Journal of Scientific Computing
https://doi.org/10.1007/s10915-019-01045-7

A Two-Step Fixed-Point Proximity Algorithm for a Class of Non-differentiable Optimization Models in Machine Learning

Zheng Li[1] · **Guohui Song**[2] · **Yuesheng Xu**[2]

Received: 26 September 2018 / Revised: 22 May 2019 / Accepted: 26 August 2019

Abstract
Sparse learning models are popular in many application areas. Objective functions in sparse learning models are usually non-smooth, which makes it difficult to solve them numerically. We develop a fast and convergent two-step iteration scheme for solving a class of non-differentiable optimization models motivated from sparse learning. To overcome the difficulty of the non-differentiability of the models, we first present characterizations of their solutions as fixed-points of mappings involving the proximity operators of the functions appearing in the objective functions. We then introduce a two-step fixed-point algorithm to compute the solutions. We establish convergence results of the proposed two-step iteration scheme and compare it with the alternating direction method of multipliers (ADMM). In particular, we derive specific two-step iteration algorithms for three models in machine learning: ℓ^1-SVM classification, ℓ^1-SVM regression, and the SVM classification with the group LASSO regularizer. Numerical experiments with some synthetic datasets and some benchmark datasets show that the proposed algorithm outperforms ADMM and the linear programming method in computational time and memory storage costs.

Keywords Two-step iteration methods · Proximity algorithms · Sparse learning · Non-differentiable

This research was supported in part by the Ministry of Science and Technology of China under Grant 2016YFB0200602, by the Natural Science Foundation of China under Grants 11471013, 11771464, and by the US National Science Foundation under Grants DMS-1521661, DMS-1522332, DMS-1912958 and DMS-1939203.

✉ Yuesheng Xu
y1xu@odu.edu

Zheng Li
li_zheng_2011@163.com

Guohui Song
gsong@odu.edu

[1] School of Mathematics, and Guangdong Province Key Lab of Computational Science, Sun Yat-sen University, Guangzhou 510275, People's Republic of China

[2] Department of Mathematics and Statistics, Old Dominion University, Norfolk, VA 23529, USA

Published online: 03 September 2019

1 Introduction

Sparse learning models have received increasing attention in machine learning [2] due to the fact that models with sparsity usually bring advantages both in estimation and interpretation [8]. For example, it is well received that the ℓ^1-norm regularizer would produce sparse solutions [29]. Moreover, the LASSO regularized models have shown their great power in producing sparse solutions [29]. The group LASSO [9,34] regularization is a natural generalization of the LASSO regularization that can be seen as a group-wise ℓ^1-norm. It was shown in [9,20,34] that the group LASSO regularized model would generate a group sparse solution and would perform better when the optimal variable has a group sparse structure.

Sparse learning models are often non-smooth, which brings great challenges in solving them numerically. In particular, when both the fidelity term and the penalty term in the model are non-smooth, it is often quite challenging to efficiently solve it since no gradient information is available for these terms due to their non-smoothness. For instance, in the ℓ^1-SVM model both the hinge loss function and the ℓ^1 norm penalty are non-differentiable, which make it difficult to solve the model. A typical treatment of the non-smoothness is to use smooth (differentiable) approximations of the non-smooth terms involved in the optimization problems and solve the resulting approximate models by using the standard optimization algorithms for smooth functions, see [33] for a survey of the treatment. On the other hand, some methods consider approximate models of the dual problem [7,19]. However, solutions of the approximate models might not perform as well as solutions of the original model. We would focus on developing fast and efficient algorithms for solving a class of non-differentiable optimization models motivated from sparse learning. In general, both the fidelity term and the penalty term are non-differentiable in such models.

