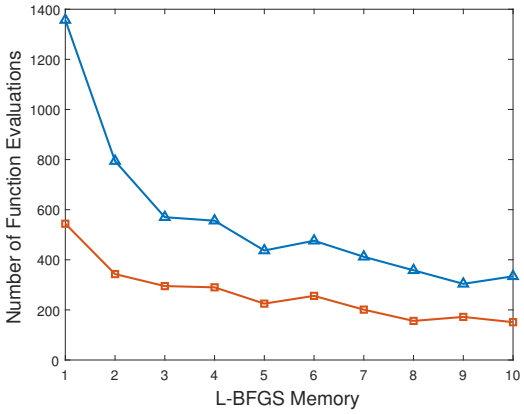
Figure 2.3: APQNA-LBFGS vs. PQNA-LBFGS in terms of number of iterations.



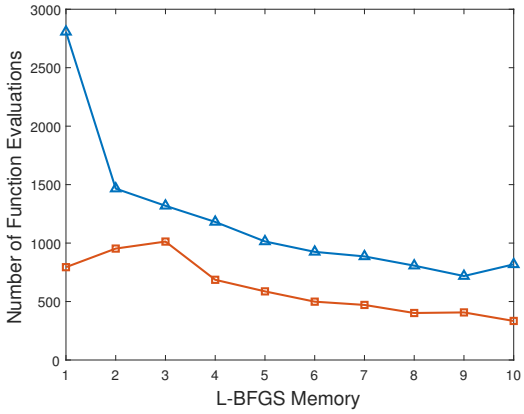
(a) a9a (d=123, m=32561)



(b) mnist (d=782, m=100000)



(c) connect-4 (d=126, m=10000)



(d) HAPT (d=561, m=7767)

Figure 2.4: APQNA-LBFGS vs. PQNA-LBFGS in terms of number of function evaluations.

Clearly, as we can see in Figure 2.3 and 2.4, not only the accelerated scheme does not achieve practical acceleration compared to PQNA-LBFGS in terms of the number of iterations, but it is also inferior in terms of the number of function evaluations, since every iteration requires two function evaluations. Thus, we believe that the practical experiments support our theoretical analysis in that applying acceleration scheme in the case of variable Hessian estimates may not result in a faster algorithm.

2.6 Conclusion

In this work, we established a linear convergence rate of PQNA proposed in [48] under the strong convexity assumption. To our knowledge, this is the first such result, for proximal quasi-Newton type methods, which have lately been popular in the literature. We also show that this convergence rate is preserved when subproblems are solved inexactly. We provide a simple and practical rule for the number of inner iterations which guarantees sufficient accuracy of subproblem solutions. Moreover, we allow a relaxed sufficient decrease condition during backtracking, which preserves the convergence rate, while it is known to improve the practical performance of the algorithm.

Furthermore, we presented a variant of APQNA as an extension of PQNA. We have shown that this algorithm has the convergence rate of $\mathcal{O}(1/k^2)$ under a strong condition on the Hessian estimates, which can not always be guaranteed in practice. We have shown that this condition holds when Hessian estimates are a multiple of a fixed matrix, which is computationally less expensive than the more common methods, such as L-BFGS scheme. Although, this proposed algorithm has the same rate of convergence as the classic APGA, it is significantly faster in terms of the final number of iterations and also the total solution time. Based on the theory, using L-BFGS Hessian approximation, may result in worse convergence rate, however, our computational results show that the practical performance is about the same as that while the fixed matrix. On the other hand, although in these two algorithms, we are applying the accelerated scheme, their practical performances are inferior to that of PQNA-LBFGS, which does not use any accelerated scheme and potentially has a slower sublinear rate of convergence in the absence of strong convexity. We conclude that using variable Hessian estimates is the most efficient approach which will result in the linear convergence rate in the presence of strong convexity, however, a standard accelerated scheme is not useful in this setting. Exploring other possibly more effective accelerated schemes for the proximal quasi-Newton methods is a subject of the future research.

Chapter 3

Black-Box Optimization in Machine Learning

3.1 Introduction

Many machine learning (ML) models rely on optimization tools to perform training. Typically these models are formed so that at least stochastic estimates of the gradient can be computed; for example, when optimizing least squares or logistic loss of a neural network on a given data set. Lately, however, with the increasing need to tune hyperparameters of ML models, black-box optimization methods have been given significant consideration. These methods do not rely on any explicit gradient computation, but assume that only function values can be computed, usually with noise.

There are two, relatively independent, directions of research for black-box optimization, Bayesian Optimization (BO) [6, 34], predominantly popular in the ML community, and derivative free optimization (DFO) [13], popular in the optimization community. There are other classes of methods for black-box optimization developed in the fields of simulation optimization and engineering [1, 21, 28], but they are more specialized and we will not focus on them here.

Both BO and DFO methods are usually applied to functions that are not known

to be convex. The key difference between the BO and DFO methods, is that BO methods always contain a component that aims at the exploration of the space, hence seeking a global solution, while DFO methods are content with a local optimum. However, it has been shown in DFO literature [35] that DFO methods tend to escape local minima and are quite well suited for problems with a few well defined local basins (and possibly many small local basins that appear due to noise).

BO, until recently, has been established as the method of choice for hyperparameter optimization (HPO). While BO methods have been shown to be effective at finding good solutions (not always globally optimal, as that can only be achieved in the limit), their efficiency slows down significantly as the number of iterations grows. Overall, the methods are quite computationally costly and scale poorly with the number of hyperparameters. Recently, BO efficiency has been called into question in comparison with a simple random search [31], whose iterations require nothing but function evaluations. Moreover, some improvements on random search have been proposed to incorporate cheaper function evaluations and further increase its efficiency for HPO.

In this chapter, we will explore properties of an efficient class of DFO methods–model-based trust region methods–in application to problems in ML. We will show that these methods can be more efficient than BO and random search, especially for problems of dimensions higher than two or three. In the specific case of HPO, hyperparameters can be continuous, discrete or categorical. While some DFO methods have been developed for the case of optimization over categorical or binary variables, these methods essentially rely on local search heuristics and we do not consider them here. Our goal is to examine, in detail, the behavior of various black-box methods in a purely continuous setting. We also aim to explore practical scalability of the methods with respect to the dimension of the search space and nonlinearity of the function. While we will list some experiments on HPO problems, these problems are limited to three continuous hyperparameters. Hence, to perform our comparison on problems of larger dimension, we mainly focus on a different problem–optimizing Area Under Receiver Operating Characteristic (ROC) Curve (AUC) [22], over a set of linear classifiers.

This chapter is organized as follows. In the next section, we describe the framework of Trust Region Based Derivative Free Optimization (DFO-TR). In Section 3.3, we compare DFO-TR versus BO. We present numerical results in Section 3.4. Finally, we state the conclusions in Section 3.5.

3.2 Algorithmic Framework of DFO-TR

Model-based trust region DFO methods [11, 44] have been proposed for a class of optimization problems of the form

$$\min_{w \in \mathbb{R}^d} f(w),$$

when computing the gradient and the Hessian of $f(w)$ is not possible, either because it is unknown or because the computable gradient is too noisy to be of use. It is, however, assumed that some local first-order or even second-order information of the objective function is possible to construct to an accuracy sufficient for optimization. If the function is smooth, then such information is usually constructed by building an interpolation or regression model of $f(w)$ using a set of points for which function value is, approximately, known [13]. By using quadratic models, these methods are capable of approximating the second-order information efficiently to speed up convergence and to guarantee convergence to local minima, rather than simply local stationary points. They have been shown to be the most practical black-box optimization methods in deterministic settings [35]. Extensive convergence analysis of these methods over smooth deterministic functions have been summarized in [13].

Recently, several variants of trust region methods have been proposed to solve stochastic optimization problem

$$\min_{w \in \mathbb{R}^d} \mathbb{E}_\xi \left[f(w, \xi) \right],$$

where $f(w, \xi)$ is a stochastic function of a deterministic vector $w \in \mathbb{R}^d$ and a random variable ξ [4, 10, 55]. In particular, in [10], a trust region based stochastic method,

referred to STORM (STochastic Optimization with Random Models), is introduced and shown to converge, almost surely, to a stationary point of $\mathbb{E}_\xi[f(w,\xi)]$, under the assumption that $\mathbb{E}_\xi[f(w,\xi)]$ is smooth. Moreover, in recent work [5], a convergence rate of this method has been analyzed.

This class of stochastic methods utilizes samples of $f(w,\xi)$ to construct models that approximate $\mathbb{E}_\xi[f(w,\xi)]$, sufficiently accurately, with high enough probability.

In Algorithm 7, we present the specific practical implementation of a deterministic algorithm, which can work with finite training sets rather than infinite distributions, but shares many properties with STORM and produces very good results in practice. The key difference between STORM and DFO-TR is that the former requires resampling $f(w,\xi)$ for various w's, at each iteration, since $f(w,\xi)$ is a random value for any fixed w, while DFO-TR computes only one value of deterministic $f(w)$ per iteration. When applied to deterministic smooth functions, this algorithm converges to a local solution [13], but here we apply it to a nonsmooth function which can be viewed as a noisy version of a smooth function (as argued in the next section). While there are no convergence results for DFO-TR or STORM for deterministic, nonsmooth, noisy functions, the existing results indicate that the DFO-TR method will converge to a neighborhood of the solution before the noise in the function estimates prevents further progress. Our computational results confirm this.

We note a few key properties on the algorithm. At each iteration, a quadratic model, not necessarily convex, is constructed using previously evaluated points that are sufficiently close to the current iterate. Then, this model is optimized inside the trust region $\mathcal{B}(w_k,\Delta_k) := \{w : \|w - w_k\| \leq \Delta_k\}$. The global solution for the trust region subproblem is well known and can be obtained efficiently in $O(n^3)$ operations [12], which is not expensive, since in our setting d is small. The number of points that are used to construct the model is at most $\frac{1}{2}(n+1)(n+2)$, but good models that exploit some second-order information can be constructed with $O(n)$ points. Each iteration requires only one new evaluation of the function and the new function value either provides an improvement over the best observed value or can be used to improve the local model [49]. Thus, the method utilizes function evaluations very

efficiently.

Algorithm 7 Trust Region based Derivative-Free Optimization (DFO-TR)

Initializations:
Initialize w_0, $\Delta_0 > 0$, and choose $0 < \eta_0 < \eta_1 < 1$, $\theta > 1$, and $0 < \gamma_1 < 1 < \gamma_2$.
Define an interpolation set $\mathcal{W} \in \mathcal{B}(w_0, \Delta_0)$.
Compute $f(w)$ for all $w \in \mathcal{W}$, let $m = |\mathcal{W}|$.
Let $w_0 := \bar{w_0} = \arg\min_{w \in \mathcal{W}} f(w)$.
for $k = 1, 2, \cdots$ **do**
    Build the model:
    Discard all $w \in \mathcal{W}$ such that $\|w - w_k\| \geq \theta \Delta_k$.
    Using $\mathcal{W}$ construct an interpolation model:

$$Q_k(w) = f_k + g_k^T(w - w_k) + \frac{1}{2}(w - w_k)^T H_k(w - w_k).$$

    Minimize the model within the trust region:
    Compute $\hat{w}_k = \arg\min_{w \in \mathcal{B}(w_k, \Delta_k)} Q_k(w)$, and consequently $f(\hat{w}_k)$ and

$$\rho_k := \frac{f(w_k) - f(\hat{w}_k)}{Q_k(w_k) - Q_k(\hat{w}_k)}.$$

    Update the interpolation set:
    if $m < \frac{1}{2}(d+1)(d+2)$ **then**
     Add new point $\hat{w}_k$ to the interpolation set $\mathcal{W}$, and $m := m + 1$.
    else
        if $\rho_k \geq \eta_0$ **then** replace $\arg\max_{w \in \mathcal{W}} \|w - w_k\|$ with $\hat{w}_k$.
        else
            if $\|w - w_k\| < \max_{w \in \mathcal{W}} \|w - w_k\|$ **then** do the same.
    Update the trust region radius:
    if $\rho_k \geq \eta_1$ **then** $w_{k+1} \leftarrow \hat{w}_k$ and $\Delta_{k+1} \leftarrow \gamma_2 \Delta_k$.
    if $\rho_k < \eta_0$ **then** $w_{k+1} \leftarrow w_k$
        if $m > d + 1$ **then** update $\Delta_{k+1} \leftarrow \gamma_1 \Delta_k$
        else $\Delta_{k+1} \leftarrow \Delta_k$.

3.3 Bayesian Optimization versus DFO-TR

Bayesian optimization is known in the ML community as a powerful tool for optimizing nonconvex objective functions, which are expensive to evaluate, and whose

derivatives are not accessible. In terms of the required number of objective function evaluations, Bayesian optimization methods are considered to be some of the most efficient techniques [6, 29, 34] for black-box problems of low effective dimensionality. A Bayesian optimization framework, as outlined in Algorithm 8, like DFO-TR framework operates by constructing a (probabilistic) model $M(w)$ of the true function $f(w)$ by using function values computed thus far by the algorithm. The next iterate w_k is computed by optimizing an acquisition function a_M, which presents a trade-off between minimizing and improving the model. In theory, Bayesian optimization methods seek globally optimal solutions, due to their sampling schemes, which trade-off between exploitation and exploration [6, 18] by exploring areas where $f(w)$ has not been sampled. Specifically, Bayesian optimization methods construct a probabilistic model by using point evaluations of the true function. Then, by using this model, the subsequent configurations of the parameters will be selected [6] by optimizing an acquisition function derived from the model. The model is built based on all past evaluation points in an attempt to approximate the true function globally. As a result, the acquisition function is often not trivial to maintain and optimize and per iteration complexity of BO methods increases. On the other hand, DFO-TR and other model-based DFO methods content themselves with building a local model of the true function, hence maintenance of such models remains moderate and optimization step on each iteration is cheap. Different Bayesian optimization algorithms use different models and different acquisition functions, for instance, expected improvement [32] over the best observed function values is a popular acquisition function in the literature.

The key drawback and difficulty of BO methods is that the acquisition function may have a complex structure, and needs to be optimized globally on each iteration. For example, the algorithm in [6] uses deterministic derivative free optimizer DIRECT [28] to maximize the acquisition function. When evaluation of $f(w)$ is very expensive, the expense of optimizing the acquisition function may be small in comparison. However, in many cases, as we will see in our computational experiments, this expense can be dominant.

In contrast, the DFO-TR method, as described in Algorithm 7, maintains a

quadratic model by using only the points in the neighborhood of the current iterate and global optimization of this model subject to the trust region constraint can be done efficiently, as was explained in the previous section. While $Q(w)$ is a local model, it can capture nonconvexity of the true function $f(w)$ and hence allows the algorithm to follow negative curvature directions. As we will see in §3.4, for the same amount number of function evaluations, DFO-TR achieved better or comparable function values, while requiring significantly less computational time than Bayesian optimization algorithms (TPE, SMAC, and SPEARMINT).

Algorithm 8 Bayesian Optimization (BO)

1: Choose the probabilistic model M of the true function $f(w)$.
2: Initialize set $\mathcal{D}_0$ containing sample pairs of $(w, f(w))$.
3: **for** $k = 1, 2, \cdots$ **do**
4: Find w_k by optimizing the acquisition function over model M:

$$w_k \leftarrow \arg\min_w a_M(w \mid \mathcal{D}_{k-1})$$

5: Sample the objective function: $v_k := f(w_k)$.
6: Augment the data $\mathcal{D}_k = \{\mathcal{D}_{k-1}, (w_k, v_k)\}$ and update the model M.

3.4 Numerical Experiments

3.4.1 Optimizing Smooth, NonConvex Benchmark Functions

In this section, we compare the performance of DFO-TR and Bayesian optimization algorithms on optimizing three nonconvex smooth benchmark functions. We compare the precision Δf_{opt} with the global optimal value, which is known, and is computed after a given number of function evaluations.

Algorithm 7 is implemented in Python 2.7.11 [1] . We start from a zero vector as the initial point. In addition, the trust region radius is initialized as $\Delta_0 = 1$ and the initial interpolation set has $d + 1$ random members. The parameters are chosen as $\eta_0 = 0.001$, $\eta_1 = 0.75$, $\theta = 10$, $\gamma_1 = 0.98$, and $\gamma_2 = 1.5$. We have used

[1] https://github.com/TheClimateCorporation/dfo-algorithm

the hyperparameter optimization library, HPOlib [2], to perform the experiments on TPE, SMAC, and SPEARMINT algorithms, implemented in Python, Java 3, and MATLAB, respectively. Each benchmark function is evaluated on its known search space, as is defined in the default setting of the HPOlib (note that DFO-TR does not require nor utilizes a restricted search space).

We can see that on all three problems, DFO-TR reaches the global value accurately and quickly, outperforming BO methods. This is because DFO-TR utilizes second-order information effectively, which helps following negative curvature and significantly improving convergence in the absence of noise. Among the three Bayesian optimization algorithms SPEARMINT performs better while the performance of TPE and SMAC is comparable to each other, but inferior to those of SPEARMINT and DFO-TR.

Table 3.1: DFO-TR vs. BO on *Branin* function in terms of Δf_{opt}, over number of function evaluations. *Branin* is a two dimensional function with $f_{opt} = 0.397887$.

Algorithm	1	5	**11**	100
DFO-TR	15.7057	0.1787	0	0
TPE	30.0880	4.8059	3.4743	0.0180
SMAC	23.7320	10.3842	6.7017	0.0208
SPEARMINT	34.3388	17.1104	1.1615	3.88e-08

Table 3.2: DFO-TR vs. BO on *Camelback* function in terms of Δf_{opt}, over number of function evaluations. *Camelback* is a two dimensional function with $f_{opt} = -1.031628$.

Algorithm	1	10	**21**	100
DFO-TR	2.3631	0.1515	0	0
TPE	3.3045	0.5226	0.3226	0.0431
SMAC	1.0316	0.0179	0.0179	0.0036
SPEARMINT	2.3868	1.6356	0.1776	2.29e-05

[2] www.automl.org/hpolib

Table 3.3: DFO-TR vs. BO on *Hartmann* function in terms of Δf_{opt}, over number of function evaluations. *Hartmann* is a six dimensional function with $f_{opt} = -3.322368$.

Algorithm	1	25	**64**	250
DFO-TR	3.1175	0.4581	0	0
TPE	3.1862	2.5544	1.4078	0.4656
SMAC	2.8170	1.5311	0.6150	0.2357
SPEARMINT	2.6832	2.6671	2.5177	9.79e-05

3.4.2 Optimizing the AUC Function

In this section, we compare the performance of DFO-TR and the three Bayesian optimization algorithms, TPE, SMAC, and SPEARMINT, on the task of optimizing AUC function. While we will show that AUC is a smooth function, in practice we have a finite data set, hence we compute the noisy nonsmooth estimate of AUC. This essentially means that we can only expect to optimize the objective up to some accuracy, after which the noise will prevent further progress.

We have used 12 low dimensional data sets of Table 4.3. The average value of AUC and its standard deviation, using five-fold cross-validation, is reported as the performance measure. Table 3.4 summarizes the results. The initial vector w_0 for DFO-TR is set to zero and the search space of Bayesian optimization algorithms is set to the interval $[-1, 1]$. For each data set, a fixed total budget of number of function evaluations is given to each algorithm and the final AUC computed on the test set is compared.

Table 3.4: Comparing DFO-TR vs. BO algorithms in terms of solution time.

Data	num.	DFO-TR		TPE		SMAC		SPEARMINT	
	fevals	AUC ± std	time	AUC ± std	time	AUC ± std	time	AUC ± std	time
fourclass	100	0.835±0.019	0.31	**0.839**±0.021	12	0.839±0.021	77	0.838±0.020	5229
svmguide1	100	0.988±0.004	0.71	0.984±0.009	13	0.986±0.006	72	**0.987**±0.006	6435
diabetes	100	0.829±0.041	0.58	0.824±0.044	15	0.825±0.045	75	**0.829**±0.060	8142
shuttle	100	0.990±0.001	43.4	0.990±0.001	17	0.989±0.001	76	**0.990**±0.001	13654
vowel	100	0.975±0.027	0.68	0.965±0.029	16	0.965±0.038	77	**0.968**±0.025	9101
magic04	100	0.842±0.006	10.9	0.824±0.009	16	0.821±0.012	76	**0.839**±0.006	7947
letter	200	0.987±0.003	10.2	0.959±0.008	49	0.953±0.022	166	**0.985**±0.004	21413
segment	300	0.992±0.007	9.1	0.962±0.021	99	**0.997**±0.004	263	0.976±0.021	216217
ijcnn1	300	0.913±0.005	57.3	0.677±0.015	109	0.805±0.031	268	**0.922**±0.004	259213
svmguide3	300	0.776±0.046	13.5	0.747±0.026	114	**0.798**±0.035	307	0.7440±0.072	185337
german	300	0.795±0.024	9.9	0.771±0.022	120	0.778±0.025	310	**0.805**±0.020	242921
satimage	300	0.757±0.013	14.2	0.756±0.020	164	0.750±0.011	341	**0.761**±0.028	345398

For each data set, the bold number indicates the best average AUC value found by a Bayesian optimization algorithm. We can see that DFO-TR attains comparable or better AUC value to the best one of BO, in almost all cases. Since for each data set, all algorithms are performed for the same budget of number of function evaluations, we do not include the time spent on function evaluations in the reported time. Thus, the time reported in Table 3.4 is only the optimizer time. As we can see, DFO-TR is significantly faster than Bayesian optimization algorithms, while it performs competitively in terms of the average value of AUC. Note that the problems are listed in the order of increasing dimension d. Even thought the MATLAB implantation of SPEARMINT probably puts it at a certain disadvantage in terms of computational time comparisons, we observe that it is clearly a slow method, whose complexity grows significantly as d increases.