Difficulty of solving an optimization problem having a non-smooth fidelity term and a non-smooth penalty term lies in treating simultaneously the two non-smooth terms. The alternating direction method of multipliers (ADMM) has been used to solve an optimization problem whose objective function is a sum of two non-smooth convex functions. It provides a specific way of decoupling the two functions through introducing auxiliary variables such that at each iteration step one can minimize only a quadratic perturbation of these two functions alternatively. ADMM has been proved to be a powerful method in solving convex optimization problems [4,25]. However, ADMM may suffer from non-convergence or slow convergence for certain problems due to its fixed scheme of splitting the terms in iterations [13,21].

The goal of this paper is to develop efficient methods for solving the non-smooth optimization models directly rather than working on their smooth approximations. Following a general idea described in [13,21], we shall design a two step iteration method suitable for solving the non-smooth optimization models motivated from sparse learning. We shall use the fixed-point methodology by employing the fixed-point equation of the optimization problem expressed in terms of the proximity operators of the non-smooth functions involved in their objective functions. Specifically, we shall first characterize its solutions as fixed-points of some nonlinear mapping through employing the connection between the sub-differential and the proximity operator, following the approach developed in [1,12,13,15,21,22]. In other words, we transform the optimization problem into an equivalent fixed-point problem, and develop fast algorithms of finding such fixed points numerically. We note that such proximity operators in the fixed point reformulation would have closed forms that would reduce the computational costs. However, it turns out that it may not lead to a convergent algorithm by directly applying iterative fixed-point algorithm on this characterization. To overcome this difficulty, we further apply matrix splitting techniques to derive a two-step iteration scheme of computing the fixed-points that is proved to be convergent. Moreover, the convergence will speed up due to the usage of the fully updated two-step information. In summary, we would obtain a fast and convergent direct solver combining two-fold advantages: employing the closed forms of the proximity operators in the fixed point reformulation and using two-step information in each iteration through matrix splitting techniques according to the special structure of the operators in the fixed point reformulation.

We shall demonstrate the advantages of the proposed two-step method by comparing it with ADMM. We shall show that ADMM could be viewed as a specific way of splitting that satisfies the proposed general conditions. However, it uses only one-step information in each iteration, unlike the proposed method that uses two-step information. We remark that a carefully designed two-step iteration scheme might have a faster convergence than one-step iteration scheme, such as the seminal Nesterov's accelerated gradient method [24]. In addition to providing a more general framework than the specific splitting scheme of ADMM, the proposed two-step method has also shown superior convergence speed to ADMM. In particular, we introduce some other specific choices of splitting that is able to use more (two-step) information at each iteration than that (one-step) of ADMM. Moreover, due to its flexible choices of splitting, we might also be able to obtain some optimal splitting strategy to improve the convergence. We shall confirm the superior convergence performance in numerical experiments on several benchmark datasets. In particular, these numerical experiments show that the proposed method is 2–8 times faster than ADMM while maintaining the same prediction accuracy.

This paper is organized in seven sections. In Sect. 2, we shall introduce a non-differentiable optimization model motivated from sparse learning. We shall also present there three important examples in sparse learning that could be formulated in such a general non-differentiable optimization model. We shall focus on, in Sect. 3, developing fast and efficient numerical algorithms for solving such a model. Specifically, we shall first characterize its solutions as the fixed-points of certain non-linear operators and develop a two-step iterative scheme for computing the fixed points. The convergence results of the proposed two-step iterative scheme are also shown in the section. In Sect. 4, we shall present some specific two-step explicit algorithms based on the general two-step iterative scheme. We shall provide in Sect. 5 a detailed comparison of the proposed two-step iteration scheme with ADMM. Section 6 is devoted to numerical experiments. We shall implement the proposed method on both synthetic datasets and some benchmark real-world datasets. Finally, we draw conclusion remarks in Sect. 7.