Table 3.5: Comparing DFO-TR vs. random search algorithm.

Data	DFO-TR		Random Search		Random Search	
	AUC ± std	fevals	AUC ± std	fevals	AUC ± std	fevals
fourclass	0.835±0.019	100	0.836±0.017	100	0.839±0.021	200
svmguide1	0.988±0.004	100	0.965±0.024	100	0.977±0.009	200
diabetes	0.829±0.041	100	0.783±0.038	100	0.801±0.045	200
shuttle	0.990±0.001	100	0.982±0.006	100	0.988±0.001	200
vowel	0.975±0.027	100	0.944±0.040	100	0.961±0.031	200
magic04	0.842±0.006	100	0.815±0.009	100	0.817±0.011	200
letter	0.987±0.003	200	0.920±0.026	200	0.925±0.018	400
segment	0.992±0.007	300	0.903±0.041	300	0.908±0.036	600
ijcnn1	0.913±0.005	300	0.618±0.010	300	0.629±0.013	600
svmguide3	0.776±0.046	300	0.690±0.038	300	0.693±0.039	600
german	0.795±0.024	300	0.726±0.028	300	0.739±0.021	600
satimage	0.757±0.013	300	0.743±0.029	300	0.750±0.020	600

Next, we compare the performance of DFO-TR versus the random search algorithm (implemented in Python 2.7.11) on maximizing AUC. Table 3.5 summarizes the results, in a similar manner to Table 3.4. Moreover, in Table 3.5, we also allow random search to use twice of the budget of the function evaluations, as is done in [25] when comparing random search to BO. The random search algorithm is competitive with DFO-TR on a few problems, when using twice the budget, however, it can be seen that as the problem dimension grows, the efficiency of the random search goes down substantially. Overall, DFO-TR consistently surpasses random search when function evaluation budgets are equal, while not requiring very significant overhead, as the BO methods.

Finally, figures 3.1 and 3.2 illustrate the per iteration behavior of DFO method versus the Bayesian optimization algorithms for optimizing AUC function. We can see that DFO method improves the objective values faster than the Bayesian optimization algorithms.

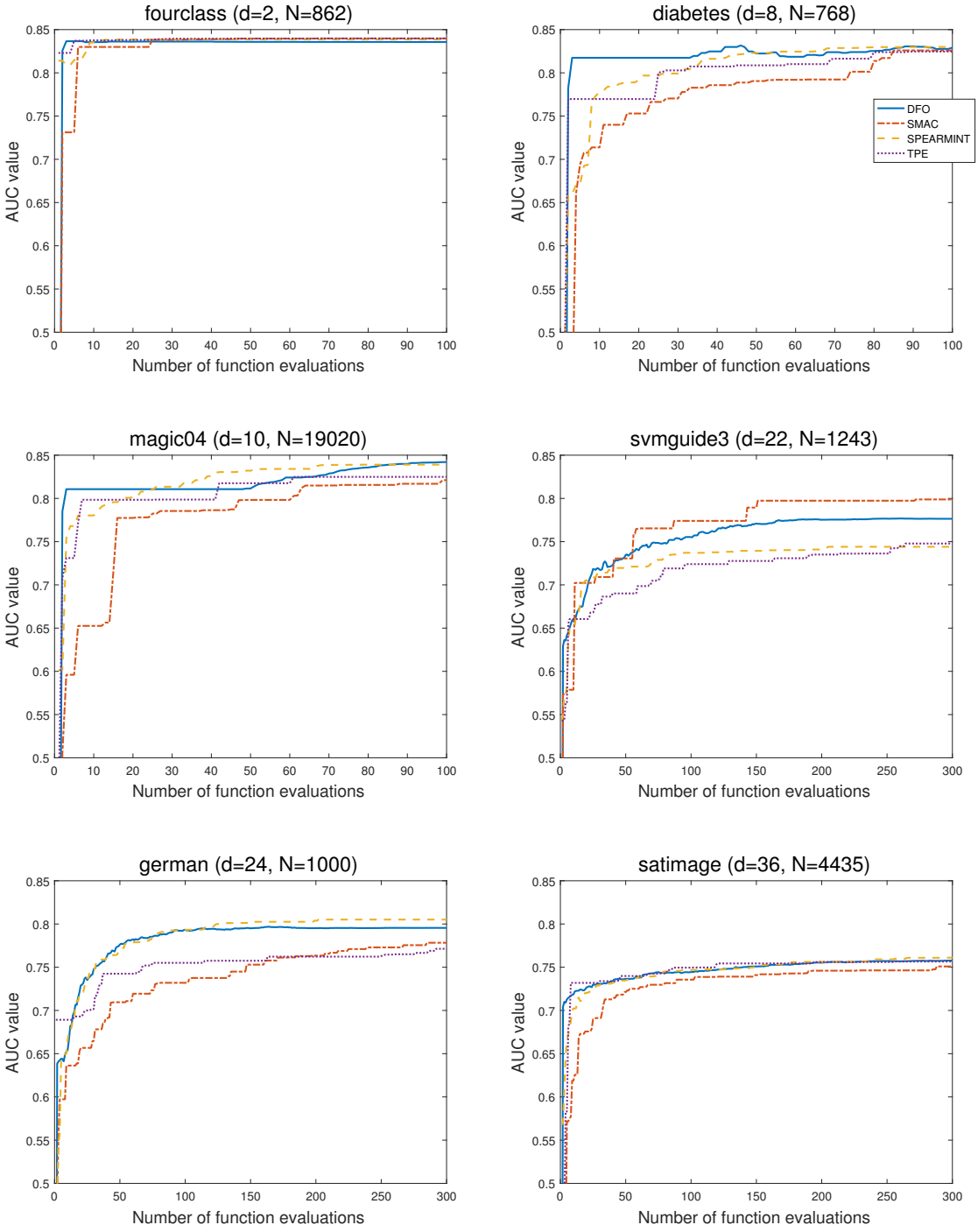


Figure 3.1: DFO-TR vs. BO algorithms.

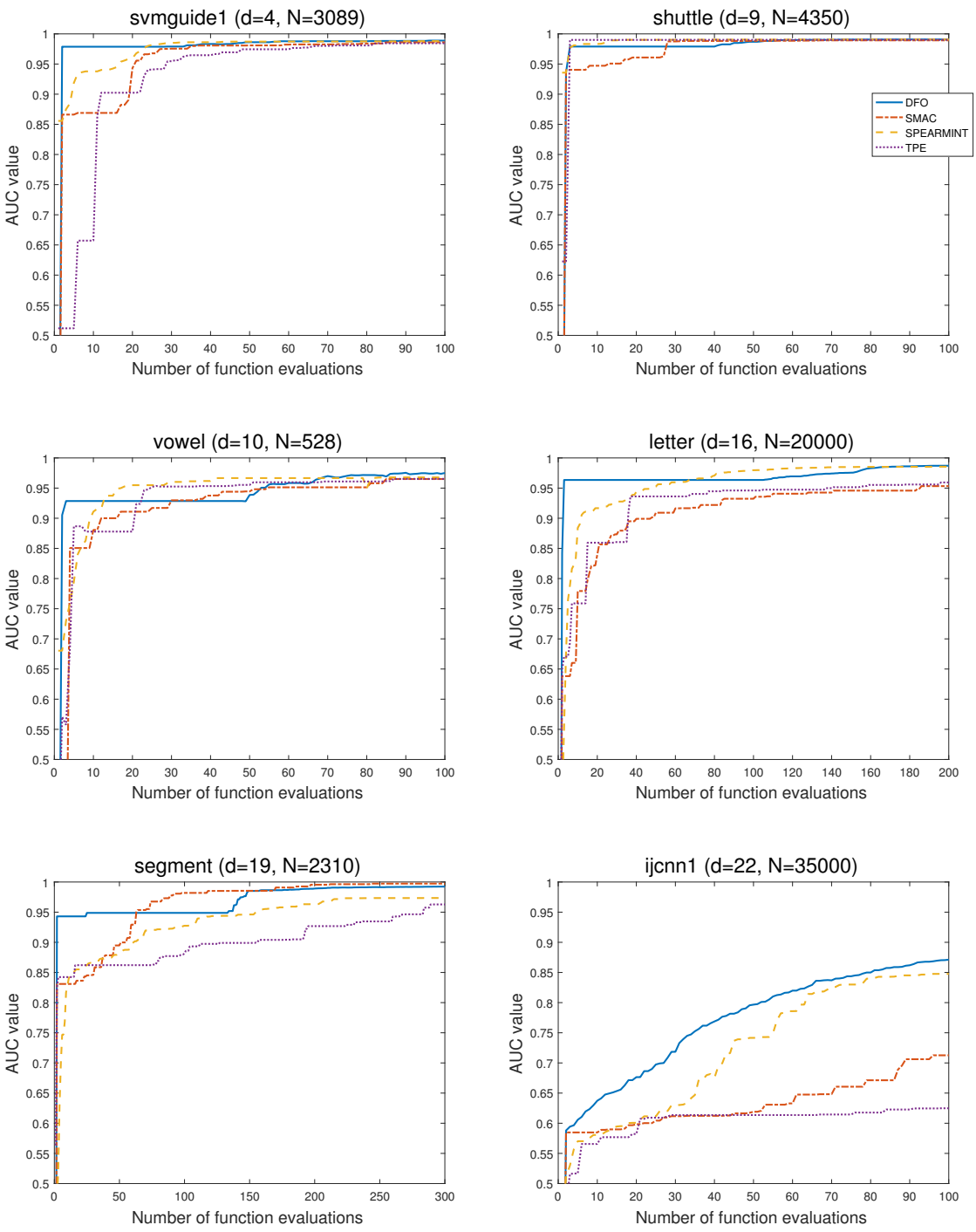


Figure 3.2: DFO-TR vs. BO algorithms.

3.4.2.1 Stochastic versus Deterministic DFO-TR

In order to further improve efficiency of DFO-TR, we observe that STORM framework and theory [10] suggests that noisy function evaluations do not need to be accurate far away from the optimal solution. In our context, this means that AUC can be evaluated on small subsets of the training set, which gradually increase as the algorithm progresses. In particular, at each iteration, we compute AUC on a subset of data, which is sampled from positive and negative sets, with size n^+ and n^-, uniformly at random, at the rate of

$$\min\{n, \max\{k \times \lfloor 50 \times (n/(n^+ + n^-))\rfloor + \lfloor 1000 \times (n/(n^+ + n^-))\rfloor, \lfloor 0.1 \times n\rfloor\}\},$$

where $n = n^+$ when we sample from the positive class and $n = n^-$ when we sample from the negative one. For each class, at least 10 percent of the whole training data is used.

We include an additional modification–after each unsuccessful step with $\rho_k < \eta_0$, we compute $f_{new}(w_k)$ by resampling over data points. Then, we update $f(w_k)$ such that $f(w_k) = (f(w_k) + f_{new}(w_k))/2$. This is done, so that accidental incorrectly high AUC values are not preventing the algorithm from making progress. This results in a less expensive (in terms of function evaluation cost) algorithm, while, as we see in Figure 3.3, the convergence to the optimal solution is comparable.

We chose two data sets shuttle and letter to compare the performance of the stochastic variant of the DFO-TR with the deterministic one. These sets were chosen because they contain a relatively large number of data points and hence the effect of subsampling can be observed. We repeated each experiment four times using five-fold cross-validation (due to the random nature of the stochastic sampling). Hence, for each problem, the algorithms have been applied 20 times in total and the average AUC values are reported in Figure 3.3. At each round, all parameters of DFO-TR and S-DFO-TR are set as described in §3.4.1, except w_0 which is a random vector evenly distributed over $[-1, 1]$.

As we see in Figure 3.3, the growth rate of AUC over iterations in S-DFO-TR

is as competitive as that of DFO-TR. However, by reducing the size of the data sets, the iterations of S-DFO-TR are significantly cheaper than that of DFO-TR, especially at the beginning. S-DFO-TR is comparable with DFO-TR in terms of the progress in the AUC value while having access to less number of data points.

This indicates that the methods can handle large data sets.

We finally note that we chose to optimize AUC over linear classifiers for simplicity only. Any other classifier parametrized by w can be trained using a black-box optimizer in a similar way. However, the current DFO-TR method has some difficulties in convergence when dimension of w is very large.

3.4.3 Hyperparameter Tuning of Cost-Sensitive RBF-Kernel SVM

Finally, we turn to hyperparamater tuning to show that DFO-TR can also outperform state-of-the-art methods on this problem. We consider tuning parameters of the cost-sensitive RBF-kernel SVM with ℓ_2 regularization parameter λ, kernel width γ, and positive class cost C^+. In this setting, we compare the performance of DFO-TR, random search, and Bayesian optimization algorithms in tuning a three-dimensional hyperparameter $w = (\lambda, \gamma, C^+)$, in order to achieve a high test accuracy.

For the random search algorithm, as well as the Bayesian optimization algorithms, the search space is chosen as $\lambda \in [10^{-6}, 10^0]$, $\gamma \in [10^0, 10^3]$, as is done in [25], and $C^+ \in [10^{-2}, 10^2]$. The setting of Algorithm 7 is as described in §3.4.1, while $w_0 = (\lambda_0, \gamma_0, C_0^+)$ is a three-dimensional vector randomly drawn from the search space defined above.

We have used the five-fold cross-validation with the train-validate-test framework as follows: we used two folds as the training set for the SVM model, other two folds as the validation set to compute and maximize the validation accuracy, and the remaining one as the test set to report the test accuracy.

Figure 3.4 illustrates the performance of DFO-TR versus random search and Bayesian optimization algorithms, in terms of the average test accuracy over the

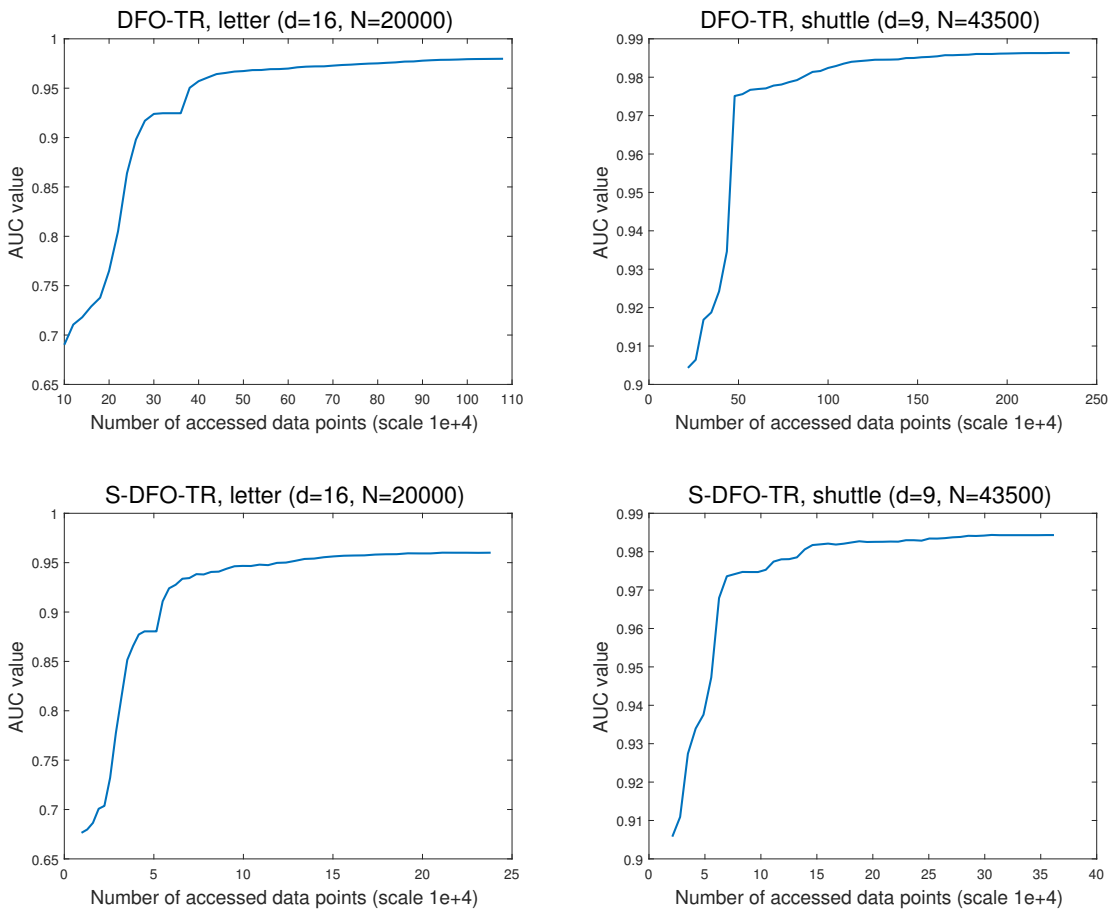


Figure 3.3: Comparison of stochastic DFO-TR and deterministic one in optimizing AUC function.

number of function evaluations. As we can see, DFO-TR constantly surpasses random search and Bayesian optimization algorithms. It is worth mentioning that random search is competitive with BO methods and in contrast to §3.4.1 and §3.4.2, SMAC performs the best among the Bayesian optimization algorithms.

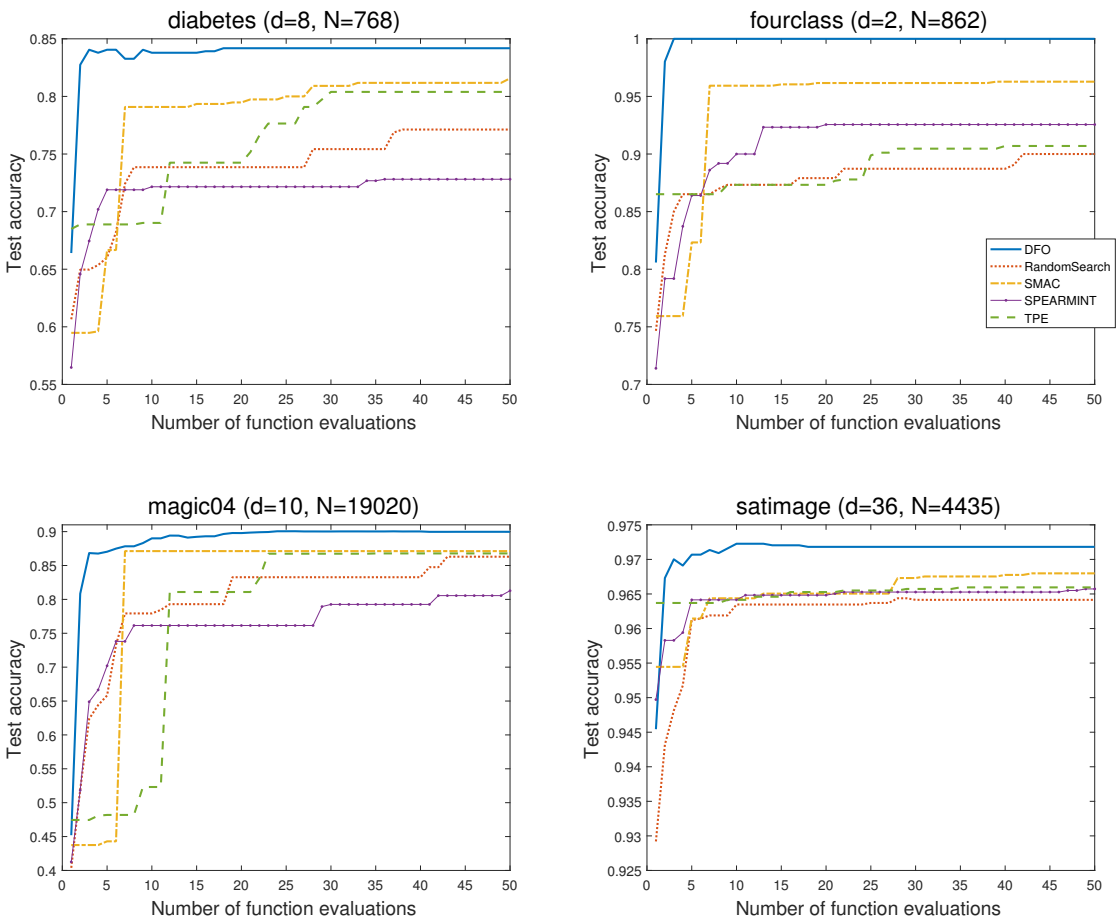


Figure 3.4: Comparison of DFO-TR, random search, and BO algorithms on tuning cost-sensitive RBF-kernel SVM hyperparameters.

3.4.4 Hyperparameter Tuning of Cost-Sensitive Logistic Regression

In this section, we describe how one can maximize AUC as a function of the costs and the regularization parameter of the cost-sensitive logistic regression. First, consider an imbalance data set $\mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^-$ which contains $|\mathcal{S}^+| = n^+$ positive examples as the minority class and $|\mathcal{S}^-| = n^-$ negative examples as the majority class, such that $|\mathcal{S}| = n = n^+ + n^-$. Assume that all data points in the set $\mathcal{S}$ are in $d-$dimensional

space, so that $x_i \in \mathbb{R}^d$ with corresponding labels $y_i \in \{-1, +1\}$. The cost-sensitive logistic regression is defined as the following

$$f(w) = \frac{1}{n}\sum_{i=1}^{n} C(y_i)\log(1+\exp(-y_i \cdot w^T x_i)) + \frac{\lambda}{2}\|w\|_2, \tag{3.1}$$

where $w \in \mathbb{R}^{d+1}$ contains the intercept of the classifier and

$$C(y_i) = \begin{cases} C^+ & \text{if } \ y_i > 0, \\ C^- & \text{otherwise.} \end{cases}$$

The composite function $f(w)$ contains $\sum_{i=1}^{n}\log(1+\exp(-y_i \cdot w^T x_i))$, as the average logistic loss, and $\frac{1}{2}\lambda\|w\|_2$, as the ℓ_2-regularization term. The cost function $C(y_i)$ has been defined in order to bias the classifier $w^T x$ toward the rare positive class. However, evaluating the value of the unknown positive and negative costs $\{C^+, C^-\}$ is the key challenge.

In this section, in order to ensure that Algorithm 7 performs well in optimizing the parameters of the problem (3.1), in terms of the ranking quality of the resulting classifier, we need to show that the expected value of AUC is a smooth function of the parameters $\{C^+, C^-, \lambda\}$. To this end, since in the following chapter we will show that under some assumptions AUC is a smooth function of the vector w, one needs to prove that w also is a smooth function of the parameters $\{C^+, C^-, \lambda\}$. In what follows, we prove this smoothness under some practical conditions.

Theorem 13. *In (3.1), the vector w is a smooth function of the non-zero parameters C^+ ,C^-, and λ if the matrix M and the added matrix $M|v$ have the same rank (one special case which satisfies this condition is when the matrix M is a full rank matrix)*

$$\begin{aligned} M &= \sum_{i=1}^{n^+} \frac{\exp(y_i \cdot w^T x_i)}{(1+\exp(y_i \cdot w^T x_i))^2}(x_i x_i^T) \\ \text{and} \quad v &= \frac{1}{C^+}\sum_{i=1}^{n^+} \frac{y_i x_i}{1+\exp(y_i \cdot w^T x_i)}. \end{aligned} \tag{3.2}$$

Similar condition is required when M and v are constructed based on the negative class.

Proof. Consider the cost-sensitive logistic regression function as the following

$$f(w) = \frac{1}{n}\left[\sum_{i=1}^{n^+} C^+ \log(1+\exp(-y_i \cdot w^T x_i)) + \sum_{i=1}^{n^-} C^- \log(1+\exp(-y_i \cdot w^T x_i))\right] + \frac{\lambda}{2}\|w\|_2^2.$$

The first derivative of the above function with respect to w will be as the following

$$\partial_w f(w) = \frac{1}{n}\left[C^+ \sum_{i=1}^{n^+} \frac{-y_i x_i}{1+\exp(y_i \cdot w^T x_i)} + C^- \sum_{i=1}^{n^-} \frac{-y_i x_i}{1+\exp(y_i \cdot w^T x_i)}\right] + \lambda w.$$

Based on the optimality condition, we have $\partial_w f(w) = 0$. On the other hand, if we define w as a function of the three parameters C^+, C^-, and λ, so that $w = q(\lambda, C^+, C^-)$, we will have

$$Q(\lambda, C^+, C^-) = \frac{1}{n}[C^+ \sum_{i=1}^{n^+} \frac{-y_i x_i}{1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i)} + C^- \sum_{i=1}^{n^-} \frac{-y_i x_i}{1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i)}] + \lambda q(\lambda, C^+, C^-) = 0.$$

Now, all the partial derivatives of function $Q(\lambda, C^+, C^-)$ are equal to zero vector as the following

$$0 = \partial_\lambda Q(\lambda, C^+, C^-) = q(\lambda, C^+, C^-) + \lambda \partial_\lambda q(\lambda, C^+, C^-) \tag{3.3}$$

and

$$
\begin{aligned}
0 &= \partial_{C^+} Q(\lambda, C^+, C^-) \\
&= \frac{1}{n}[\sum_{i=1}^{n^+} \frac{-y_i x_i}{1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i)} \\
&\quad + C^+ \sum_{i=1}^{n^+} \frac{(y_i x_i)\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i)(y_i x_i^T \partial_{C^+} q(\lambda, C^+, C^-))}{(1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i))^2}] \\
&= \frac{1}{n}[\sum_{i=1}^{n^+} \frac{-y_i x_i}{1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i)} \\
&\quad + C^+ \sum_{i=1}^{n^+} \frac{\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i) y_i^2 (x_i x_i^T)}{(1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i))^2} \partial_{C^+} q(\lambda, C^+, C^-)].
\end{aligned}
$$

Therefore,

$$
M \partial_{C^+} q(\lambda, C^+, C^-) = b, \tag{3.4}
$$

where

$$
\begin{aligned}
M &= \sum_{i=1}^{n^+} \frac{\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i)}{(1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i))^2} (x_i x_i^T) \quad \text{and} \\
v &= \frac{1}{C^+} \sum_{i=1}^{n^+} \frac{y_i x_i}{1+\exp(y_i \cdot q^T(\lambda, C^+, C^-)x_i)}.
\end{aligned} \tag{3.5}
$$

Based on (3.3), the partial derivative $\partial_\lambda q(\lambda, C^+, C^-)$ is defined for nonzero value of λ. On the other hand, based on (3.4) and (3.5), the partial derivatives $\partial_{C^+} q(\lambda, C^+, C^-)$ and similarly $\partial_{C^-} q(\lambda, C^+, C^-)$ can be defined respectively for nonzero values of C^+ and C^- if the matrix M and the added matrix $M|v$ have the same rank (one special case which satisfies this condition is when matrix M is a full rank matrix). □

Now, we can guarantee that AUC is a smooth function of the class weights $\{C^+, C^-\}$ as well as the regularization term λ.

Here, the input of Algorithm 7 is a three dimensional point $\{C^+, C^-, \lambda\}$. In fact, we use the value of these parameters in problem (3.1), and we minimize $f(w)$ via proximal quasi-Newton algorithm, as presented in [48]. Then, we use the resulting optimized vector w to compute the corresponding AUC value using the finite sets

$\mathcal{S}^+$ and $\mathcal{S}^-$. Thus, AUC function has the role of the black-box function, which can be obtained via Algorithm 9, efficiently.

Algorithm 9 Computing AUC Value via Sorting Scheme

1: Initialize $sum = 0$ and $sum_{partial} = 0$.
2: Using quick sort scheme, sort the rank of all positive and negative points in descending order and store them in set $\mathcal{L}_{rank}$.
3: Define set $\mathcal{L}_{label}$ which keeps the label of the corresponding points in the set $\mathcal{L}_{rank}$.
4: **while** $k \leq n$ **do**
5: **if** $x_i \in \mathcal{L}_{label}$ is positive **then** $sum \leftarrow sum + sum_{partial}$.
6: **else** $sum_{partial} \leftarrow sum_{partial} + 1$.
7: $k \leftarrow k + 1$.
8: Compute the final value of AUC function as $F_{AUC} := sum/n^+n^-$.

In this section, we compare the following algorithms:

- Cost-Sensitive Logistic Regression based on the Class Distribution Ratio (CSLR-CDR),
- Cost-Sensitive Logistic Regression based on the Optimized Ratio which is obtained through AUC maximization via DFO-TR with Warm Starting Point (CSLR-OR-WSP).

In terms of choosing the algorithm parameters, in CSLR-CDR, we use the class distribution ratio to set the cost values such that $C^+ = n^-/n^+$ and $C^- = 1$. We also set the regularization parameter to $\lambda = 1/n$, which is a common choice in the literature. In CSLR-OR-WSP, for each run, the initial cost values (C_0^+, C_0^-) have been set to the class distribution ratio, such that $(n^-/n^+, 1)$ and $\lambda^0 = 1/n$. The trust region radius has been initialized as $\Delta_0 = 5$. The initial interpolation set has the minimum required members, which is $m = 10$. The parameters of evaluating the success of each trust region step have been set to $\eta_0 = 0.001$ and $\eta_1 = 0.75$. Finally, in terms of updating the trust region radius, the parameters have been set to $\gamma_1 = 0.98$ and $\gamma_2 = 1.5$. Table 3.6 shows the average value of AUC for these experiments. In our experiments, we use 20 binary class data sets, which can be

downloaded from LIBSVM website [3] and UCI machine learning repository [4]. Table 4.3, in the next chapter, shows the information of all of these data sets.

In CSLR-OR-WSP, in order to use each data set, we randomly divide it into five different folds. We use three folds as the training set of the inner "Logistic Regression Optimization", done via proximal quasi-Newton algorithm, one fold as the training set of "AUC Optimization" through DFO-TR, and finally one fold as the test set. In CSLR-CDR, we use four of those folds as the training set and one of them as the test set. We did such an experiment four times, so in total we did 20 experiments for each data set. The results of the performance of the CSLR-OR-WSP are obtained after 50 iterations. The stopping criterion for the inner logistic regression optimization is the maximum number of iterations, which has been set to 100. In order to have a fair comparison, the running time of CSLR-CDR has been restricted to the running time of CSLR-OR-WSP, so both of these algorithms are running for the same amount of time.

[3] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
[4] http://archive.ics.uci.edu/ml/

Table 3.6: Comparing CSLR-OR-WSP vs. CSLR-CDR.
in terms of the averaged AUC value.

Algorithm	**sonar**	**fourclass**	**svmguide1**	**magic04**
CSLR-CDR	0.837656	0.835398	0.9819	0.8415
CSLR-OR-WSP	0.84183	0.835387	0.988035	0.843396
Algorithm	**diabetes**	**german**	**a9a**	**svmguide3**
CSLR-CDR	0.827639	0.78878	0.902774	0.774338
CSLR-OR-WSP	0.82269	0.784636	0.902868	0.797397
Algorithm	**connect-4**	**shuttle**	**HAPT**	**segment**
CSLR-CDR	0.899115	0.988924	0.999992	0.999791
CSLR-OR-WSP	0.898861	0.991161	0.999994	0.999841
Algorithm	**mnist**	**ijcnn1**	**satimage**	**vowel**
CSLR-CDR	0.994864	0.934975	0.761618	0.978475
CSLR-OR-WSP	0.995341	0.934986	0.760733	0.975405
Algorithm	**poker**	**letter**	**w1a**	**w8a**
CSLR-CDR	0.502508	0.987558	0.960954	0.960331
CSLR-OR-WSP	0.52133	0.988293	0.962368	0.961645

As we can see in Table 3.6, optimizing the parameters of the cost-sensitive logistic regression improved the ranking performance of the final classifier, only in two data sets "svmguide3" and "poker".

3.5 Conclusion

In this chapter, we demonstrated that the model-based derivative free optimization is a better alternative to Bayesian optimization for some black-box optimization tasks arising in machine learning. We relied on an existing convergent stochastic trust region framework to provide theoretical foundation for the chosen algorithm, and we demonstrated the efficiency of a practical implementation of DFO-TR for optimizing AUC function over the set of linear classifiers, hyperparameter tuning, and on other benchmark problems.

Chapter 4

Directly Optimizing Prediction Error and AUC

4.1 Introduction

In this chapter, we consider classical binary linear classification problems in supervised Machine Learning (ML). In other words, given a finite set labeled data (labeled to form a positive and a negative class), the aim is to obtain a linear classifier that predicts the positive/negative labels of unseen data points as accurately as possible. To measure the accuracy of a classifier, the expected prediction error, which measures the percentage of mislabeled data points, also known as the zero-one loss function, is often used. However, since the empirical approximation of the prediction error is a nonsmooth nonconvex function, whose gradient is either zero or not defined, other surrogate loss functions are typically used to determine the linear classifier. For example, standard ML tools, such as support vector machines [14, 43, 53] and logistic regression [16], aim to optimize empirical prediction error, while using hinge loss and logistic loss, respectively, as surrogate functions of the zero-one loss function.

Many real world ML problems deal with imbalanced data sets, which contain rare positive data points, as the minority class, but numerous negative ones, as the

majority class. When these two data classes are highly imbalanced, the prediction error function is not a useful prediction measure. For example, if the data set contains only 0.01% of the positive examples, then a predictor that simply classifies every data point as negative has 99.99% accuracy, while obviously failing to achieve any meaningful prediction. The prediction measure is often modified to incorporate class importance weights, in which case it can be used for imbalanced data sets. All results of this work easily extend to such modification. However, a more established and robust measure of prediction accuracy which is used in practice is Area Under Receiver Operating Characteristic (ROC) Curve (AUC) [22]. AUC is a reciprocal of the ranking loss, which is similar to the zero-one loss, in the sense that it measures the percentage of pairs of data samples, one from the negative class and one from the positive class, such that the classifier assigns a larger label to the negative sample than to the positive one. In other words, $1-$AUC counts the percentage of incorrectly "ranked" pairs [33]. The empirical approximation of AUC, just as that of zero-one loss, is a discontinuous, nonsmooth function, whose gradient is either zero or undefined. This difficulty motivates various techniques for optimizing continuous approximations of AUC. For example, the ranking loss can be replaced by convex loss functions such as pairwise logistic loss or hinge loss [27, 46, 57, 65], which results in continuous convex optimization problem. A drawback of such approach, aside from the fact that a different objective is optimized, is that such loss has to be computed for each pair of data points, which significantly increases the complexity of the underlying optimization algorithm.

In this work, we propose a novel method of directly optimizing the expected prediction error and the expected AUC value of a linear classifier in the case of binary classification problems. First, we use the probabilistic interpretation of the expected prediction error and we show that if the distribution of the positive and negative classes obey normal distributions, then the expected prediction error of a linear classifier is a smooth function with a closed-form expression. Thus, its gradient can be computed and a gradient-based optimization algorithm can be used. The closed form of the function depends on the first and second moments of the related normal distributions, hence these moments are needed to compute the function value

as well as the gradient.

Similarly, under the assumption that the class of the positive and negative data sets jointly obey a normal distribution, we show that the corresponding expected AUC value of a linear classifier is a smooth function with closed form expression, which depends on the first and second moments of the distribution. Similarly to optimizing the prediction error, this novel result allows any gradient-based optimization algorithm to be applied to optimize the AUC value of a linear classifier.

Through empirical experiments we show that even when the data sets do not obey normal distribution, optimizing the derived functional forms of prediction error and AUC, using empirical approximate moments, often produces better predictors than those obtained by optimizing surrogate approximations, such as logistic and hinge losses. This behavior is in contrast with, for example, Linear Discriminant Analysis (LDA) [24], which is the method to compute linear classifiers under the Gaussian assumption. Unlike LDA, our method in fact relies on the linear functions of the data, but not the data itself, to obey Gaussian distribution. In §4.4, we provide theoretical justification, for our empirical observations based on the law of large numbers for dependent variables.

Another key advantage of the proposed method over the classical empirical risk minimization is that the training data is only used once at the beginning of the algorithm to compute the approximate moments. After that each iteration of an optimization algorithm only depends on the dimension of the classifier, while optimizing logistic loss or pairwise hinge loss using gradient-based method depends on the data size at each iteration.

This chapter is organized as follows. In the next section we state preliminaries and the problem description. In Section 4.3 we show that the prediction error and AUC are smooth functions if the data obey normal distribution. In Section 4.4 we state the conditions under which we can extend our results to data sets with any arbitrary distribution. We present computational results in Section 4.6, and finally, we state our conclusions in Section 4.7.

4.2 Preliminaries and Problem Description

We consider the classical setting of supervised machine learning, where we are given a finite training set $\mathcal{S}$ of n pairs,

$$\mathcal{S} := \{(x_i, y_i) \quad : \quad i = 1, \cdots, n\},$$

where $x_i \in \mathbb{R}^d$ are the input vectors of features and $y_i \in \{+1, -1\}$ are the binary output labels. It is assumed that each pair (x_i, y_i) is an i.i.d. sample of the random variable (X, Y) with some unknown joint probability distribution $P_{X,Y}(x, y)$ over the input space $\mathcal{X}$ and output space $\mathcal{Y}$.

The set $\mathcal{S}$ is known as a training set. The goal is to compute a linear classifier function $f : \mathcal{X} \to \mathcal{Y}$, so that given a random input variable X, f can accurately predict the corresponding label Y.

As discussed in §4.1, there are two different performance measures to evaluate the quality of f: the prediction error, which approximates the expected risk, and the AUC. Expected risk of a linear classifier $f(x; w) = w^T x$ for 0-1 loss function is defined as

$$\begin{aligned} F_{error}(w) &= \mathbb{E}_{\mathcal{X},\mathcal{Y}}\left[\ell_{01}\left(f(X; w), Y\right)\right] \\ &= \int_{\mathcal{X}} \int_{\mathcal{Y}} P_{X,Y}(x, y) \ell_{01}\left(f(x; w), y\right) dydx, \end{aligned} \tag{4.1}$$

where

$$\ell_{01}\left(f(x; w), y\right) = \begin{cases} +1 & \text{if} \quad y \cdot f(x; w) < 0, \\ 0 & \text{if} \quad y \cdot f(x; w) \geq 0. \end{cases}$$

A finite sample approximation of (4.1), given a training set $\mathcal{S}$, is the following empirical risk

$$\hat{F}_{error}(w; \mathcal{S}) = \frac{1}{n} \sum_{i=1}^{n} \ell_{01}\left(f(x_i; w), y_i\right). \tag{4.2}$$

The difficulty of optimizing (4.2), even approximately, arises from the fact that its gradient is either not defined or is equal to zero. Thus, gradient-based optimization methods cannot be applied. The most common alternative is to utilize the logistic regression loss function, as an approximation of the prediction error and solve the

following unconstrained convex optimization problem

$$\min_{w\in\mathbb{R}^d} \hat{F}_{log}(w) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1+\exp(-y_i\cdot f(x_i;w))\right) + \lambda r(w), \tag{4.3}$$

where $\lambda r(w)$ is the regularization term, with $r(\cdot) = \|\cdot\|_1$ or $r(\cdot) = \|\cdot\|_2$ as possible examples.

We now discuss the AUC function as the quality measure of a classifier, which is often the industry standard. For that let us define

$$\begin{aligned}
&\mathcal{S}^+ := \{x : (x,y)\in\mathcal{S},\ y=+1\} := \{x_i^+ \ :\ i=1,\cdots,n^+\}, \quad \text{where} \quad x_i^+\in\mathbb{R}^d \ \text{ and}\\
&\mathcal{S}^- := \{x : (x,y)\in\mathcal{S},\ y=-1\} := \{x_j^- \ :\ j=1,\cdots,n^-\}, \quad \text{where} \quad x_j^-\in\mathbb{R}^d.
\end{aligned}$$

Hence $\mathcal{S}^+$ and $\mathcal{S}^-$ are the sets of all positive and negative samples in $\mathcal{S}$, respectively, and they contain only inputs x, instead of pairs (x,y). Let $|\mathcal{S}^+| = n^+$ and $|\mathcal{S}^-| = n^-$. The AUC value of a classifier $f(x;w)$, given the positive set $\mathcal{S}^+$ and the negative set $\mathcal{S}^-$ can be obtained via Wilcoxon-Mann-Whitney (WMW) statistic result [33], e.g.,

$$\hat{F}_{AUC}\left(w;\mathcal{S}^+,\mathcal{S}^-\right) = \frac{\sum_{i=1}^{n^+}\sum_{j=1}^{n^-} 1\left[f(x_i^+;w) > f(x_j^-;w)\right]}{n^+\cdot n^-}, \tag{4.4}$$

where

$$1\left[f(x_i^+;w) > f(x_j^-;w)\right] = \begin{cases} +1 & \text{if} \quad f(x_i^+;w) > f(x_j^-;w),\\ 0 & \text{otherwise.}\end{cases}$$

Now, let $\mathcal{X}^+$ and $\mathcal{X}^-$ denote the space of the positive and negative input vectors, respectively, so that x_i^+ is an i.i.d. observation of the random variable X^+ from $\mathcal{X}^+$ and x_j^- is an i.i.d. observation of the random variable X^- from $\mathcal{X}^-$. Then, given the joint probability distribution $P_{X^+,X^-}(x^+,x^-)$, the expected AUC function of a classifier $f(x;w)$ is defined as

$$\begin{aligned}
\bar{F}_{AUC}(w) =&\ \mathbb{E}_{\mathcal{X}^+,\mathcal{X}^-}\left[1\left[f\left(X^+;w\right) > f\left(X^-;w\right)\right]\right]\\
=&\ \int_{\mathcal{X}^+}\int_{\mathcal{X}^-} P_{X^+,X^-}\left(x^+,x^-\right)\cdot 1\left[f\left(x^+;w\right) > f\left(x^-;w\right)\right] dx^- dx^+.
\end{aligned} \tag{4.5}$$

The $\hat{F}_{AUC}\left(w;\mathcal{S}^+,\mathcal{S}^-\right)$ computed by (4.4) is an unbiased estimator of $F_{AUC}(w)$. Similarly to the empirical risk minimization, the problem of optimizing AUC value of a predictor is not straightforward since the gradient of this function is either zero or not defined. Thus, gradient-based optimization methods cannot be applied.