2 A Class of Optimization Models

In this section, we consider a class of optimization models whose objective functions are a sum of two non-differentiable convex functions. The optimization problems have the following form:

$$\min\{\varphi(\boldsymbol{w}) + \psi(\mathsf{B}\boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^n\}, \tag{2.1}$$

where $\varphi : \mathbb{R}^n \to \overline{\mathbb{R}} := \mathbb{R} \cup \{\infty\}$ and $\psi : \mathbb{R}^m \to \overline{\mathbb{R}}$ are two non-differential convex functions, B is a $m \times n$ matrix. Here, $\mathbb{R}^d$ denotes the usual d-dimensional Euclidean space.

We remark that many optimization models in machine learning could be formulated in the above general form. We describe a few of them below.

2.1 ℓ^1-SVM Classification

The standard ℓ^2-norm support vector machine classification [6,30] aims at finding the best hyperplane that has the largest distance to the nearest point of any class. It turns out that this hyperplane is determined by a small fraction of the training points, that is, the support vectors. For the purpose of decreasing the number of the support vectors, the support vector machines with the ℓ^1 norm regularizer [26,30,36] is put forward. In this subsection, we describe the ℓ^1-SVM classification model and show how it can be reformulated in the general form Eq. (2.1).

We now describe the ℓ^1-SVM classification model. Given instances $\boldsymbol{x}_i \in \mathbb{R}^n$ with labels $y_i \in \{1, -1\}$, $i \in \mathbb{N}_m := \{1, 2, \ldots, m\}$, the standard ℓ^1- SVM classification is to solve the following minimization problem

$$\min \quad \left\{ \|\boldsymbol{\alpha}\|_1 + C \sum_{i \in \mathbb{N}_m} L(\boldsymbol{\alpha}, \boldsymbol{x}_i, y_i, b) : \boldsymbol{\alpha} \in \mathbb{R}^m, \;\; b \in \mathbb{R} \right\}, \tag{2.2}$$

where L is the hinge loss function given by

$$L(\boldsymbol{\alpha}, \boldsymbol{x}_i, y_i, b) := \max\left\{ 1 - y_i \left(\sum_{j \in \mathbb{N}_m} \alpha_j K(\boldsymbol{x}_j, \boldsymbol{x}_i) + b \right), 0 \right\}, \tag{2.3}$$

$K(\cdot, \cdot)$ is a given kernel function on $\mathbb{R}^n \times \mathbb{R}^n$ and $C > 0$ is the regularized parameter.

Below, we present a reformulation of model (2.2) in the general form (2.1). To this end, for $\boldsymbol{\alpha}$ and b appearing in (2.2), we introduce the vector $\boldsymbol{w} := \begin{pmatrix} \boldsymbol{\alpha} \\ b \end{pmatrix}$. We define the matrix $\mathsf{K} := [K(\boldsymbol{x}_i, \boldsymbol{x}_j) : i, j \in \mathbb{N}_m]$ and augment it to $\tilde{\mathsf{K}} := [\mathsf{K} \; \mathbf{1}]$, where $\mathbf{1} := (1, 1, \ldots, 1)^T \in \mathbb{R}^m$. We further introduce the diagonal matrix $\mathsf{Y} := \mathrm{diag}(y_i : i \in \mathbb{N}_m)$. Using the above notation, we define $\mathsf{B}_c := \mathsf{Y}\tilde{\mathsf{K}}$. Moreover, for any $\boldsymbol{w} \in \mathbb{R}^{m+1}$ and the m order identity matrix I_m, we let

$$\mathsf{A} := \begin{bmatrix} I_m & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix}_{(m+1)\times(m+1)},$$

and define

$$\varphi_l(\boldsymbol{w}) := \|\mathsf{A}\boldsymbol{w}\|_1. \tag{2.4}$$

For $\boldsymbol{s} \in \mathbb{R}^m$, we define

$$\psi_h(\boldsymbol{s}) := C \sum_{i \in \mathbb{N}_m} (1 - s_i)_+, \;\; \text{where } (t)_+ := \max\{t, 0\} \text{ for any } t \in \mathbb{R}. \tag{2.5}$$

It can be verified that both functions φ_l and ψ_h are convex but not differentiable.