As in the case of prediction error, various techniques have been proposed to approximate the AUC with a surrogate function. In [62], the indicator function $1[\cdot]$ in (4.4) is substituted with a sigmoid surrogate function, e.g., $1/(1+e^{-\beta\left(f(x^+;w)-f(x^-;w)\right)})$ and a gradient descent algorithm is applied to this smooth approximation. The choice of the parameter β in the sigmoid function definition significantly affects the output of this approach; although a large value of β renders a closer approximation of the step function, it also results in large oscillations of the gradients, which in turn can cause numerical issues in the gradient descent algorithm. Similarly, as is discussed in [46], pairwise exponential loss and pairwise logistic loss can be utilized as convex smooth surrogate functions of the indicator function $1[\cdot]$. In these settings, any gradient-based optimization method can be used to optimize the resulting approximate AUC value. However, due to the required pairwise comparison of the value of $f(\cdot;w)$, for each positive and negative pair, the complexity of computing function value as well as the gradient will be of order of $\mathcal{O}\left(n^+n^-\right)$, which can be very expensive. In [57], pairwise hinge loss has been used as a surrogate function, resulting the following approximate AUC value

$$F_{hinge}\left(w\right)=\frac{\sum_{i=1}^{n^+}\sum_{j=1}^{n^-}\max\left\{0,1-\left(f(x_j^-;w)-f(x_i^+;w)\right)\right\}}{n^+\cdot n^-}. \tag{4.6}$$

The advantage of pairwise hinge loss over other alternative approximations lies in the fact that the function values as well as the gradients of pairwise hinge loss can be computed in roughly $\mathcal{O}\left(n\log(n)\right)$ time, where $n=n^++n^-$, by first sorting all values $f(x_j^-;w)$ and $f(x_i^+;w)$. One can utilize numerous stochastic gradient schemes to reduce the per-iteration complexity of optimizing surrogate AUC objectives, however, the approach we propose here achieves the same or better result with a simpler method.

In this work, we propose to optimize alternative smooth approximations of expected risk and expected AUC, which display good accuracy and also have low computational cost. Towards that end, in the next section, we show that, if the data distribution is normal, then the expected risk and expected AUC of a linear classifier are both smooth functions with closed form expressions.

4.3 Prediction Error and AUC as Smooth Functions

In this section, first we show that the expected risk is a smooth function if the data points obey a normal distribution. To this end, we need to consider the probabilistic interpretation of the prediction error, e.g.,

$$\begin{aligned} F_{error}(w) &= \mathbb{E}_{\mathcal{X},\mathcal{Y}}\left[\ell_{01}\left(f(X;w),Y\right)\right] \\ &= P(Y \cdot w^T X < 0). \end{aligned} \tag{4.7}$$

Assuming that the true values of the prior probabilities $P(Y=+1)$ and $P(Y=-1)$ are known or obtainable from a trivial calculation, then we can conclude the following result.

Lemma 11. *Given the prior probabilities $P(Y=+1)$ and $P(Y=-1)$ we can write*

$$\begin{aligned} F_{error}(w) &= P(Y \cdot w^T X < 0) \\ &= P\left(w^T X^+ \leq 0\right) P\left(Y=+1\right) + \left(1 - P\left(w^T X^- \leq 0\right)\right) P\left(Y=-1\right), \end{aligned}$$

where X^+ and X^- are random variables from positive and negative classes, respectively.

Proof. Note that we can split the whole set $\left\{(X,Y) \ : \ Y \cdot w^T X < 0\right\} \subset \mathcal{X} \times \mathcal{Y}$ into two disjoint sets as the following:

$$\left\{(X,Y) \ : \ Y \cdot w^T X < 0\right\} = \left\{(X^+,+1) \ : \ w^T X^+ < 0\right\} \cup \left\{(X^-,-1) \ : \ w^T X^- \geq 0\right\}.$$

Now, by using the preceding definition in (4.7) we will have:

$$\begin{aligned}
F_{error}(w) = &\ P\left(Y \cdot w^T X < 0\right) \\
= &\ P\left(Y \cdot w^T X < 0 \cap Y = +1\right) + P\left(Y \cdot w^T X < 0 \cap Y = -1\right) \\
= &\ P\left(Y \cdot w^T X < 0 \mid Y = +1\right) P\left(Y = +1\right) \\
&\ + P\left(Y \cdot w^T X < 0 \mid Y = -1\right) P\left(Y = -1\right) \\
= &\ P\left(w^T X^+ < 0\right) P\left(Y = +1\right) + P\left(w^T X^- > 0\right) P\left(Y = -1\right) \\
= &\ P\left(w^T X^+ \leq 0\right) P\left(Y = +1\right) + \left(1 - P\left(w^T X^- \leq 0\right)\right) P\left(Y = -1\right).
\end{aligned}$$

□

Based on the result of Lemma 11, in order to obtain the properties of the function $F_{error}(w)$, we are interested in the properties of the new random variable $w^T X$ which is a linear transformation of the original multivariate random variable $X = (X_1, \cdots, X_n)$. In particular, the smoothness of the expected risk can be guaranteed if the CDF (cumulative distribution function) of the new random variable $w^T X$ be a smooth function.

In general, if $(X_1, \cdots, X_n)$ is a multivariate random variable with any arbitrary continuous distribution $f_{X_1,\cdots,X_n}(x_1, \cdots, x_n)$, then any mapping $Z = g\left(X_1, \cdots, X_n\right)$ is also a random variable with a probabilistic behavior in terms of that of $(X_1, \cdots, X_n)$. In particular, the distribution of Z depends on the functions $f_{X_1,\cdots,X_n}$ and g.

Formally, if we write $z = g(x_1, \cdots, x_n)$, the function $g(x_1, \cdots, x_n)$ defines a mapping from the original sample space of $(X_1, \cdots, X_n)$, call it $\{\mathcal{X}_1 \times \cdots \times \mathcal{X}_n\}$, to a new sample space $\mathcal{Z}$, the sample space of the random variable Z. That is,

$$g(x_1, \cdots, x_n) : \{\mathcal{X}_1 \times \cdots \times \mathcal{X}_n\} \to \mathcal{Z}.$$

The following describes a technique for finding the distribution of a transformation of a multivariate random variable, [9].

Definition 1. *If we define a region in space* $\{\mathcal{X}_1 \times \cdots \times \mathcal{X}_n\}$ *such that* $g(x_1, \cdots, x_n) \leq z$*, then we can compute the probability that* $g(x_1, \cdots, x_n) \leq z$*, i.e.,* $P\left(g(x_1, \cdots, x_n) \leq z\right)$

by integrating the density function $f_{X_1,\cdots,X_n}(x_1,\cdots,x_n)dx_1dx_2\cdots dx_n$ *over this region, i.e.,*

$$\begin{aligned} F_Z(z) &= P(Z \leq z) \\ &= P(g(X) \leq z) \\ &= P\left(\{x_1 \in \mathcal{X}_1, \cdots, x_n \in \mathcal{X}_n : g(x_1,\cdots,x_n) \leq z\}\right) \\ &= \int \int_{\{x_1 \in \mathcal{X}_1,\cdots,x_n \in \mathcal{X}_n \ : \ g(x_1,\cdots,x_n) \leq z\}} \cdots \int f_{X_1,\cdots,X_n}(x_1,\cdots,x_n)dx_1dx_2\cdots dx_n, \end{aligned}$$

[9].

Based on the above definition, depending on the choice of g, it is sometimes possible to obtain a tractable expression of the distribution of the new random variable Z. However, in the case of normal distribution there are some appealing properties which can be applied in our analysis.

The family of "multivariate" normal distributions is closed under linear transformations and linear combinations of random variables. In what follows we state Theorem 3.3.3 from [59], which shows the closure property of the multivariate normal distribution under linear transformation.

Theorem 14. *If* $X \sim \mathcal{N}(\mu, \Sigma)$ *and* $Z = CX + b$*, where* C *is any given* $m \times n$ *real matrix and* b *is any* $m \times 1$ *real vector, then* $Z \sim \mathcal{N}\left(C\mu + b, C\Sigma C^T\right)$.

Proof. The proof can be found in [59]. □

Corollary 5. *Considering Lemma 11 and Theorem 14, we can conclude that the expected risk is a smooth function, if the data points obey a normal distribution.*

In the following theorem we derive the closed form expression for the expected risk under the Gaussian assumption.

Theorem 15. *Suppose that the random variables from both positive and negative classes are obeying normal distributions, so that*

$$X^+ \sim \mathcal{N}\left(\mu^+, \Sigma^+\right) \quad \textit{and} \quad X^- \sim \mathcal{N}\left(\mu^-, \Sigma^-\right). \tag{4.8}$$

Then, we have

$$F_{error}(w) = P(Y=+1)\left(1-\phi\left(\mu_{Z^+}/\sigma_{Z^+}\right)\right) + P(Y=-1)\phi\left(\mu_{Z^-}/\sigma_{Z^-}\right), \tag{4.9}$$

where $\mu_{Z^+} = w^T\mu^+$, $\sigma_{Z^+} = \sqrt{w^T\Sigma^+ w}$, $\mu_{Z^-} = w^T\mu^-$, *and* $\sigma_{Z^-} = \sqrt{w^T\Sigma^- w}$, *and* ϕ *is the CDF of the standard normal distribution, so that* $\phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}t^2)dt$, *for* $\forall x \in \mathbb{R}$.

Proof. Let us define the new random variables Z^+ and Z^- as the following

$$Z^+ = w^T X^+ \quad \text{and} \quad Z^- = w^T X^-.$$

If X^+ and X^- are defined as (4.8), then by using Theorem 14 we have

$$\begin{aligned} Z^+ &\sim \mathcal{N}\left(w^T\mu^+, w^T\Sigma^+ w\right) \quad \text{and} \\ Z^- &\sim \mathcal{N}\left(w^T\mu^-, w^T\Sigma^- w\right). \end{aligned}$$

Then, by using Lemma 11 we conclude the following

$$\begin{aligned} F_{error}(w) &= P\left(Y \cdot w^T X < 0\right) \\ &= \left(1-\phi\left(\mu_{Z^+}/\sigma_{Z^+}\right)\right) P\left(Y=+1\right) + \phi\left(\mu_{Z^-}/\sigma_{Z^-}\right) P\left(Y=-1\right). \end{aligned}$$

where $\mu_{Z^+} = w^T\mu^+$, $\sigma_{Z^+} = \sqrt{w^T\Sigma^+ w}$, $\mu_{Z^-} = w^T\mu^-$, and $\sigma_{Z^-} = \sqrt{w^T\Sigma^- w}$. □

In what follows we aim to obtain the explicit derivative of $F_{error}(w)$ over w. To this end, first let us state the first derivative of the cumulative function $\phi\left(f(w)\right)$, where $f(w) = w^T\hat{\mu}/\sqrt{w^T\hat{\Sigma}w}$.

Lemma 12. *The first derivative of the cumulative function*

$$\phi\left(f(w)\right) = \int_{-\infty}^{f(w)} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}t^2\right) dt, \quad \textit{with} \quad f(w) = \frac{w^T\hat{\mu}}{\sqrt{w^T\hat{\Sigma}w}}$$

is

$$\nabla_w \phi(f(w)) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{w^T\hat{\mu}}{\sqrt{w^T\hat{\Sigma}w}}\right)^2\right)\left(\frac{\sqrt{w^T\hat{\Sigma}w}\cdot\hat{\mu} - \frac{w^T\hat{\mu}}{\sqrt{w^T\hat{\Sigma}w}}\cdot\hat{\Sigma}w}{w^T\hat{\Sigma}w}\right).$$

Proof. Note that based on the chain rule we have

$$\frac{d}{dw}\phi\left(f(w)\right) = \phi'(f(w))f'(w). \tag{4.10}$$

By substituting

$$\phi'(x) = \frac{d}{dx}\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}t^2\right)dt = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}x^2\right)$$

and

$$f'(w) = \frac{\sqrt{w^T\hat{\Sigma}w}\cdot\hat{\mu} - \frac{w^T\hat{\mu}}{\sqrt{w^T\hat{\Sigma}w}}\cdot\hat{\Sigma}w}{w^T\hat{\Sigma}w}$$

in (4.10) we conclude the result. □

Theorem 16. *Using the result of Lemma 12, the derivative of the smooth function* $F_{error}(w)$ *is defined as*

$$\begin{aligned}\nabla_w F_{error}(w) = \; & P\left(Y=-1\right)\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{w^T\mu^-}{\sqrt{w^T\Sigma^-w}}\right)^2\right)\\ & \cdot\left(\frac{\sqrt{w^T\Sigma^-w}\cdot\mu^- - \frac{w^T\mu^-}{\sqrt{w^T\Sigma^-w}}\cdot\Sigma^-w}{w^T\Sigma^-w}\right)\\ & - P\left(Y=+1\right)\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{w^T\mu^+}{\sqrt{w^T\Sigma^+w}}\right)^2\right)\\ & \cdot\left(\frac{\sqrt{w^T\Sigma^+w}\cdot\mu^+ - \frac{w^T\mu^+}{\sqrt{w^T\Sigma^+w}}\cdot\Sigma^+w}{w^T\Sigma^+w}\right).\end{aligned}$$

For the rest of this section, we show that $F_{AUC}(w)$ is a smooth function and derive its closed form expression under the Gaussian assumption. First, let us

restate (4.21) using probabilistic interpretation, e.g.,

$$\begin{aligned} F_{AUC}(w) &= 1 - \bar{F}_{AUC}(w) \\ &= 1 - \mathbb{E}_{\mathcal{X}^+,\mathcal{X}^-}\left[1\left[f\left(X^+;w\right) > f\left(X^-;w\right)\right]\right] \\ &= 1 - P\left(w^T X^+ > w^T X^-\right) \\ &= 1 - P\left(w^T\left(X^- - X^+\right) < 0\right). \end{aligned} \tag{4.11}$$

As in the case of $F_{error}(w)$, the smoothness of $F_{AUC}(w)$ follows from the smoothness of the CDF of $w^T(X^- - X^+)$. We will also use *Corollary 3.3.1* from [59], stated as what follows.

Theorem 17. *If two $d-$dimensional random vectors X^+ and X^- have a joint multivariate normal distribution, such that*

$$\begin{pmatrix} X^+ \\ X^- \end{pmatrix} \sim \mathcal{N}(\mu, \Sigma), \tag{4.12}$$

$$\text{where } \mu = \begin{pmatrix} \mu^+ \\ \mu^- \end{pmatrix} \quad \text{and} \quad \Sigma = \begin{pmatrix} \Sigma^{++} & \Sigma^{+-} \\ \Sigma^{-+} & \Sigma^{--} \end{pmatrix}.$$

Then, the marginal distributions of X^+ and X^- are normal distributions with the following properties

$$X^+ \sim \mathcal{N}\left(\mu^+, \Sigma^{++}\right) \quad \text{and} \quad X^- \sim \mathcal{N}\left(\mu^-, \Sigma^{--}\right).$$

Proof. The proof can be found in [59]. □

Further, we need to use *Corollary 3.3.3* in [59], as is stated in the following.

Theorem 18. *Consider two random vectors X^+ and X^-, as is defined in (4.12), then for any vector $w \in \mathbb{R}^d$, we have*

$$Z = w^T\left(X^- - X^+\right) \sim \mathcal{N}(\mu_Z, \sigma_Z), \tag{4.13}$$

where

$$\begin{aligned} \mu_Z &= w^T \left(\mu^- - \mu^+\right) \quad \textit{and} \\ \sigma_Z &= \sqrt{w^T \left(\Sigma^{--} + \Sigma^{++} - \Sigma^{-+} - \Sigma^{+-}\right) w}. \end{aligned} \tag{4.14}$$

Proof. The proof can be found in [59]. □

Now, in what follows, we derive the formula for $F_{AUC}(w)$ under the Gaussian assumption.

Theorem 19. *If two random vectors X^+ and X^-, have a joint normal distribution as is defined in Theorem 17, then we have*

$$F_{AUC}(w) = 1 - \phi\left(\frac{\mu_Z}{\sigma_Z}\right), \tag{4.15}$$

where ϕ is the CDF of the standard normal distribution, so that $\phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}t^2)dt$, for $\forall x \in \mathbb{R}$ and μ_Z and σ_Z are defined in (4.14).

Proof. By using (4.11) and Theorem 18 we have

$$\begin{aligned} F_{AUC}(w) &= 1 - P(w^T \left(X^- - X^+\right) < 0) \\ &= 1 - P\left(Z \leq 0\right) \\ &= 1 - P\left(\frac{Z - \mu_Z}{\sigma_Z} \leq \frac{-\mu_Z}{\sigma_Z}\right) \\ &= 1 - \phi\left(\frac{\mu_Z}{\sigma_Z}\right), \end{aligned}$$

where the random variable Z is defined in (4.13), with the stated mean and standard deviation in (4.14). □

In Theorem 19, since the CDF of the standard normal distribution $\phi(\cdot)$ is a smooth function, we can conclude that for linear classifiers, the corresponding $F_{AUC}(w)$ is a smooth function of w. Moreover, it is possible to compute the derivative of this function, if the first and second moments of the normal distribution are known, as is stated in the following theorem.

Theorem 20. *Using the result of Lemma 12, and the symmetric property of $\phi(\cdot)$, the derivative of the smooth function $F_{AUC}(w)$ is defined as*

$$\nabla_w F_{AUC}(w) = -\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{w^T\hat{\mu}}{\sqrt{w^T\hat{\Sigma}w}}\right)^2\right) \cdot \left(\frac{\sqrt{w^T\hat{\Sigma}w}\cdot\hat{\mu} - \frac{w^T\hat{\mu}}{\sqrt{w^T\hat{\Sigma}w}}\cdot\hat{\Sigma}w}{w^T\hat{\Sigma}w}\right).$$

where $\hat{\mu} = \mu^- - \mu^+$ and $\hat{\Sigma} = \Sigma^{--} + \Sigma^{++} - \Sigma^{-+} - \Sigma^{+-}$.

In the next section, we will apply the classical gradient descent with backtracking line search to optimize the expected risk and the expected AUC directly and compare the results of this optimization to optimizing $F_{log}(w)$ and $F_{hinge}(w)$, respectively. We apply our method to standard data sets for which Gaussian assumption may not hold. It is important to note that our proposed method relies on the assumption that $w^T X$ and $w^T\left(X^- - X^+\right)$ are Gaussian random variables with moments that are derived from the moments of the original distribution of X. In [19], it is shown that the distribution of the sums of partially dependent random variables approach normal distribution under some conditions of the dependency. Based on these results we believe that while the data itself may not be Gaussian, the random variables $w^T X$ and $w^T\left(X^- - X^+\right)$ may have a nearly normal distribution whose CDF is well approximated by the CDF in Theorems 15 and 18, respectively. To support our observation further, we compared the linear classifiers obtained by our proposed methods to those obtained by LDA which is a well-known method to produce linear classifiers under the Gaussian assumption (see Appendix A). We observed that the accuracy obtained by the LDA classifiers is significantly worse than that of obtained by either our approach or by optimizing surrogate loss function.

4.4 Prediction Error as Smooth Function in the Case of Data Sets with any Arbitrary Distribution

In this section, we address this question that under which conditions one can directly optimize the expected error for data sets with any arbitrary distribution. First, we need to restate the classical Central Limit Theorem (CLT) ([3, Theorem 27.1]).

Theorem 21 (classical CLT)**.** *Consider a sequence of independent and identically distributed (i.i.d) random variables $X_1, \cdots, X_n$ drawn from distributions with expected values μ and finite variance σ^2, i.e.,*

$$\mathbb{E}[X_i] = \mu \quad \textit{and} \quad Var[X_i] = \sigma^2 < \infty.$$

If $S_n = \sum_{i=1}^{n} X_i$, then

$$\lim_{n\to\infty} \left(\frac{S_n - n\mu}{\sigma\sqrt{n}} \right) \to \mathcal{N}(0, 1).$$

Proof. The proof can be found in [3]. □

The conventional CLT stated in the preceding theorem has three restrictions: the random variables should be independent, with identical distribution and finite variance.

4.4.1 Directly Optimizing Expected Error for Data with Not Identically Distributed Independent Features

Consider the following theorem ([3, Theorem 27.3]), which weakens the requirement of identical distribution while strengthening the requirement of the finite variance.