A direct computation confirms that model (2.2) is equivalent to

$$\min\{\varphi_l(\boldsymbol{w}) + \psi_h(\mathsf{B}_c\boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^{m+1}\}. \tag{2.6}$$

Clearly, model (2.6) is in the form of (2.1).

2.2 ℓ^1-SVM Regression

The regression problem is concerned with a more general setting with the labels $y_i \in \mathbb{R}$, comparing to the classification problem which deals with the labels $y_i \in \{1, +1\}$. In this subsection, we describe the ℓ^1-SVM regression model.

Given instances $\boldsymbol{x}_i \in \mathbb{R}^n$ with labels $y_i \in \mathbb{R}$, $i \in \mathbb{N}_m$, the standard ℓ^1- SVM regression is to solve the minimization problem (2.2), having the same form as for the classification model, with a different fidelity function L, which depends on the parameter $\epsilon > 0$. This fidelity function L_ϵ, called the ϵ-insensitive loss function [30], has the form

$$L_\epsilon(\boldsymbol{\alpha}, \boldsymbol{x}_i, y_i, b) := \max\left\{\left|\sum_{j\in\mathbb{N}_m} \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b - y_i\right| - \epsilon, 0\right\},$$

where $K(\cdot,\cdot)$ is a given kernel function on $\mathbb{R}^n \times \mathbb{R}^n$.

In a way similar to the classification model, we could rewrite the ℓ^1-SVM regression model in the form of (2.1). Specifically, for the given labels $\boldsymbol{y} := (y_j : j \in \mathbb{N}_m)^T$ we define

$$\psi_{\epsilon,\boldsymbol{y}}(\boldsymbol{s}) := C \sum_{i\in\mathbb{N}_m} (|s_i - y_i| - \epsilon)_+, \quad \boldsymbol{s} \in \mathbb{R}^m, \tag{2.7}$$

where $C > 0$ is a given parameter. Likewise, $\psi_{\epsilon,\boldsymbol{y}}$ is convex and non-differentiable. For $\boldsymbol{\alpha}$ and b appearing in (2.2), we let $\boldsymbol{w} := \begin{pmatrix}\boldsymbol{\alpha}\\ b\end{pmatrix}$. Moreover, for $\mathsf{K} := [K(\boldsymbol{x}_i, \boldsymbol{x}_j) : i, j \in \mathbb{N}_m]$, we define $\mathsf{B}_r := [\mathsf{K}\ \mathbf{1}]$. In this notation, the ℓ^1-SVM regression model can be rewritten as

$$\min\{\varphi_l(\boldsymbol{w}) + \psi_{\epsilon,\boldsymbol{y}}(\mathsf{B}_r\boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^{m+1}\},$$

where φ_l is defined by (2.4). This is clearly in the form of (2.1).

2.3 Group LASSO Regularized-SVM Classification

In this subsection, we consider the group LASSO regularized-SVM classification problem. Selection of groups of a variable is an important problem in statistical learning. Such a selection process may be efficiently realized by using the group LASSO penalty that is the sum of the ℓ^2-norm of the groups of the variable [34]. It often gives superior performance over the standard ℓ^1-regularization when the underlying model has the prior information of group sparsity [9].

We now present the group LASSO regularized-SVM classification model. Suppose that the m-dimensional variable can be divided into l disjoint groups G_j, $j \in \mathbb{N}_l$. For $\boldsymbol{\alpha} \in \mathbb{R}^m$, we define a variable associated with a group G_i by $\boldsymbol{\alpha}_{G_i} := (\alpha_j : j \in G_i)$. The group LASSO regularized-SVM can be written as

$$\min \left\{\sum_{i\in\mathbb{N}_l} \delta_i \|\boldsymbol{\alpha}_{G_i}\|_2 + C \sum_{i\in\mathbb{N}_m} L(\boldsymbol{\alpha}, \boldsymbol{x}_i, y_i, b) : \boldsymbol{\alpha} \in \mathbb{R}^m,\ b \in \mathbb{R}\right\}, \tag{2.8}$$

where $\delta_i > 0$, $i \in \mathbb{N}_l$, are prescribed parameters and L is the hinge loss function defined by (2.3).

The group LASSO regularized-SVM classification model (2.8) can also be reformulated as a special case of model (2.1). To this end, we define ψ_h as in (2.5) and let

$$\varphi_g(\boldsymbol{s}) := \sum_{i\in\mathbb{N}_{l+1}} \delta_i \|\boldsymbol{s}_{G_i}\|_2, \quad \boldsymbol{s} \in \mathbb{R}^{m+1}, \tag{2.9}$$

where we set $G_{l+1} := \{m+1\}$ and $\delta_{l+1} := 0$. For $\boldsymbol{\alpha}, b$ appearing in (2.8), we let $\boldsymbol{w} := \begin{pmatrix}\boldsymbol{\alpha}\\ b\end{pmatrix}$. It is straightforward to verify that in this notation model (2.8) can be rewritten as

$$\min\{\varphi_g(\boldsymbol{w}) + \psi_h(\mathsf{B}_g\boldsymbol{w}) : \boldsymbol{w} \in \mathbb{R}^{m+1}\},$$

where $\mathsf{B}_g := \mathsf{Y}[\mathsf{K}\ \mathbf{1}]$ with $\mathsf{Y} := \operatorname{diag}(y_i : i \in \mathbb{N}_m)$ and $\mathsf{K} := [K(\boldsymbol{x}_i, \boldsymbol{x}_j) : i, j \in \mathbb{N}_m]$. Once again, this is in the form of (2.1).

2.4 Learning in Reproducing Kernel Banach Spaces

Reproducing kernel Banach spaces (RKBS) [16,18,27,28,32,35] were studied recently for sparse approximation models in machine learning. In particular, the RKBS with the ℓ^1 norm [27,28] was shown to produce sparse approximation of functions in supervised learning.

We next present the learning model in such RKBS and show that it could also be formulated in the general form (2.1). Suppose that $K(\cdot,\cdot)$ is a given kernel function and $\{(\boldsymbol{x}_i, y_i) : i \in \mathbb{N}_m\}$ are given training data. The learning model in the RKBS with the ℓ^1 norm is

$$\min\{\|\mathsf{K}\boldsymbol{c} - \boldsymbol{y}\|_2^2 + \mu\|\boldsymbol{c}\|_1 : \boldsymbol{c} \in \mathbb{R}^m\},$$

where $\mathsf{K} := [K(\boldsymbol{x}_i, \boldsymbol{x}_j) : i, j \in \mathbb{N}_m]$ is the kernel matrix, $\boldsymbol{y} := (y_i : i \in \mathbb{N}_m)^T$ and $\mu > 0$ is the prescribed regularization parameter. It is straightforward to observe that the above model may be reformulated as a special case of the general model (2.1) by identifying $\varphi(\boldsymbol{w}) := \|\mathsf{K}\boldsymbol{w} - \boldsymbol{y}\|_2^2$, $\psi(\boldsymbol{w}) := \|\boldsymbol{w}\|_1$ and $\mathsf{B} := \mathsf{I}$.

We now return to the general case. The difficulty of solving the general optimization model (2.1) is originated from the non-differentiability of the functions φ and ψ. A popular approach is to use differentiable functions to approximate the non-differentiable functions φ and ψ and to solve the approximate models by using existing optimization algorithms designed for smooth functions (see, for example, [33]). However, their performance depends on the choices of such smooth functions and the corresponding smoothing parameters. Hence, it requires extra efforts to determine the appropriate smooth functions and smoothing parameters. Moreover, solutions of the smoothing models may not perform as well as solutions of the original models, since the non-differentiability of the functions φ and ψ often characterizes the sparsity and other desirable features of the solutions of the corresponding models. We propose to solve the original optimization model (2.1) directly rather than its approximate models. Specifically, we shall characterize in the next section solutions of the model as fixed-points of certain nonlinear operators determined by the functions φ and ψ and develop two-step fast iteration algorithms to compute the solutions based on the resulting fixed-point equation.