Theorem 22 (Lyapunov CLT)**.** *Suppose that the sequence $X_1, \cdots, X_n$ is independent with*

$$\mathbb{E}[X_i] = 0 \quad \textit{and} \quad \mathbb{E}[X_i^2] = \sigma_i^2 < \infty.$$

Defining the variable $S_n = \sum_{i=1}^{n} X_i$ *and its variance* $s_n^2 = \sum_{i=1}^{n} \sigma_i^2$*, if for some positive constant* δ*, the following holds*

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{s_n^{2+\delta}} \mathbb{E}\left[|X_i|^{2+\delta}\right] = 0, \tag{4.16}$$

then as $n \to \infty$ *the distribution of* S_n/s_n *converges to the standard normal distribution.*

Proof. The proof can be found in [3]. □

A practical example ([3, Example 27.4]), which satisfies condition (4.16) and consequently Theorem 22 can be applied on, is an independent and uniformly bounded sequence $X_1, X_2, \cdots$ which has mean 0. If K bounds X_i, for $i = 1, \cdots, n$, and variance s_n goes to ∞ when $n \to \infty$, then

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{s_n^3} \mathbb{E}\left[|X_i|^3\right] \leq \lim_{n\to\infty} \sum_{i=1}^{n} \frac{K\mathbb{E}\left[X_i^2\right]}{s_n^3} = \lim_{n\to\infty} \frac{K}{s_n} \to 0,$$

which is condition (4.16) for $\delta = 1$. Thus, using Theorem 22, we conclude that the distribution of S_n/s_n converges to the standard normal distribution when $n \to \infty$.

First, let us assume that the features (X_i for $i = 1, \cdots, d$) in the input vector $X = (X_1, X_2, \cdots, X_d)$ are independent from each other. On the other hand, since each feature has been extracted from a different property of our samples, they are obeying not identically distributions. We assume that these are distributions with finite second moments. As is mentioned in ([3, Page 359]), without loss of generality, we can assume that all features are centered at 0, thus have finite values as variance, i.e.,

$$\mathbb{E}[X_i] = 0 \quad \text{and} \quad \mathbb{E}[X_i^2] = \sigma_i^2 < \infty.$$

If we define the weight vector $w = (w_1, \cdots, w_d)$, then after multiplying each component X_i with w_i we will have

$$\mathbb{E}[w_i X_i] = 0 \quad \text{and} \quad \mathbb{E}[w_i^2 X_i^2] = w_i^2 \sigma_i^2 < \infty.$$

Let us define $\bar{S}_d = \sum_{i=1}^{d} w_i X_i$ and $\bar{s}_d^2 = Var(\bar{S}_d) = \sum_{i=1}^{d} w_i^2 \sigma_i^2$. Now if $K = \max\{|w_i X_i|\}_{i=1,\cdots,d}$, and $\lim_{d\to\infty} \bar{s}_d \to \infty$, then we have

$$\lim_{d\to\infty} \sum_{i=1}^{d} \frac{1}{\bar{s}_d^3} \mathbb{E}\left[|w_i X_i|^3\right] \leq \lim_{d\to\infty} \sum_{i=1}^{d} \frac{K\mathbb{E}\left[w_i^2 X_i^2\right]}{\bar{s}_d^3} = \lim_{d\to\infty} \frac{K}{\bar{s}_d} \to 0,$$

which means, for $\delta = 1$, the "Lyapunov condition" stated in (4.16) is satisfied. It means, in this setting, the distribution of the random variable $\bar{S}_d/\bar{s}_d$ converges to the standard normal distribution. Based on the preceding discussion, we can conclude our main result as what follows.

Theorem 23. *Consider two random vectors $X^+ = (X_1^+, \cdots, X_d^+)$ and $X^- = (X_1^-, \cdots, X_d^-)$ individually have independent components (positive/negative class has independent features) and*

$$\begin{aligned} &\mathbb{E}[X_i^+] = \mu_i^+, \quad Var[X_i^+] = {\sigma_i^+}^2 < \infty, \quad \forall i = 1, \cdots, d, \quad \textit{and} \\ &\mathbb{E}[X_i^-] = \mu_i^-, \quad Var[X_i^-] = {\sigma_i^-}^2 < \infty, \quad \forall i = 1, \cdots, d. \end{aligned}$$

In other words, X^+ and X^- have means $\mu^+ = [\mu_1^+, \cdots, \mu_d^+]$ and $\mu^- = [\mu_1^-, \cdots, \mu_d^-]$ and covariances Σ^+ and Σ^-, which are respectively diagonal matrices of vectors $[{\sigma_1^+}^2, \cdots, {\sigma_d^+}^2]$ and $[{\sigma_1^-}^2, \cdots, {\sigma_d^-}^2]$.

If K^+ and K^- bound $(X_i^+ - \mu_i^+)$ and $(X_i^- - \mu_i^-)$, respectively for all $i = 1, \cdots, d$, and $\sum_{i=1}^{d} w_i^2 {\sigma_i^+}^2$ and $\sum_{i=1}^{d} w_i^2 {\sigma_i^-}^2$ go to ∞ when $d \to \infty$, then for a bounded weight vector w, i.e., $W = \max\{|w_i|\}_{i=1,\cdots,d}$, we have

$$\lim_{d\to\infty} F_{error}(w) = P(Y = +1)\left(1 - \phi\left(\mu_{Z^+}/\sigma_{Z^+}\right)\right) + P(Y = -1)\phi\left(\mu_{Z^-}/\sigma_{Z^-}\right), \quad (4.17)$$

where $\mu_{Z^+} = w^T\mu^+$, $\sigma_{Z^+} = \sqrt{w^T\Sigma^+ w}$, $\mu_{Z^-} = w^T\mu^-$, and $\sigma_{Z^-} = \sqrt{w^T\Sigma^- w}$, and ϕ is the CDF of the standard normal distribution, so that $\phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}t^2)dt$, for $\forall x \in \mathbb{R}$.

Proof. We restrict our analysis to positive class, and any result can be extended to negative class as well. Defining $\hat{Z}^+ = w^T X^+ - w^T\mu^+$ we have

$$\mathbb{E}[\hat{Z^+}_i] = 0 \quad \text{and} \quad \mathbb{E}[\hat{Z^+}_i^2] = w_i^2 {\sigma_i^+}^2 < \infty.$$

Using the boundedness of vectors $(X^+ - \mu^+)$ and w, and defining $Var(\hat{Z^+}) = (\bar{s}_d^+)^2 = \sum_{i=1}^d w_i^2 {\sigma_i^+}^2 = w^T \Sigma^+ w$ we have

$$\lim_{d\to\infty} \sum_{i=1}^{d} \frac{1}{(\bar{s}_d^+)^3} \mathbb{E}\left[|\hat{Z^+}_i|^3\right] \leq \lim_{d\to\infty} \sum_{i=1}^{d} \frac{WK^+ \mathbb{E}[\hat{Z^+}_i^2]}{(\bar{s}_d^+)^3} = \lim_{d\to\infty} \frac{WK^+}{\bar{s}_d^+} \to 0,$$

which means for $\delta = 1$ the Lyapunov condition stated in (4.16) is satisfied. Thus, the distribution of the random variable $\hat{Z^+}/\bar{s}_d^+ = (w^T X^+ - w^T \mu^+)/\sqrt{w^T \Sigma^+ w}$ converges to the standard normal distribution. Similarly, with the same preceding analysis we can prove the same property for random variable $\hat{Z^-}/\bar{s}_d^- = (w^T X^- - w^T \mu^-)/\sqrt{w^T \Sigma^- w}$. Now, using the definition of the expected error provided in §4.3 we have

$$\begin{aligned}
F_{error}(w) = &\ P(Y \cdot w^T X < 0) \\
= &\ P\left(w^T X^+ \leq 0\right) P\left(Y = +1\right) + \left(1 - P\left(w^T X^- \leq 0\right)\right) P\left(Y = -1\right) \\
= &\ P\left(w^T X^+ - w^T \mu^+ \leq -w^T \mu^+\right) P\left(Y = +1\right) \\
&\ + \left(1 - P\left(w^T X^- - w^T \mu^- \leq -w^T \mu^-\right)\right) P\left(Y = -1\right) \\
= &\ P\left(\frac{w^T X^+ - w^T \mu^+}{\sqrt{w^T \Sigma^+ w}} \leq -\frac{w^T \mu^+}{\sqrt{w^T \Sigma^+ w}}\right) P\left(Y = +1\right) \\
&\ + \left(1 - P\left(\frac{w^T X^- - w^T \mu^-}{\sqrt{w^T \Sigma^- w}} \leq -\frac{w^T \mu^-}{\sqrt{w^T \Sigma^- w}}\right)\right) P\left(Y = -1\right).
\end{aligned}$$

Therefor, in limit we will have

$$\begin{aligned}
\lim_{n\to\infty} F_{error}(w) = &\ \phi(-\mu_{Z^+}/\sigma_{Z^+}) P\left(Y = +1\right) + (1 - \phi(-\mu_{Z^-}/\sigma_{Z^-})) P\left(Y = -1\right) \\
= &\ (1 - \phi(\mu_{Z^+}/\sigma_{Z^+})) P\left(Y = +1\right) + \phi(\mu_{Z^-}/\sigma_{Z^-}) P\left(Y = -1\right).
\end{aligned}$$

□

4.4.2 Directly Optimizing Expected Error for Data with Not Identically Distributed Dependent Features

First, let us define an $m-$dependent sequence of random variables. This is the case that "the random variables temporally far apart from one another are nearly independent" [3].

Definition 2 ($m-$dependent sequence)**.** *Consider a sequence of one-dimensional random variables, i.e.,*

$$X_1, X_2, \cdots . \tag{4.18}$$

If for some constant m the inequality $s > r + m$ implies that the two sets

$$\{X_1, X_2, \cdots, X_r\} \quad \textit{and} \quad \{X_s, X_{s+1}, \cdots, X_n\}$$

are independent, then the sequence (4.18) is said to be $m-$dependent [20] .

Based on the above definition, if the sequence (4.18) is 0-dependent, it is equivalent to independence. In other words, 0-dependent means that any number of blocks of successive terms of (4.18) are independent, whenever the first index of each block is greater than the last index of the preceding block.

In the following, we are stating some results of [20] as what follows.

Lemma 13. *Let (4.18) be $m-$dependent and $\mathbb{E}[X_i] = 0$, and $\mathbb{E}[X_i^2] < \infty, \quad \forall i = 1, 2, \cdots$; we define*

$$A_i = \mathbb{E}[X_{i+m}^2] + 2\sum_{j=1}^{m} \mathbb{E}[X_{i+m-j} X_{i+m}], \quad \forall i = 1, 2, \cdots . \tag{4.19}$$

Then for $s > m$

$$\mathbb{E}[X_{i+1} + \cdots + X_{i+s}]^2 = \mathbb{E}[X_{i+1} + \cdots + X_{i+s-1}]^2 + A_{i+s-m}, \quad \forall i = 0, 1, \cdots ,$$

so that

$$\mathbb{E}[X_{i+1} + \cdots + X_{i+s}]^2 = \mathbb{E}[X_{i+1} + \cdots + X_{i+m}]^2 + \sum_{h=1}^{s-m} A_{i+h}, \quad \forall i = 0, 1, \cdots ;$$

where $s > m$.

Proof. The proof can be simply justify by utilizing the definition of A_i, stated in (4.19). □

Theorem 24 (CLT for $m-$dependent not identically distributed random variables)**.** *Let* (4.18) *be an* $m-$*dependent sequence of random variables such that*
(a) $\mathbb{E}[X_i] = 0$ *and* $\mathbb{E}[|X_i|^3] \leq \gamma < \infty, \quad \forall i = 1, 2, \cdots$.
(b) $\lim_{p\to\infty} p^{-1} \sum_{h=1}^{p} A_{i+h} = A$ *exists, uniformly for all* $i = 0, 1, \cdots$.
Then, as $n \to \infty$ *the random variable* $n^{-\frac{1}{2}}(X_1 + \cdots + X_n)$ *has a limiting normal distribution with mean* 0 *and variance* A.

Proof. The proof can be found in [20]. □

In this section, we aim to prove the same result as (4.17), while relaxing the assumption of independent features as we had in §4.4.1. To this end, we use one of the variants of CLT presented in Theorem 24 in the preceding section.

Here, we need to assume that we have a $m-$dependent sequence of features, in which we have blocks of features with size m so that the components of two consecutive blocks are independent from each other, i.e.,

$$\underbrace{X_1, \cdots, X_m}, \underbrace{X_{m+1}, \cdots, X_{2m}}, \underbrace{X_{2m+1}, \cdots, X_{3m}}, \cdots, \underbrace{X_{km+1}, \cdots, X_{km}}, \cdots .$$

Condition (a) in Theorem 24 can be satisfied easily. In terms of condition (b), suppose $i = 0$, then we need to have $\lim_{p\to\infty} (A_1 + A_2 + \cdots + A_p)/p = A$, where

$$
\begin{aligned}
A_1 &= Var(X_{m+1}^2) + 2\left(\underbrace{Cov(X_{m+1}X_m) + Cov(X_{m+1}X_{m-1}) + \cdots + Cov(X_{m+1}X_1)}_{=0}\right), \\
A_2 &= Var(X_{m+2}^2) + 2\left(Cov(X_{m+2}X_{m+1}) + \underbrace{Cov(X_{m+2}X_m) + \cdots + Cov(X_{m+2}X_2)}_{=0}\right), \\
A_3 &= Var(X_{m+3}^2) \\
&\quad + 2\left(Cov(X_{m+3}X_{m+2}) + \cdots + \underbrace{Cov(X_{m+3}X_m) + \cdots + Cov(X_{m+3}X_3)}_{=0}\right), \\
&\quad \vdots \\
A_m &= Var(X_{2m}^2) + 2\left(Cov(X_{2m}X_{2m-1}) + \cdots + Cov(X_{2m}X_{m+1}) + \underbrace{Cov(X_{2m}X_m)}_{=0}\right).
\end{aligned}
$$

Based on the definition of the covariance of N number of dependent random variables, i.e.,

$$
Var\left(\sum_{i=1}^{N} X_i\right) = \sum_{i=1}^{N} Var\left(X_i\right) + \sum_{i \neq j} Cov\left(X_i, X_j\right),
$$

we can conclude that

$$
A_1 + \cdots + A_m = Var\left(X_{m+1} + \cdots + X_{2m}\right).
$$

In other words, condition (b) requires that the average of the variance of each block, when the number of blocks (or number of features) goes to ∞, converges to A. When condition (a) and (b) satisfy, Theorem 24 shows that the distribution of the random variable $S_n = X_1 + \cdots + X_n$ converges to a normal distribution with mean 0 and variance nA.

Along similar discussion as §4.4.1, we can present similar representation to (4.17) for the expected risk.

Theorem 25. *If two random vectors* $X^+ = (X_1^+, \cdots, X_d^+)$ *and* $X^- = (X_1^-, \cdots, X_d^-)$ *individually have independent blocks of features with size* m *and*

$$\mathbb{E}[X_i^+] = \mu_i^+, \quad Var[X_i^+] < \infty, \quad \forall i = 1, \cdots, d, \quad \textit{and}$$
$$\mathbb{E}[X_i^-] = \mu_i^-, \quad Var[X_i^-] < \infty, \quad \forall i = 1, \cdots, d.$$

In other words, X^+ *and* X^- *have means* $\mu^+ = [\mu_1^+, \cdots, \mu_d^+]$ *and* $\mu^- = [\mu_1^-, \cdots, \mu_d^-]$ *and bounded variances.*

Let K^+ *and* K^- *bound* $(X_i^+ - \mu_i^+)$ *and* $(X_i^- - \mu_i^-)$*, respectively for all* $i = 1, \cdots, d$ *and assume that we have a bounded weight vector* w*, i.e.,* $W = \max\{|w_i|\}_{i=1,\cdots,d}$.

In this setting, if $\lim_{p\to\infty} p^{-1} \sum_{h=1}^{p} A_{i+h}^+ = A^+$ *and* $\lim_{p\to\infty} p^{-1} \sum_{h=1}^{p} A_{i+h}^- = A^-$*, where*

$$A_i^+ = w_{i+m}^2 Var(X_{i+m}^+) + 2\sum_{j=1}^{m} w_{i+m-j} w_{i+m} Cov(X_{i+m-j}^+ X_{i+m}^+), \quad \forall i = 1, 2, \cdots, \quad \textit{and}$$
$$A_i^- = w_{i+m}^2 Var(X_{i+m}^-) + 2\sum_{j=1}^{m} w_{i+m-j} w_{i+m} Cov(X_{i+m-j}^- X_{i+m}^-), \quad \forall i = 1, 2, \cdots,$$

then

$$\lim_{n\to\infty} F_{error}(w) = (1 - \phi(\mu_{Z^+}/\hat{\sigma}_{Z^+}))P(Y = +1) + \phi(\mu_{Z^-}/\hat{\sigma}_{Z^-})P(Y = -1).$$

where $\mu_{Z^+} = w^T\mu^+$*,* $\hat{\sigma}_{Z^+} = \sqrt{nA^+}$*,* $\mu_{Z^-} = w^T\mu^-$*, and* $\sigma_{Z^-} = \sqrt{nA^-}$*, and* ϕ *is the CDF of the standard normal distribution, so that* $\phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}t^2)dt$*, for* $\forall x \in \mathbb{R}$.

Proof. We restrict our analysis to positive class, and any result can be extended to negative class as well. Defining $\hat{Z^+} = w^T X^+ - w^T\mu^+$ and using the assumption that the components of $(X^+ - \mu^+)$ and w are bounded, we conclude

$$\mathbb{E}[\hat{Z^+}_i] = 0 \quad \text{and} \quad \mathbb{E}[|\hat{Z^+}_i|^3] < \infty.$$

Thus, the distribution of the random variable $\hat{Z^+}/\sqrt{nA^+} = (w^T X^+ - w^T\mu^+)/\sqrt{nA^+}$

converges to the standard normal distribution. Similarly, with the same preceding analysis we can prove the same property for random variable $\hat{Z^-}/\sqrt{nA^-} = (w^T X^- - w^T \mu^-)/\sqrt{nA^-}$. Now, using the definition of the expected error we will have

$$\begin{aligned}
F_{error}(w) = &\ P(Y \cdot w^T X < 0) \\
= &\ P\left(w^T X^+ \leq 0\right) P\left(Y = +1\right) + \left(1 - P\left(w^T X^- \leq 0\right)\right) P\left(Y = -1\right) \\
= &\ P\left(w^T X^+ - w^T \mu^+ \leq -w^T \mu^+\right) P\left(Y = +1\right) \\
&\ + \left(1 - P\left(w^T X^- - w^T \mu^- \leq -w^T \mu^-\right)\right) P\left(Y = -1\right) \\
= &\ P\left(\frac{w^T X^+ - w^T \mu^+}{\sqrt{nA^+}} \leq -\frac{w^T \mu^+}{\sqrt{nA^+}}\right) P\left(Y = +1\right) \\
&\ + \left(1 - P\left(\frac{w^T X^- - w^T \mu^-}{\sqrt{nA^-}} \leq -\frac{w^T \mu^-}{\sqrt{nA^-}}\right)\right) P\left(Y = -1\right).
\end{aligned}$$

Which in limit we have

$$\begin{aligned}
\lim_{n \to \infty} F_{error}(w) = &\ \phi(-\mu_{Z^+}/\hat{\sigma}_{Z^+}) P\left(Y = +1\right) + (1 - \phi(-\mu_{Z^-}/\hat{\sigma}_{Z^-})) P\left(Y = -1\right) \\
= &\ (1 - \phi(\mu_{Z^+}/\hat{\sigma}_{Z^+})) P\left(Y = +1\right) + \phi(\mu_{Z^-}/\hat{\sigma}_{Z^-}) P\left(Y = -1\right).
\end{aligned}$$

□

4.5 Online AUC Optimization

4.5.1 Online Binary Classification Framework

First let us briefly introduce the concept of online learning in binary linear classification problems. Let $x \in \mathbb{R}^d$ denotes the input vector of features which is associated with a unique label $y \in \{+1, -1\}$, as the output. Online binary classification performs in a sequence of rounds, in which, at each round t, the algorithm observes a new instance x_t, and predicts its label to be $+1$ or -1. After predicting the label

of the new observed point x_t, at the end of each round t, the true label y_t is revealed. Now, by using the true label versus the predicted one, we can measure the performance of the learning process provided by our algorithm, so far. Then, the algorithm utilizes the new observed pair (x_t, y_t) to improve its prediction policy for the upcoming rounds $t+1, t+2, \cdots$.

Here, the main purpose of the learning process of the algorithm is to obtain the weight vector $w \in \mathbb{R}^d$ defining the linear classifier $w^T x$. Thus, the goal of the algorithm is to incrementally learn the vector of weights w over the rounds (the algorithm keeps w in its internal memory and update it from round to round). Let w_t denote the weight vector used by the algorithm on round t and $y_t \cdot w_t^T x_t$ denote the margin obtained by the algorithm on round t. In this learning process, both sign and the magnitude of the margin value matter. A positive margin value means that the algorithm made a correct prediction in which $y_t = sign(y_t \cdot w_t^T x_t)$. On the other hand, the magnitude $|w_t^T x_t|$ provides the degree of confidence in predicting the label of x_t, in which a classifier with high level of prediction confidence is the matter of interest.

Here, we consider hinge loss, as an instantaneous loss, which the algorithm suffers from at the end of each round t, i.e.,

$$\ell_t(w_t; x_t, y_t) = \begin{cases} 0, & \text{if } \; y_t \cdot w_t^T x_t \geq 1, \\ 1 - y_t \cdot w_t^T x_t, & \text{otherwise}, \end{cases}$$

or equivalently

$$\ell_t(w_t; x_t, y_t) = \max\{0, 1 - y_t \cdot w_t^T x_t\}.$$

The algorithms used to perform on online binary classification problems aim to minimize the cumulative squared loss, i.e., $\sum_{t=1}^{T} \ell_t^2$ over a given sequence of sample points with length T.