3 A Two-Step Fixed-Point Proximity Iterative Algorithm

We develop fast numerical algorithms for solving the general non-differentiable optimization model (2.1). As we have pointed out in the last section, the challenge of solving this model lies in the non-differentiability of both functions involved in its objective function. To overcome this difficulty, we will first present the characterization of its solutions as fixed points of certain proximity operators. That is, we will convert the non-smooth minimization problem to an equivalent fixed point problem such that the extensive literature of fixed point algorithms could be employed to guide us to design convergent iterative schemes for finding its approximate solutions. In particular, the fixed-point methodology allows us to develop a two-step fast algorithm of computing such fixed points through a matrix splitting technique to improve convergence rate.

The solutions of (2.1) may be characterized as fixed points of certain proximity operators. We first review the definition of proximity operators. For $p \geq 1$, let $\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^m |x_i|^p\right)^{1/p}$

be the ℓ^p norm for $\boldsymbol{x} \in \mathbb{R}^m$. We denote by $\Gamma_0(\mathbb{R}^d)$ the class of all lower semi-continuous convex functions $f : \mathbb{R}^d \to (-\infty, +\infty]$ such that

$$\mathrm{dom}(f) := \{\boldsymbol{x} \in \mathbb{R}^d : f(\boldsymbol{x}) < +\infty\} \neq \emptyset.$$

The proximity operator of a function $f \in \Gamma_0(\mathbb{R}^n)$ is defined by

$$\mathrm{prox}_f(\boldsymbol{z}) := \mathrm{argmin}\left\{\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|_2^2 + f(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n\right\}, \quad \boldsymbol{z} \in \mathbb{R}^n.$$

It follows from a direct computation through using the relation between the sub-differential and the proximity operator [21] that a vector $\boldsymbol{w} \in \mathbb{R}^n$ is a minimizer of (2.1) if and only if there exists $\lambda, \beta > 0$ and $\boldsymbol{y} \in \mathbb{R}^m$ such that

$$\begin{aligned} \boldsymbol{w} &= \mathrm{prox}_{\frac{1}{\lambda}\varphi}\left(\boldsymbol{w} - \frac{C\beta}{\lambda}\mathsf{B}^T\boldsymbol{y}\right) \\ \boldsymbol{y} &= \left(\mathsf{I} - \mathrm{prox}_{\frac{1}{C\beta}\psi}\right)(\mathsf{B}\boldsymbol{w}+\boldsymbol{y}). \end{aligned} \tag{3.10}$$

We let $\boldsymbol{v} := \begin{pmatrix}\boldsymbol{w}\\ \boldsymbol{y}\end{pmatrix} \in \mathbb{R}^{n+m}$,

$$T(\boldsymbol{v}) := \begin{pmatrix} \mathrm{prox}_{\frac{1}{\lambda}\varphi}(\boldsymbol{w}) \\ \mathsf{I} - \mathrm{prox}_{\frac{1}{C\beta}\psi}(\boldsymbol{y}) \end{pmatrix}, \quad \text{and} \quad \mathsf{E} := \begin{bmatrix} \mathsf{I} & -\frac{C\beta}{\lambda}\mathsf{B}^T \\ \mathsf{B} & \mathsf{I} \end{bmatrix}.$$

We readily observe that the characterization (3.10) can be reformulated as

$$\boldsymbol{v} = T \circ \mathsf{E}(\boldsymbol{v}). \tag{3.11}$$

Namely, the solutions of problem (2.1) can be equivalently characterized by the fixed-points of the non-linear operator $T \circ \mathsf{E}$. That is, solving the minimization problem (2.1) is equivalent to solve the fixed-point problem (3.11). This formulation brings us two advantages. The first one is that the non-differentiability of the objective functions is no longer a problem if the proximity operators of the functions ϕ and ψ are easy to compute. In particular, as we will show later in Sect. 4, the proximity operators in all the three specific models we presented in Sect. 2 have closed forms. Moreover, the fixed point reformulation is more convenient to derive the convergent algorithms that are completely determined by properties of the non-linear operator $T \circ \mathsf{E}$.