Passive-Aggressive method stated in Algorithm 10, as the simplest algorithm has

been used in online binary classification, solves the following constrained optimization problem to obtain the new weight vector w_{t+1} at the end of round t,

$$w_{t+1} \leftarrow \arg\min_{w \in \mathbb{R}^d} \frac{1}{2}\|w - w_t\|^2, \quad s.t. \quad \ell_t(w_t; x_t, y_t) = 0, \tag{4.20}$$

with the following closed form solution [15],

$$w_{t+1} = w_t + \tau_t y_t x_t, \quad \text{where} \quad \tau_t = \frac{\ell_t}{\|x_t\|^2}.$$

Algorithm 10 Passive-Aggressive Algorithm for Binary Classification

1: Initialize $w_1 \in \mathbb{R}^d$.
2: **for** $t = 1, 2, \cdots$ **do**
3: Receive instance $x_t \in \mathbb{R}^d$.
4: Predict $\hat{y}_t = \text{sign}(w_t^T x_t)$.
5: Receive correct label $y_t = \{+1, -1\}$.
6: Compute instantaneous loss $\ell_t = \max\{0, 1 - y_t \cdot w_t^T x_t\}$.
7: Update $w_{t+1} \leftarrow w_t + \tau_t y_t x_t$, where $\tau_t = \ell_t / \|x_t\|^2$.

In conventional online binary classification problems, the ultimate purpose of the algorithm is to optimize the prediction error, while in the case of learning from skewed data sets, utilizing AUC (Area Under ROC Curve) as another performance measure is more useful. As we discussed earlier, in online binary classification algorithms, the overall loss is defined as the sum of losses experienced by individual training samples over passed rounds. However, in binary classification, in the case of AUC optimization, the loss function is defined as the sum of the pairwise losses between two instances from different classes. Thus, optimizing AUC in online setting turns to a challenging problem, in the sense that the algorithm needs to remember all the received training examples from the initial round up to the current round.

In what follows, we summarize the work has been done on AUC optimization in online setting.

4.5.2 Online AUC Optimization

Given a training data set $\mathcal{S} := \{(x_i, y_i) \ : \ i = 1, \cdots, T\}$, we divide it into two sets:

$$\mathcal{S}^+ := \{x : (x, y) \in \mathcal{S}, \ y = +1\} := \{x_i^+ \ : \ i = 1, \cdots, T^+\}, \quad \text{where} \quad x_i^+ \in \mathbb{R}^d \quad \text{and}$$
$$\mathcal{S}^- := \{x : (x, y) \in \mathcal{S}, \ y = -1\} := \{x_j^- \ : \ j = 1, \cdots, T^-\}, \quad \text{where} \quad x_j^- \in \mathbb{R}^d.$$

Hence, $\mathcal{S}^+$ and $\mathcal{S}^-$ are the sets of all positive and negative samples in $\mathcal{S}$–they contain only inputs x, instead of pairs (x, y)–in which $|\mathcal{S}^+| = T^+$ and $|\mathcal{S}^-| = T^-$.

The AUC value of a linear classifier $w^T x$ is defined as:

$$\begin{aligned} F_{AUC}\left(w; \mathcal{S}^+, \mathcal{S}^-\right) &= \frac{\sum_{i=1}^{T^+} \sum_{j=1}^{T^-} 1\left[w^T x_i^+ > w^T x_j^-\right]}{T^+ \cdot T^-} \\ &= 1 - \frac{\sum_{i=1}^{T^+} \sum_{j=1}^{T^-} 1\left[w^T x_i^+ \leq w^T x_j^-\right]}{T^+ \cdot T^-}. \end{aligned} \tag{4.21}$$

where $1[\cdot]$ is an indicator function that takes value 1 if the argument is true and 0 otherwise. Thus, maximizing $F_{AUC}\left(w; \mathcal{S}^+, \mathcal{S}^-\right)$ is equivalent to minimizing

$$\sum_{i=1}^{T^+} \sum_{j=1}^{T^-} 1\left[w^T x_i^+ \leq w^T x_j^-\right].$$

The indicator function $1[\cdot]$ in (4.21) can be replaced with a hinge loss as a convex surrogate loss, i.e.,

$$\ell(w; x_i^+ - x_j^-) = \max\{0, 1 - w^T(x_i^+ - x_j^-)\},$$

and the final objective function is considered to to be:

$$\sum_{i=1}^{T^+} \sum_{j=1}^{T^-} \ell(w; x_i^+ - x_j^-),$$

In [64], the above formulation is rewritten into a sum of losses for individual

instances, so that

$$\sum_{t=1}^{T} \mathcal{L}_t(w),$$

where $\mathcal{L}_t(w)$ is defined as

$$\mathcal{L}_t(w) = 1[y_t = +1]h_t^+(w) + 1[y_t = -1]h_t^-(w),$$

so that

$$h_t^+(w) = \sum_{t'=1}^{t-1} 1[y_t = -1]\ell(w; x_t - x_t') \quad \textit{and}$$
$$h_t^-(w) = \sum_{t'=1}^{t-1} 1[y_t = +1]\ell(w; x_t' - x_t).$$

Now, if we utilize the Passive-Agressive method stated in (4.20), then the main challenge is the need of storing all received example points so far. In [64], in order to solve this issue, the idea of "caching a small number of positive and negative received sample points" is presented.

Different representation of AUC function proposed in §4.3 solves the challenge of applying online setting on AUC optimization, as discussed in the following.

4.5.3 Directly Optimizing AUC in Online Setting

As is discussed in §4.3, AUC is a smooth function of the weight vector w. Hence, any gradient-based optimization approach can be utilized to optimize this function. To this end, having approximate first and second moments of the distribution of data points is required.

Practically speaking, if we are given a set of training data points $\mathcal{S} := \{(x_i, y_i) \ : \ i = 1, \cdots, T\}$, where $x_i \in \mathbb{R}^d$ and $y_i \in \{+1, -1\}$ are the corresponding labels, then we just need to use these data points and compute the sample mean and covariance of the feature vectors x_i's. Thus, no matter which kind of algorithm we are using in our learning process, we do not need to store data points over the iterations.

In online setting, after receiving the new pair of data (x_t, y_t), we can update the mean and the covariance accordingly, and update the vector w.

Algorithm 11 Directly Optimizing Online AUC

Initialize $w_1 \in \mathbb{R}^d$, $x^+ = \mathbf{0}$, $x^- = \mathbf{0}$, $t^+ = 0$, and $t^- = 0$.
for $t = 1, 2, \cdots$ **do**
  Receive instance (x_t, y_t).
  if $y_t = +1$ **then**
    $x^+ \leftarrow x_t$ and $t^+ \leftarrow t^+ + 1$.
    Update $\hat{\mu}_t^+$, and $\hat{\Sigma}_t^{++}$, and $\hat{\Sigma}_t^{+-}$, and consequently $\hat{\Sigma}_t^{-+}$.
    Set $\hat{\mu}_t^- \leftarrow \hat{\mu}_{t-1}^-$ and $\hat{\Sigma}_t^{--} \leftarrow \hat{\Sigma}_{t-1}^{--}$.
  else
    $x^- \leftarrow x_t$ and $t^- \leftarrow t^- + 1$.
    Update $\hat{\mu}_t^-$, and $\hat{\Sigma}_t^{--}$ and $\hat{\Sigma}_t^{-+}$, and consequently $\hat{\Sigma}_t^{+-}$.
    Set $\hat{\mu}_t^+ \leftarrow \hat{\mu}_{t-1}^+$ and $\hat{\Sigma}_t^{++} \leftarrow \hat{\Sigma}_{t-1}^{++}$.
  Update $w_{t+1} \leftarrow \arg\max_{w \in \mathbb{R}^d} F_{AUC_t}(w)$.

In Algorithm 11, $F_{AUC_t}(w)$ denotes the AUC function obtained by using the sample mean and covariance after receiving new point x_t. Sample mean and covariance are indicated by $\hat{\mu}$ and $\hat{\Sigma}$ (with different signs and indices).

Now, the question is, on round t, after obtaining new pair (x_t, y_t), how can we update the sample mean and covariance (as a key procedure in Algorithm 11) efficiently, without storing previously seen data points up to round $t-1$.

First, note that considering two random variables X^+ and X^-, we have

$$\mu^+ = \mathbb{E}[X^+], \quad \Sigma^{++} = Cov(X^+, X^+), \quad Cov(X^+, X^-), \quad \text{and}$$
$$\mu^- = \mathbb{E}[X^-], \quad \Sigma^{--} = Cov(X^-, X^-), \quad Cov(X^-, X^+) = Cov(X^+, X^-)^T,$$

where $Cov(X, Y) = \mathbb{E}\left[(X - \mathbb{E}[X])\,(Y - \mathbb{E}[Y])^T\right]$.

Practically speaking, since we do not have a through knowledge about the distribution of X^+ and X^-, we use sample mean and covariance, using the sample set $\mathcal{S}$. Generally, when we are given n number of points, the sample mean and covariance

are computed as what follows,

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{and} \quad \hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T.$$

In our setting, suppose we are at the end of round $t-1$, and we have recorded t^+ and t^-, as the number of positive and negative points seen so far. On round t, after receiving new point (x_t, y_t), depending on the sign of y_t, we need to update either positive moments or negative ones. Suppose $y_t = +1$ and $x_t = x_t^+$, so we need to update $\hat{\mu}_{t^+}^+$ and $\hat{\Sigma}_{t^+}^{++}$ as the sample mean and covariance of the positive class as well as $\hat{\Sigma}_t^{+-}$ as the sample covariance between positive and negative classes ($\hat{\Sigma}_t^{-+}$ can be updated consequently), i.e.,

$$\hat{\mu}_{t^+}^+ = \frac{(t^+ - 1)\hat{\mu}_{t^+-1}^+ + x_t^+}{t^+}, \qquad \hat{\Sigma}_{t^+}^{++} = \frac{(t^+ - 1)\hat{\Sigma}_{t^+-1}^{++} + (x_t^+ - \hat{\mu}_t^+)(x_t^+ - \hat{\mu}_t^+)^T}{t^+}, \quad \text{and}$$
$$\hat{\Sigma}_t^{+-} = \frac{(t-1)\hat{\Sigma}_{t-1}^{+-} + (x_t^+ - \hat{\mu}_t^+)(x^- - \hat{\mu}_t^-)^T}{t}. \tag{4.22}$$

In Algorithm 11, one can store more than one point from each classes (as is done in [64]) and approximate more accurate covariance matrix $\hat{\Sigma}_t^{+-}$ between positive and negative classes in (4.22). However, intuitively speaking, correlation between positive and negative features is not supposed to be significant, thus, the covariance between the positive and negative classes would not have any main effect on the proposed method. Thus, storing more than one point from each class seems to be a redundant effort. Along this discussion, in computational results provided in the next section, we assumed that $\hat{\Sigma}^{+-}$ and $\hat{\Sigma}^{-+}$ are zero matrices and we still obtained a linear classifier with high AUC value via optimizing the smooth function provided in (4.15).

4.5.3.1 Analyzing the Performance of Directly Optimizing AUC in Online Setting

As this is the case in conventional online optimization approaches, in order to measure the performance of the proposed online method in optimizing AUC, we can compare it with the offline setting in which we have access to all data points in the beginning of the algorithm. To this end, we define the following regret,

$$R_T = F_{AUC_T}(w_T) - \min_{w \in \mathbb{R}^d} F_{AUC_T}(w). \tag{4.23}$$

As we explained earlier, in the process of directly optimizing AUC in online setting, after receiving a new point x_t we need to update the sample mean and covariance, as we presented in (4.22). Thus, we compute the bound on the difference between the sample moments obtained with the access to the whole data points and the sample moments obtained at the end of each round.

Let us just consider the positive class, any result can be similarly extended to the negative class as well. If we have access to the whole positive data points with size T^+, then the sample mean and covariance will have the form of

$$\hat{\mu}^+ = \frac{1}{T^+} \sum_{t=1}^{T^+} x_t^+ \quad \text{and} \quad \hat{\Sigma}^{++} = \frac{1}{T^+} \sum_{t=1}^{T^+} (x_t^+ - \hat{\mu}^+)(x_t^+ - \hat{\mu}^+)^T.$$

By defining the sample mean and covariance $\hat{\mu}^+_{T^+}$ and $\hat{\Sigma}^{++}_{T^+}$, as the ones obtained after the last round T^+, we can define the following bounds

$$R^{mean}_{T^+} = \hat{\mu}^+ - \hat{\mu}^+_{T^+} \quad \text{and} \quad R^{var}_{T^+} = \hat{\Sigma}^{++} - \hat{\Sigma}^{++}_{T^+}.$$

Based on the process of updating $\hat{\mu_t}^+$ at each round, we simply have $R^{mean}_{T^+} = 0$. In terms of computing $R^{var}_{T^+}$, first we need to consider the pattern. Let us obtain the regret $R^{var}_{T^+}$ for $T^+ = 2$ and $T^+ = 3$.

For $T^+ = 2$ we have

$$\hat{\Sigma}^{++} = \frac{1}{2}\left((x_1^+ - \hat{\mu}_2^+)(x_1^+ - \hat{\mu}_2^+)^T + (x_2^+ - \hat{\mu}_2^+)(x_2^+ - \hat{\mu}_2^+)^T\right) \quad \text{and}$$
$$\hat{\Sigma}_2^{++} = \frac{1}{2}\left((x_1^+ - \hat{\mu}_1^+)(x_1^+ - \hat{\mu}_1^+)^T + (x_2^+ - \hat{\mu}_2^+)(x_2^+ - \hat{\mu}_2^+)^T\right),$$

then

$$\begin{aligned} R_2^{var} &= \hat{\Sigma}^{++} - \hat{\Sigma}_2^{++} \\ &= \frac{1}{2}\left((x_1^+ - \hat{\mu}_2^+)(x_1^+ - \hat{\mu}_2^+)^T - (x_1^+ - \hat{\mu}_1^+)(x_1^+ - \hat{\mu}_1^+)^T\right). \end{aligned}$$

Similarly for $T^+ = 3$ we have

$$\hat{\Sigma}^{++} = \frac{1}{3}\left((x_1^+ - \hat{\mu}_3^+)(x_1^+ - \hat{\mu}_3^+)^T + (x_2^+ - \hat{\mu}_3^+)(x_2^+ - \hat{\mu}_3^+)^T + (x_3^+ - \hat{\mu}_3^+)(x_3^+ - \hat{\mu}_3^+)^T\right) \quad \text{and}$$
$$\begin{aligned} \hat{\Sigma}_3^{++} &= \frac{1}{3}\left(2\hat{\Sigma}_2^{++} + (x_3^+ - \hat{\mu}_3^+)(x_3^+ - \hat{\mu}_3^+)^T\right) \\ &= \frac{1}{3}\left((x_1^+ - \hat{\mu}_1^+)(x_1^+ - \hat{\mu}_1^+)^T + (x_2^+ - \hat{\mu}_2^+)(x_2^+ - \hat{\mu}_2^+)^T + (x_3^+ - \hat{\mu}_3^+)(x_3^+ - \hat{\mu}_3^+)^T\right), \end{aligned}$$

which results

$$\begin{aligned} R_3^{var} = &\ \hat{\Sigma}^{++} - \hat{\Sigma}_3^{++} \\ = &\ \frac{1}{3}\left((x_1^+ - \hat{\mu}_3^+)(x_1^+ - \hat{\mu}_3^+)^T - (x_1^+ - \hat{\mu}_1^+)(x_1^+ - \hat{\mu}_1^+)^T\right) \\ &+ \frac{1}{3}\left((x_2^+ - \hat{\mu}_3^+)(x_2^+ - \hat{\mu}_3^+)^T - (x_2^+ - \hat{\mu}_2^+)(x_2^+ - \hat{\mu}_2^+)^T\right). \end{aligned} \tag{4.24}$$

If for all $t = t, \cdots, T^+$ we define α_t and β_t as

$$\hat{\mu}_{T^+}^+ = \hat{\mu}_t^+ + \alpha_t \quad \text{and} \quad x_t = \hat{\mu}_t^+ + \beta_t \quad \forall t = 1, \cdots, T^+,$$

then we can conclude

$$\begin{aligned}
R_{T^+}^{var} &= \frac{1}{T^+}\left(\sum_{t=1}^{T^+}(x_t^+ - \hat{\mu}_{T^+}^+)(x_t^+ - \hat{\mu}_{T^+}^+)^T - \sum_{t=1}^{T^+}(x_t^+ - \hat{\mu}_t^+)(x_t^+ - \hat{\mu}_t^+)^T\right) \\
&= \frac{1}{T^+}\left(\sum_{t=1}^{T^+}(x_t^+ - \hat{\mu}_t^+ - \alpha_t)(x_t^+ - \hat{\mu}_t^+ - \alpha_t)^T - \sum_{t=1}^{T^+}(x_t^+ - \hat{\mu}_t^+)(x_t^+ - \hat{\mu}_t^+)^T\right) \\
&= \frac{1}{T^+}\sum_{t=1}^{T^+}\left(-(x_t^+ - \hat{\mu}_t^+)\alpha_t^T - \alpha_t(x_t^+ - \hat{\mu}_t^+)^T + \alpha_t\alpha_t^T\right) \\
&= \frac{1}{T^+}\sum_{t=1}^{T^+}\left(-\beta_t\alpha_t^T - \alpha_t\beta_t^T + \alpha_t\alpha_t^T\right).
\end{aligned}$$

If we define $M = \max\{\|\alpha_t\|, \|\beta_t\|\}_{t=1,\cdots,T^+}$, then we have $\|R_{T^+}^{var}\| = 3M^2$.

4.6 Numerical Experiments

First we compare the performance of the linear classifiers obtained by directly optimizing the expected risk versus those obtained by regularized logistic regression. We use gradient descent as is stated in Algorithm 12.

Algorithm 12 Gradient Descent with Backtracking Line Search

1: Initialize $w_0 \in \mathbb{R}^d$, and choose $c \in (0,1)$, and $\beta \in (0,1)$.
2: **for** $i = 1, 2, \cdots$ **do**
3: Choose α_k^0 and define $\alpha_k := \alpha_k^0$.
4: Compute the trial point

$$w_k^{trial} \leftarrow w_{k-1} - \alpha_k \nabla_w F(w_k).$$

5: **while** $F(w_k^{trial}) > F(w_k) + c\alpha_k\|\nabla_w F(w_k)\|^2$ **do**
6: Set $\alpha_k \leftarrow \beta\alpha_k$.
7: Compute $w_k^{trial} \leftarrow w_{k-1} - \alpha_k \nabla_w F(w_k)$.
8: Set $w_k \leftarrow w_k^{trial}$.

We perform Algorithm 12 to $F(w) = F_{error}(w)$ defined in (4.9), and to $F(w) = F_{log}(w)$ defined in (4.3).

$F_{error}(w)$ is a nonconvex function, thus in an attempt to avoid bad local minima we generate a starting point as follows

$$w_0 = \frac{\bar{w}_0}{\|\bar{w}_0\|}, \quad \text{where} \quad \bar{w}_0 = \mu^+ - \frac{\mu^{-T}\mu^+}{\|\mu^-\|^2}\mu^-.$$

We set the parameters of Algorithm 12 as $c = 10^{-4}$, $\beta = 0.5$, and $\sigma_k^0 = 1$ and terminate the algorithm when $\|\nabla_w F(w_k)\| < 10^{-7}\|\nabla_w F(w_0)\|$ or when the maximum number of iterations 250 is reached. For the logistic regression, the regularization parameter in (4.3) is set as $\lambda = 1/n$, and the initial point w_0 is selected randomly, since the optimization problem is convex.

All experiments, implemented in Python 2.7.11, were performed on a computational cluster consisting of 16-cores AMD Operation, 2.0 GHz nodes with 32 Gb of memory.

We considered artificial data sets generated from normal distribution and real data sets. We have generated 9 different artificial Gaussian data sets of various dimensions using random first and second moments, summarized in Table 4.1. Moreover, we generated data sets with some percentage of outliers by swapping a specified percentage of positive and negative examples.

The corresponding numerical results are summarized in Table 4.2, where we used 80 percent of the data points as the training data and the rest as the test data. The reported average accuracy is based on 20 runs for each data set. When minimizing $F_{error}(w)$, we used the exact moments from which the data set was generated, and also the approximate moments, empirically obtained through the sampled data points.

We see in Table 4.2 that, as expected, minimizing $F_{error}(w)$ using the exact moments produces linear classifiers with superior performance overall, while minimizing $F_{error}(w)$ using approximate moments outperforms minimizing $F_{log}(w)$. In Table 4.2, the bold numbers indicate the average testing accuracy attained by minimizing $F_{error}(w)$ using approximate moments, when this accuracy is significantly better

Table 4.1: Artificial data sets statistics. d : number of features, n : number of data points,
P^+, P^- : prior probabilities, out : percentage of outlier data points.

Name	d	n	P^+	P^-	$out\%$
$data_1$	500	5000	0.05	0.95	0
$data_2$	500	5000	0.35	0.65	5
$data_3$	500	5000	0.5	0.5	10
$data_4$	1000	5000	0.15	0.85	0
$data_5$	1000	5000	0.4	0.6	5
$data_6$	1000	5000	0.5	0.5	10
$data_7$	2500	5000	0.1	0.9	0
$data_8$	2500	5000	0.35	0.65	5
$data_9$	2500	5000	0.5	0.5	10

than that obtained by minimizing $F_{log}(w)$. Note also that minimizing $F_{error}(w)$ requires less time than minimizing $F_{log}(w)$.

Table 4.2: $F_{error}(w)$ vs. $F_{log}(w)$ minimization via Algorithm 12 on artificial data sets.

Data	$F_{error}(w)$ Minimization Exact moments		$F_{error}(w)$ Minimization Approximate moments		$F_{log}(w)$ Minimization	
	Accuracy ± std	Time (s)	Accuracy ± std	Time (s)	Accuracy ± std	Time (s)
$data_1$	0.9965±0.0008	0.25	0.9907±0.0014	1.04	0.9897±0.0018	3.86
$data_2$	0.9905±0.0023	0.26	**0.9806**±0.0032	0.86	0.9557±0.0049	13.72
$data_3$	0.9884±0.0030	0.03	**0.9745**±0.0037	1.28	0.9537±0.0048	15.79
$data_4$	0.9935±0.0017	0.63	0.9791±0.0034	5.51	0.9782±0.0031	10.03
$data_5$	0.9899±0.0026	5.68	**0.9716**±0.0048	10.86	0.9424±0.0055	28.29
$data_6$	0.9904±0.0017	0.83	**0.9670**±0.0058	5.18	0.9291±0.0076	25.47
$data_7$	0.9945±0.0019	4.79	0.9786±0.0028	32.75	0.9697±0.0031	43.20
$data_8$	0.9901±0.0013	9.96	0.9290±0.0045	119.64	0.9263±0.0069	104.94
$data_9$	0.9899±0.0028	1.02	0.9249±0.0096	68.91	0.9264±0.0067	123.85

Further, we used 19 real data sets downloaded from LIBSVM website[1] and UCI machine learning repository[2], summarized in Table 4.3. We have normalized the data sets so that each dimension has mean 0 and variance 1. The data sets from UCI machine learning repository with categorical features are transformed into grouped

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
[2] http://archive.ics.uci.edu/ml/

binary features.

Table 4.3: Real data sets statistics. d : number of features, n : number of data points, P^+, P^- : prior probabilities, AC : attribute characteristics.

Name	AC	d	n	P^+	P^-
fourclass	$[-1, 1]$, real	2	862	0.35	0.65
svmguide1	$[-1, 1]$, real	4	3089	0.35	0.65
diabetes	$[-1, 1]$, real	8	768	0.35	0.65
shuttle	$[-1, 1]$, real	9	43500	0.22	0.78
vowel	$[-6, 6]$, int	10	528	0.09	0.91
magic04	$[-1, 1]$, real	10	19020	0.35	0.65
poker	$[1, 13]$, int	11	25010	0.02	0.98
letter	$[0, 15]$, int	16	20000	0.04	0.96
segment	$[-1, 1]$, real	19	210	0.14	0.86
svmguide3	$[-1, 1]$, real	22	1243	0.23	0.77
ijcnn1	$[-1, 1]$, real	22	35000	0.1	0.9
german	$[-1, 1]$, real	24	1000	0.3	0.7
landsat satellite	$[27, 157]$, int	36	4435	0.09	0.91
sonar	$[-1, 1]$, real	60	208	0.5	0.5
a9a	binary	123	32561	0.24	0.76
w8a	binary	300	49749	0.02	0.98
mnist	$[0, 1]$, real	782	100000	0.1	0.9
colon-cancer	$[-1, 1]$, real	2000	62	0.35	0.65
gisette	$[-1, 1]$, real	5000	6000	0.49	0.51

Table 4.4 summarizes the performance comparison between the linear classifiers obtained by minimizing $F_{error}(w)$ versus $F_{log}(w)$. We used five-fold cross-validation and repeated each experiment four times, and the average test accuracy over the 20 runs are reported for each problem.

Table 4.4: $F_{error}(w)$ vs. $F_{log}(w)$ minimization via Algorithm 12 on real data sets.

Data	$\boldsymbol{F_{error}(w)}$ **Minimization**		$\boldsymbol{F_{log}(w)}$ **Minimization**	
	Accuracy ± std	Time (s)	Accuracy ± std	Time (s)
fourclass	0.8782±0.0162	0.02	0.8800±0.0147	0.12
svmguide1	**0.9735**±0.0047	0.42	0.9506±0.0070	0.28
diabetes	0.8832±0.0186	1.04	0.8839±0.0193	0.13
shuttle	0.8920±0.0015	0.01	**0.9301**±0.0019	4.05
vowel	0.9809±0.0112	0.91	0.9826±0.0088	0.11
magic04	0.8867±0.0044	0.66	0.8925±0.0041	1.75
poker	0.9897±0.0008	0.17	0.9897±0.0008	10.96
letter	0.9816±0.0015	0.01	0.9894±0.0009	4.51
segment	0.9316±0.0212	0.17	**0.9915**±0.0101	0.36
svmguide3	**0.9118**±0.0136	0.39	0.8951±0.0102	0.17
ijcnn1	0.9512±0.0011	0.01	0.9518±0.0011	4.90
german	0.8780±0.0125	1.09	0.8826±0.0159	0.62
landsat satellite	0.9532±0.0032	0.01	0.9501±0.0049	3.30
sonar	**0.8926**±0.0292	0.49	0.8774±0.0380	0.92
a9a	0.9193±0.0021	0.98	0.9233±0.0020	11.45
w8a	0.9851±0.0005	0.36	0.9876±0.004	24.16
mnist	0.9909±0.0004	3.79	0.9938±0.0004	136.83
colon-cancer	**0.9364**±0.0394	15.92	0.8646±0.0555	1.20
gisette	0.9782±0.0025	310.72	0.9706±0.0036	136.71

As we can see in Table 4.4, the linear classifier obtained by minimizing $F_{error}(w)$ has comparable test accuracy to the one obtained from minimizing $F_{log}(w)$ in 13 cases out of 19. In four cases minimizing $F_{error}(w)$ surpasses minimizing $F_{log}(w)$ in terms of the average test accuracy, while performs worse in the case of the two remaining data sets. Finally, we note that the solution time of optimizing $F_{error}(w)$ is significantly less than that of optimizing $F_{log}(w)$ when d is smaller than n.

Figure 4.1 illustrates the progress of the linear classifiers obtained through these two different approaches in terms of the average test accuracy over iterations. In Figure 4.1 we selected the data sets in which minimizing $F_{error}(w)$ has a better performance in terms of the final test accuracy compared to minimizing $F_{log}(w)$ or vice versa. We note that in two cases where optimizing $F_{error}(w)$ performs worse than minimizing $F_{log}(w)$, the algorithm achieved its best $F_{error}(w)$ value during the first few iterations and then stalled. This may be due to the inaccurate gradient or simply a local minimum. Further, an illustration of the distribution of $w^T X^+$ and $w^T X^-$ for a randomly chosen weight vector w can be found in Appendix B.

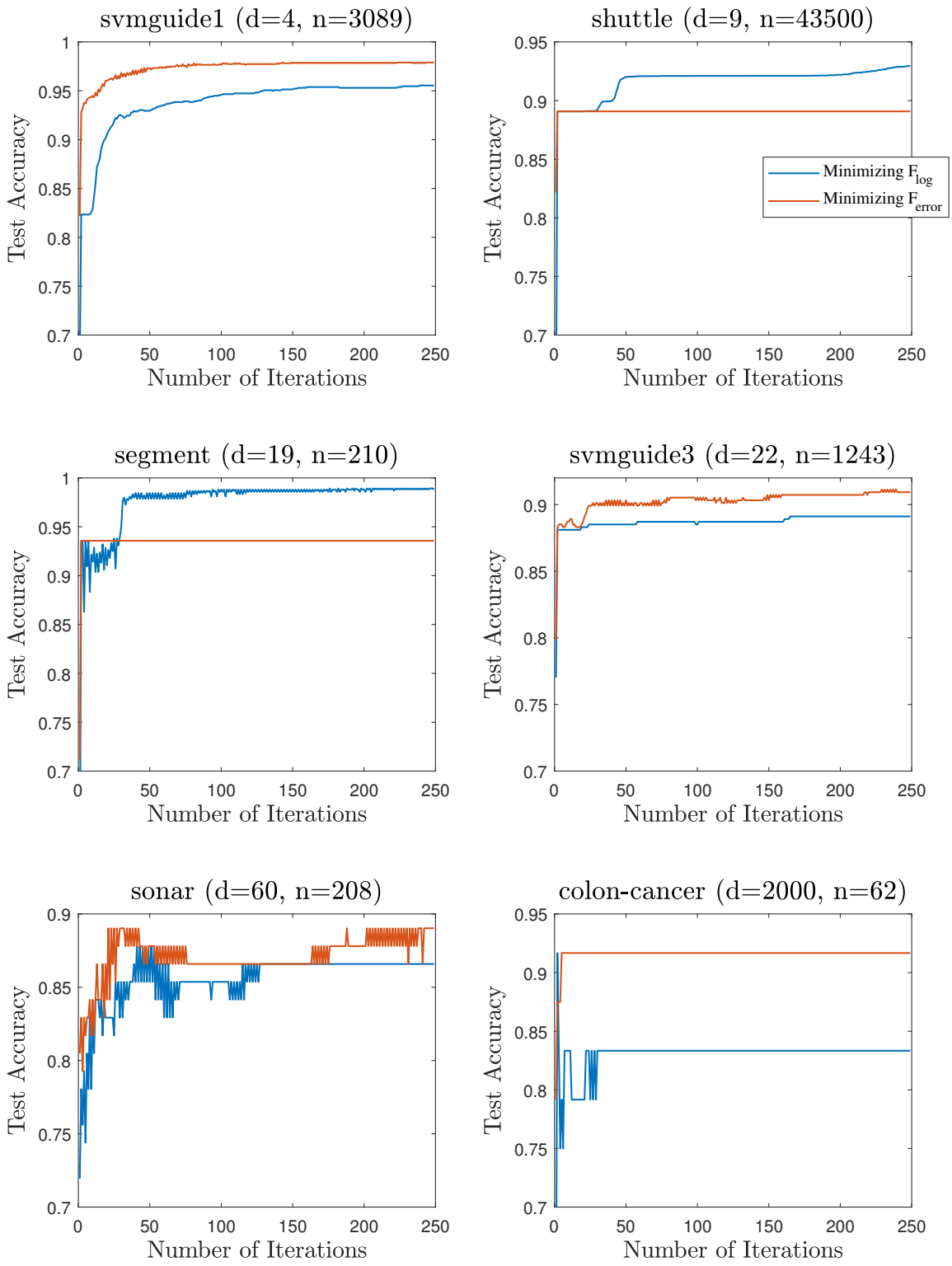


Figure 4.1: Performance of minimizing $F_{error}(w)$ vs. $F_{log}(w)$ via Algorithm 12.

We now turn to comparing the performance of linear classifiers obtained by optimizing the AUC function, e.g., $F(w) = F_{AUC}(w)$ defined in (4.15) and its approximation via pairwise hinge loss, e.g., $F(w) = F_{hinge}(w)$ as is defined in (4.6). The setting of the parameters and the type of the artificial and real data sets are the same as in Tables 4.1 and 4.3.

The results for artificial data sets are summarized in Table 4.5 as the same manner of Table 4.2, except that we report the AUC value as the performance measure of the resulting classifiers. As we can see in Table 4.5, in the process of minimizing $F_{AUC}(w)$, the only advantage of using the exact moments rather than the approximate moments is in terms of the solution time, since both approaches result in comparable average AUC values. On the other hand, the performance of the linear classifier obtained through minimizing $F_{AUC}(w)$ using approximate moments surpasses that of the classifier obtained via minimizing $F_{hinge}(w)$, both in terms of the average AUC value as well as the required solution time.

Table 4.5: $F_{AUC}(w)$ vs. $F_{hinge}(w)$ minimization via Algorithm 12 on artificial data sets.

Data	$\boldsymbol{F_{AUC}(w)}$ **Minimization** **Exact moments**		$\boldsymbol{F_{AUC}(w)}$ **Minimization** **Approximate moments**		$\boldsymbol{F_{hinge}(w)}$ **Minimization**	
	AUC ± std	Time (s)	AUC ± std	Time (s)	AUC ± std	Time (s)
$data_1$	0.9972±0.0014	0.01	**0.9941**±0.0027	0.23	0.9790±0.0089	5.39
$data_2$	0.9963±0.0016	0.01	**0.9956**±0.0018	0.22	0.9634±0.0056	159.23
$data_3$	0.9965±0.0015	0.01	**0.9959**±0.0018	0.24	0.9766±0.0041	317.44
$data_4$	0.9957±0.0018	0.02	**0.9933**±0.0022	0.83	0.9782±0.0054	23.36
$data_5$	0.9962±0.0011	0.02	**0.9951**±0.0013	0.80	0.9589±0.0068	110.26
$data_6$	0.9962±0.0013	0.02	**0.9949**±0.0015	0.82	0.9470±0.0086	275.06
$data_7$	0.9965±0.0021	0.08	**0.9874**±0.0034	4.61	0.9587±0.0092	28.31
$data_8$	0.9966±0.0008	0.07	**0.9929**±0.0017	4.54	0.9514±0.0051	104.16
$data_9$	0.9962±0.0014	0.08	**0.9932**±0.0020	4.54	0.9463±0.0085	157.62

Table 4.6 summarizes the results on real data sets, in a manner similar to Table 4.4, while, again using AUC of the resulting classifier as the performance measure. As we can see in Table 4.6, the average AUC values of the linear classifiers obtained

through minimizing $F_{AUC}(w)$ and $F_{hinge}(w)$ are comparable in 14 cases out of 19, in four cases minimizing $F_{AUC}(w)$ performs better than minimizing $F_{hinge}(w)$ in terms of the average test AUC, while their performance is worse in the remaining two cases, where the algorithm stalled after a few iterations of optimizing $F_{AUC}(w)$ as is shown in Figure 4.2. In terms of solution time, minimizing $F_{AUC}(w)$ significantly outperforms minimizing $F_{hinge}(w)$, due to the high per-iteration complexity dependence on n of $F_{hinge}(w)$ minimization.

Table 4.6: $F_{AUC}(w)$ vs. $F_{hinge}(w)$ minimization via Algorithm 12 on real data sets.

Data	$F_{AUC}(w)$ Minimization		$F_{hinge}(w)$ Minimization	
	AUC ± std	Time (s)	AUC ± std	Time (s)
fourclass	0.8362±0.0312	0.01	0.8362±0.0311	6.81
svmguide1	0.9717±0.0065	0.06	0.9863±0.0037	35.09
diabetes	0.8311±0.0311	0.03	0.8308±0.0327	12.48
shuttle	0.9872±0.0013	0.11	0.9861±0.0017	2907.84
vowel	0.9585±0.0333	0.12	**0.9765**±0.0208	2.64
magic04	0.8382±0.0071	0.11	0.8419±0.0071	1391.29
poker	0.5054±0.0224	0.11	0.5069±0.0223	1104.56
letter	0.9830±0.0029	0.12	0.9883±0.0023	121.49
segment	0.9948±0.0035	0.21	0.9992±0.0012	4.23
svmguide3	**0.8013**±0.0420	0.34	0.7877±0.0432	23.89
ijcnn1	0.9269±0.0036	0.08	0.9287±0.0037	2675.67
german	0.7938±0.0292	0.14	0.7919±0.0294	32.63
landsat satellite	**0.7587**±0.0160	0.43	0.7458±0.0159	193.46
sonar	0.8214±0.0729	0.88	**0.8456**±0.0567	2.15
a9a	0.9004±0.0039	0.92	0.9027±0.0037	15667.87
w8a	0.9636±0.0055	0.54	0.9643±0.0057	5353.23
mnist	0.9943±0.0009	0.64	0.9933±0.0009	28410.2393
colon-cancer	**0.8942**±0.1242	2.50	0.8796±0.1055	0.05
gisette	**0.9957**±0.0015	31.32	0.9858±0.0029	3280.38

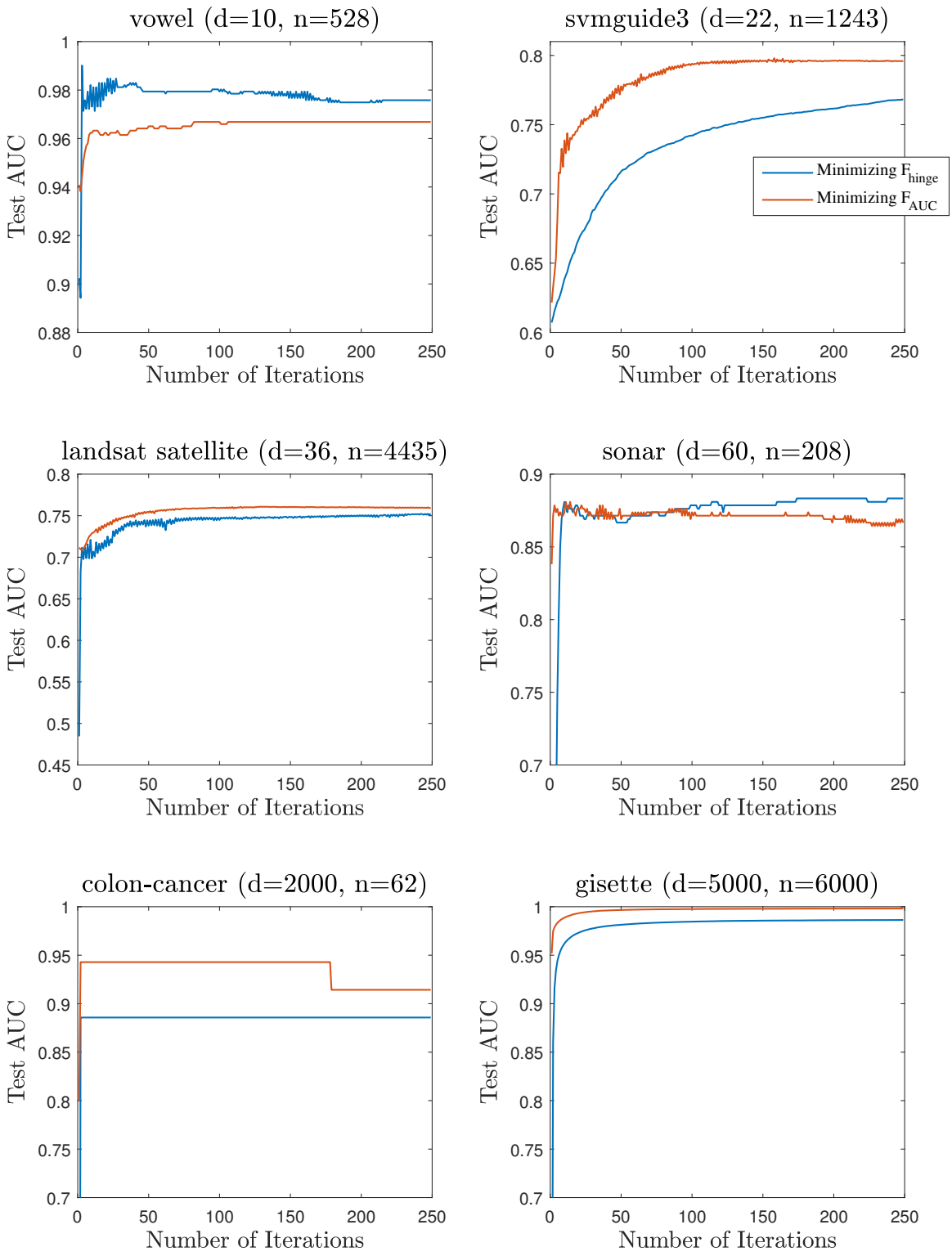


Figure 4.2: Performance of minimizing $F_{AUC}(w)$ vs. $F_{hinge}(w)$ via Algorithm 12.

4.7 Conclusion

In this work, we showed that under the Gaussian assumption, the expected prediction error and AUC of linear predictors in binary classification are smooth functions whose derivatives can be computed using the first and second moments of the related normal distribution. We then show that empirical moments of real data sets (not necessarily Gaussian) can be utilized to obtain approximate derivatives. This implies that gradient-based optimization approach can be used to optimize the prediction error and AUC function. In this work, for simplicity, we used gradient descent with backtracking line search and we demonstrated the efficiency of directly optimizing prediction error and AUC function compared to their approximations– logistic regression and pairwise hinge loss, respectively. The main advantage of these approaches is that the proposed objective functions and their derivatives are independent of the size of the data sets. Clearly more efficient second-order methods can also be utilized for optimizing these functions, which is a subject for the future research.

Appendix A

Numerical Comparison vs. LDA

In the following we provide the numerical results comparing minimizing $F_{error}(w)$ and $F_{log}(w)$ versus LDA, while using the artificial data sets as well as real data sets.

Table A.1: $F_{error}(w)$ and $F_{log}(w)$ minimization via Algorithm 12 vs. LDA on artificial data sets.

Data	$\boldsymbol{F_{error}(w)}$ **Minim. Exact moments** Accuracy ± std	$\boldsymbol{F_{error}(w)}$ **Minim. Approximate moments** Accuracy ± std	$\boldsymbol{F_{log}(w)}$ **Minim.** Accuracy ± std	**LDA** Accuracy ± std
$data_1$	0.9965±0.0008	0.9907±0.0014	0.9897±0.0018	0.9851±0.0035
$data_2$	0.9905±0.0023	**0.9806**±0.0032	0.9557±0.0049	0.9670±0.0057
$data_3$	0.9884±0.0030	**0.9745**±0.0037	0.9537±0.0048	0.9630±0.0081
$data_4$	0.9935±0.0017	0.9791±0.0034	0.9782±0.0031	0.9672±0.0049
$data_5$	0.9899±0.0026	**0.9716**±0.0048	0.9424±0.0055	0.9455±0.0074
$data_6$	0.9904±0.0017	**0.9670**±0.0058	0.9291±0.0076	0.9417±0.0085
$data_7$	0.9945±0.0019	0.9786±0.0028	0.9697±0.0031	0.9086±0.0137
$data_8$	0.9901±0.0013	0.9290±0.0045	0.9263±0.0069	0.8526±0.0182
$data_9$	0.9899±0.0028	0.9249±0.0096	0.9264±0.0067	0.8371±0.0149

Table A.2: $F_{error}(w)$ and $F_{log}(w)$ minimization via Algorithm 12 vs. LDA on real data sets.

Data	$\boldsymbol{F_{error}(w)}$ **Minimization**	$\boldsymbol{F_{log}(w)}$ **Minimization**	**LDA**
	Accuracy ± std	Accuracy ± std	Accuracy ± std
fourclass	0.8782±0.0162	0.8800±0.0147	0.7572±0.0314
svmguide1	**0.9735**±0.0047	0.9506±0.0070	0.8972±0.0159
diabetes	0.8832±0.0186	0.8839±0.0193	0.7703±0.0366
shuttle	0.8920±0.0015	**0.9301**±0.0019	0.9109±0.0027
vowel	0.9809±0.0112	0.9826±0.0088	0.9600±0.0224
magic04	0.8867±0.0044	0.8925±0.0041	0.7841±0.0093
poker	0.9897±0.0008	0.9897±0.0008	0.9795±0.0017
letter	0.9816±0.0015	0.9894±0.0009	0.9711±0.0029
segment	0.9316±0.0212	**0.9915**±0.0101	0.9617±0.0331
svmguide3	**0.9118**±0.0136	0.8951±0.0102	0.8238±0.0259
ijcnn1	0.9512±0.0011	0.9518±0.0011	0.9081±0.0029
german	0.8780±0.0125	0.8826±0.0159	0.7675±0.0275
landsat satellite	0.9532±0.0032	0.9501±0.0049	0.9061±0.0065
sonar	**0.8926**±0.0292	0.8774±0.0380	0.7622±0.0499
a9a	0.9193±0.0021	0.9233±0.0020	0.8452±0.0038
w8a	0.9851±0.0005	0.9876±0.004	0.9839±0.0012
mnist	0.9909±0.0004	0.9938±0.0004	0.9778±0.0013
colon cancer	**0.9364**±0.0394	0.8646±0.0555	0.8875±0.0985
gisette	0.9782±0.0025	0.9706±0.0036	0.5875±0.0207

Appendix B

Illustration of the Linear Transformation of an Arbitrary Random Variable

Figures B.1 and B.2 show the performance of directly optimizing expected error versus logistic regression, and the distribution of $w^T X^+$ and $w^T X^-$ for a randomly chosen weight vector w.

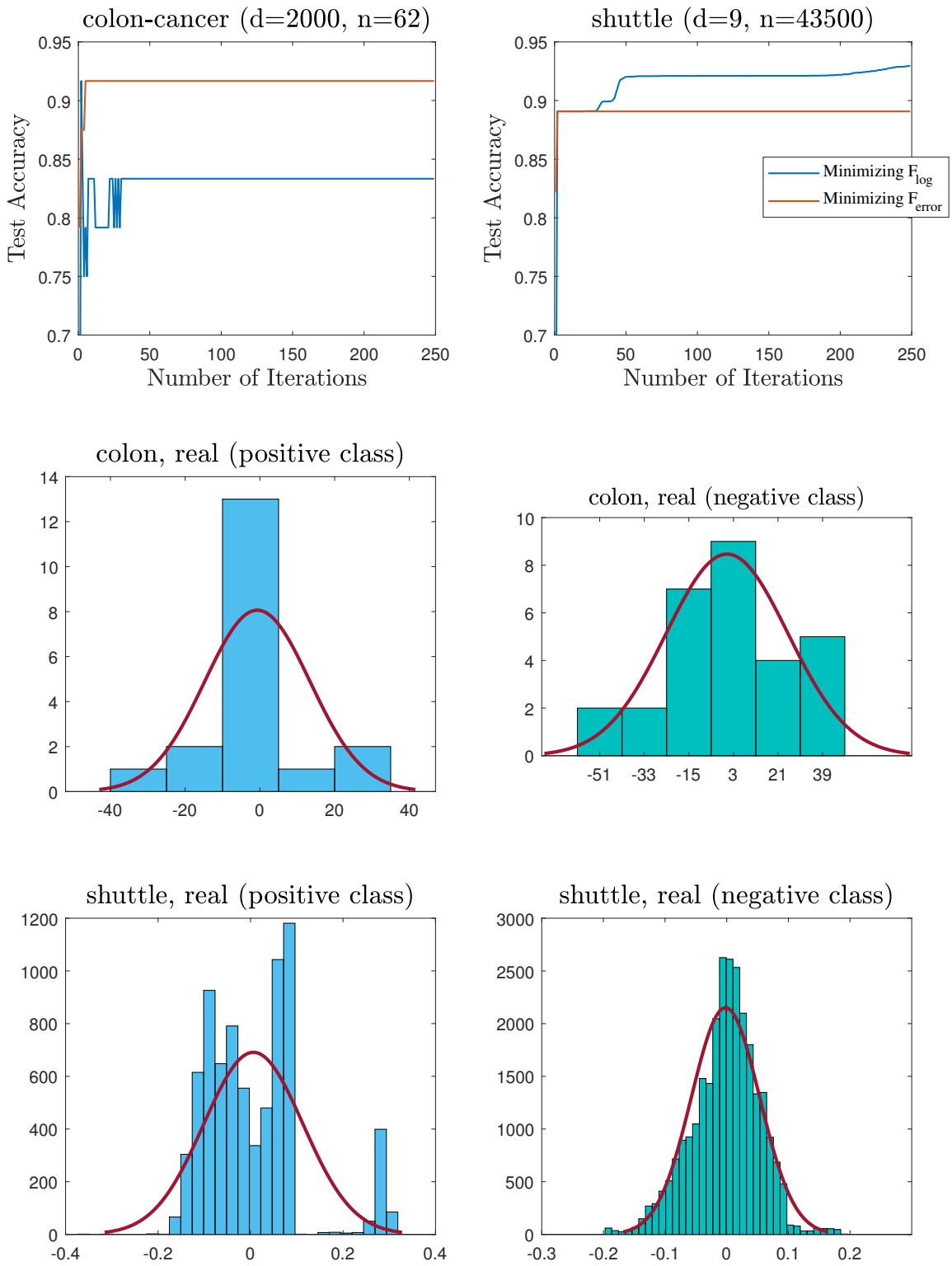


Figure B.1: Illustration of linear transformation of positive and negative variables.

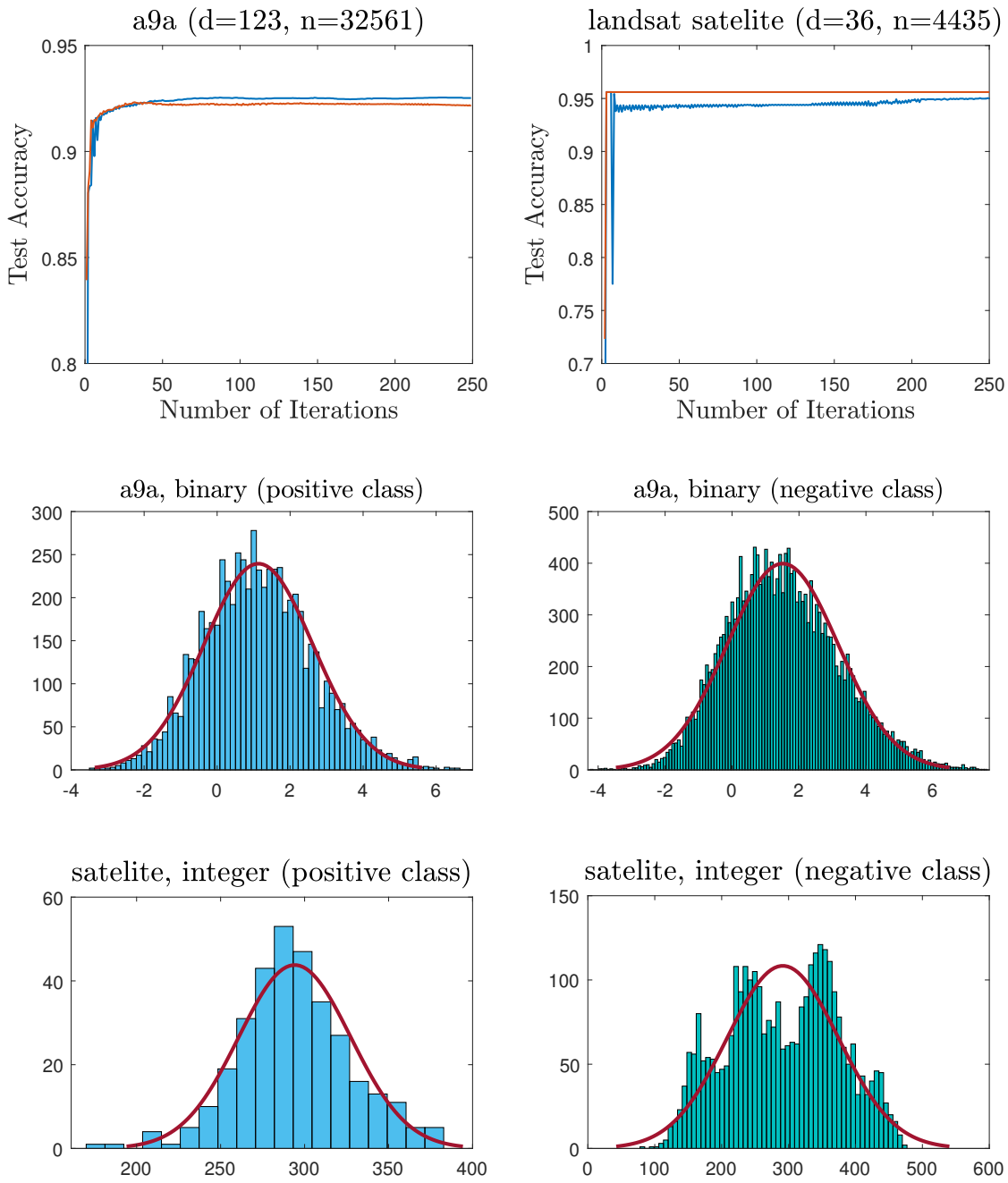


Figure B.2: Illustration of linear transformation of positive and negative variables.

Bibliography

[1] A. A. Cassioli and F. Schoen. Global optimization of expensive black-box problems with a known lower bound. *Journal of Global Optimization*, 57:177–190, 2013.

[2] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM*, 2:183–202, 2009.

[3] P. Billingsley. *Probability and Measure*. A Wiley-Interscience Publication, 1995.

[4] S. C. Billups and J. Larson. Stochastic derivative-free optimization using a trust region framework. *J. Computational Optimization and Apps*, 64(2):619–645, 2016.

[5] J. Blanchet, C. Cartis, M. Menickelly, and K. Scheinberg. Convergence rate analysis of a stochastic trust region method for nonconvex optimization. *https://arxiv.org/pdf/1609.07428*, 2016.

[6] E. Brochu, V.M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *eprint arXiv:1012.2599, arXiv.org*, 2010.

[7] R. Byrd, J. Nocedal, and F. Oztoprak. An inexact successive quadratic approximation method for convex ℓ_1-regularized optimization. *Mathematical Programming*, 157:375–396, 2016.

[8] R. H Byrd, J. Nocedal, and R. B Schnabel. Representations of quasi Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

[9] G. Casella and R.L. Berger. *Statistical Inference*. Pacific Grove, CA: Duxbury, 2, 2002.

[10] R. Chen, M. Menickelly, and K. Scheinberg. Stochastic optimization using a trust-region method and random models. *Mathematical Programming*, 169: 447–287, 2018.

[11] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79: 397–414, 1997.

[12] A. R. Conn, N. I. M. Gould, and P. T. Toint. *Trust Region Methods*. MPS/SIAM Series on Optimization. SIAM, Philadelphia, 2000.

[13] A.R. Conn, K. Scheinberg, and L.N. Vicente. *Introduction To Derivative-Free Optimization*. Society for Industrial and Applied Mathematics. Philadelphia, 2009.

[14] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20: 273–297, 1995.

[15] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[16] S. Lemeshow D. W. Hosmer. *Applied Logistic Regression*. 2nd edition, 2000.

[17] D. Drusvyatskiy and A. S. Lewis. Error bounds, quadratic growth, and linear convergence of proximal methods. To appear in Math. Oper. Res., 2017.

[18] K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. H. Hoos, and K. L. Brown. Towards an empirical foundation for assessing bayesian

optimization of hyperparameters. *In NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

[19] N. I. Fisher and P. K. Sen. The central limit theorem for dependent random variables. *in The Collected Works of Wassily Hoeffding, New York:Springer-Verlag*, pages 205–213, 1994.

[20] N. I. Fisher and P. K. Sen. The central limit theorem for dependent random variables. *in The Collected Works of Wassily Hoeffding, New York:Springer-Verlag*, pages 205–213, 1994.

[21] H. M. Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[22] J. A. Hanley and B. J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 1982.

[23] C. J. Hsieh, M. Sustik, I. Dhilon, and P. Ravikumar. Sparse inverse covariance matrix estimation using quadratic approximation. *NIPS*, pages 2330–2338, 2011.

[24] A. J. Izenman. *Modern Multivariate Statistical Techniques*. Springer Texts in Statistics, 2013.

[25] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. *https://arxiv.org/pdf/1502.07943*, 2015.

[26] K. Jiang, D. Sun, and K. Toh. An inexact accelerated proximal gradient method for large scale linearly constrained convex SDP. *SIAM*, 22:1042–1064, 2012.

[27] T. Joachims. Training linear SVMs in linear time. *In ACM SIGKDD*, pages 217–226, 2006.

[28] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *J. Optimization Theory and Apps*, 79:157–181, 1993.

[29] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *J. Global Optimization*, 13:455–492, 1998.

[30] J. D. Lee, Y. Sun, and M. A. Saunders. Proximal Newton-type methods for convex optimization for minimizing composite functions. *SIAM J. Optim.*, 24: 1420–1443, 2014.

[31] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *https://arxiv.org/abs/1603.06560*, 2016.

[32] W. J. Welch M. Schonlau and D. R. Jones. Global versus local search in constrained optimization of computer models. *In New Developments and Applications in Experimental Design*, 34:11–25, 1998.

[33] H. B. Mann and D. R. Whitney. On a test whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist*, 18:50–60, 1947.

[34] J. Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *J. Global Optimization*, 4:347–365, 1994.

[35] J. J. More and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim*, 20:172–191, 2009.

[36] A. Nemirovski and D. Yudin. Informational complexity and efficient methods for solution of convex extremal problems. *J. Wiley and Sons, New York*, 1983.

[37] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Mathematics Doklady*, 27:372–376, 1983.

[38] Y. E. Nesterov. *Introductory Lectures on Convex Programming: A Basic Course*. Springer, 2004.

[39] Y. E. Nesterov. Smooth minimization for non-smooth functions. *Mathematical Programming*, 103:127–152, 2005.

[40] Y. E. Nesterov. Gradient methods for minimizing composite objective function. *Mathematical Programming*, 140:125–161, 2013.

[41] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, NY, USA, 2nd edition, 2006.

[42] P. A. Olsen, F. Oztoprak, J. Nocedal, and S. J. Rennie. Newton-like methods for sparse inverse covariance estimation. *NIPS*, pages 764–772, 2012.

[43] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. *In IEEE Conference on Computer Vision and Pattern Recognition*, pages 130–136, 1997.

[44] M. J. D. Powell. Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. *Mathematical Programming*, 100:183–215, 2004.

[45] P. Richtarik and M. Takac. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144:1–38, 2014.

[46] C. Rudin and R. E. Schapire. Margin-based ranking and an equivalence between adaboost and rankboost. *Journal of Machine Learning Research*, 10:2193–2232, 2009.

[47] K. Scheinberg and I. Rish. A greedy coordinate ascent method for sparse inverse covariance selection problem. *SINCO, Technical Report*, 2009.

[48] K. Scheinberg and X. Tang. Practical inexact proximal quasi-Newton method with global complexity analysis. *Mathematical Programming*, 160:495–529, 2016.

[49] K. Scheinberg and Ph. L. Toint. Self-correcting geometry in model-based algorithms for derivative-free unconstrained optimization. *SIAM J. Optim*, 20(6): 3512–3532, 2010.

[50] K. Scheinberg, D. Goldfarb, and X. Bai. Fast first-order methods for composite convex optimization with backtracking. *Foundation of Mathematics*, 14:389–417, 2014.

[51] M. Schmidt, N. L. Roux, and F. Bach. Supplementary material for the paper convergence rates of inexact proximal-gradient methods for convex optimization. *NIPS*, 2011.

[52] M. Schmidt, N. L. Roux, and F. Bach. Convergence rate of inexact proximal-gradient method for convex optimization. *NIPS*, pages 1458–1466, 2011.

[53] B. Scholkopf, A. Smola, K. R. Muller, C. J. C. Burges, and V. Vapnik. Support vector methods in learning and feature extraction. *In Ninth Australian Congress on Neural Networks*, 1998.

[54] S. Shalev-Shwartz and A. Tewari. Stochastic methods for ℓ_1-regularized loss minimization. *ICML*, pages 929–936, 2009.

[55] S. Shashaani, F. S. Hashemi, and R. Pasupathy. ASTRO-DF: A class of adaptive sampling trust-region algorithms for derivative-free simulation optimization. *https://arxiv.org/pdf/1610.06506*, 2015.

[56] S. Sra, S. Nowozin, and S.J. Wright. *Optimization for Machine Learning*. Mit Pr, 2011.

[57] H. Steck. Hinge rank loss and the area under the ROC curve. *In ECML, Lecture Notes in Computer Science*, pages 347–358, 2007.

[58] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B, Methodological*, 58:267–288, 1996.

[59] Y.L. Tong. The multivariate normal distribution. *Springer Series in Statistics*, 1990.

[60] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. *Technical Report*, 2008.

[61] S. Villa, S. Salzo, L. Baldassarre, and A. Verri. Accelerated and inexact forward-backward algorithms. *SIAM*, 23:1607–1633, 2011.

[62] L. Yan, R. Dodier, M.C. Mozer, and R. Wolniewicz. Optimizing classifier performance via approximation to the wilcoxon-mann-witney statistic. *Proceedings of the Twentieth Intl. Conf. on Machine Learning, AAAI Press, Menlo Park, CA*, pages 848–855, 2003.

[63] G. X. Yuan, K. W. Chang, C. J. Hsieh, and C. J. Lin. A comparison of optimization methods and software for large-scale ℓ_1-regularized linear classification. *JMLR*, 11:3183–3234, 2010.

[64] P. Zhao, S. Hoi, R. Jin, and T. Yang. Online auc maximization. *In Proc. of the 28th Int. Conf. on Machine Learning (ICML). Omnipress*, pages 233–240, 2011.

[65] P. Zhao, S. C. H. Hoi, R. Jin, and T. Yang. Online AUC maximization. *In Proceedings of the 28th ICML, Bellevue, WA, USA*, 2011.

Biography

Hiva Ghanbari, Ph.D., joined the Department of Industrial & Systems Engineering at Lehigh University in 2013. She received her Bachelors and Masters degrees in Industrial Engineering from Sharif University of Technology, Tehran, Iran in 2012. Her research focuses on different aspects of optimization algorithms designed for machine learning problems.

Education

- Lehigh University, Ph.D., Bethlehem, PA, USA, 2013–2019.
- Sharif University of Technology, M.Sc., Tehran, Iran, 2010–2012.
- Sharif University of Technology, B.Sc., Tehran, Iran, 2006–2010.

Honors and Awards

- Van Hoesen Family Best Publication Award, Lehigh University, 2018.
- Ph.D. Student of the Year, Lehigh University, 2015.
- INFORMS Magna Cum Laude award for Lehigh University INFORMS Student Chapter, 2015.
- Rossin Doctoral Fellowship, Lehigh University, 2015–2018.
- Dean's Doctoral Student Assistantship, Lehigh University, 2013.

- Exceptionally Talented Student, awarded direct entrance to graduate studies with full scholarship by Dean of Sharif University of Technology, 2010.
- Ranked in the top 1% of the National University Entrance Examination of Mathematics Major, awarded full scholarship by Dean of Sharif University of Technology, 2006.