We next consider the numerical algorithms of computing the fixed-point of (3.11). As pointed out in [14], the Picard iteration directly applied to (3.11) may not be convergent due to the expansiveness of E. Instead, we shall employ a matrix splitting technique to obtain a two-step fast convergent iteration scheme. In particular, we shall follow the general framework of multi-step proximity algorithms in [13] and introduce a specific and efficient splitting technique. We will justify its numerical performance through its implementation in many machine learning models later. Specifically, we shall split the expansive matrix E as follows:

$$\mathsf{E} = (\mathsf{E} - \mathsf{M}_0) + \mathsf{M}_1 + \mathsf{M}_2,$$

where

$$\mathsf{M}_0 := \begin{bmatrix} \mathsf{I} & h\frac{C\beta}{\lambda}\mathsf{B}^\top \\ l\mathsf{B} & \mathsf{I} \end{bmatrix}, \quad \mathsf{M}_1 := \begin{bmatrix} \mathsf{I} & h_1\frac{C\beta}{\lambda}\mathsf{B}^\top \\ l_1\mathsf{B} & \mathsf{I} \end{bmatrix}, \quad \mathsf{M}_2 := \begin{bmatrix} 0 & h_2\frac{C\beta}{\lambda}\mathsf{B}^\top \\ l_2\mathsf{B} & 0 \end{bmatrix}, \tag{3.12}$$

h, l, h_1, h_2, l_1, l_2 are parameters to be specified later. Since we require $M_0 = M_1 + M_2$, we see that $h = h_1 + h_2$ and $l = l_1 + l_2$. These matrices are chosen to ensure that the resulting iterative scheme not only converges but also has a faster convergence. We could then rewrite Eq. (3.11) as

$$\boldsymbol{v} = T \circ ((\mathsf{E} - \mathsf{M}_0)\boldsymbol{v} + \mathsf{M}_1\boldsymbol{v} + \mathsf{M}_2\boldsymbol{v}),$$

and consider the following two-step iteration scheme

$$\boldsymbol{v}^{k+1} = T \circ ((\mathsf{E} - \mathsf{M}_0)\boldsymbol{v}^{k+1} + \mathsf{M}_1\boldsymbol{v}^k + \mathsf{M}_2\boldsymbol{v}^{k-1}). \tag{3.13}$$

We point out that the above two-step iteration scheme (3.13) is an implicit scheme in general. However, we observe that if we choose M_1 and M_2 such that $\mathsf{E} - \mathsf{M}_0$ is strictly block upper (or lower) triangular, then it will lead to an explicit iterative scheme. In particular, with the specific choices of M_0, M_1, and M_2 in (3.12), the two-step iteration scheme (3.13) is equivalent to

$$\begin{aligned} \boldsymbol{w}^{k+1} &= \mathrm{prox}_{\frac{1}{\lambda}\varphi}\left(-(1+h)\frac{C\beta}{\lambda}\mathsf{B}^\top \boldsymbol{y}^{k+1} + \boldsymbol{w}^k + h_1\frac{C\beta}{\lambda}\mathsf{B}^\top \boldsymbol{y}^k + h_2\frac{C\beta}{\lambda}\mathsf{B}\boldsymbol{y}^{k-1}\right) \\ \boldsymbol{y}^{k+1} &= \left(\mathsf{I} - \mathrm{prox}_{\frac{1}{C\beta}\psi}\right)\left((1-l)\mathsf{B}\boldsymbol{w}^{k+1} + \boldsymbol{y}^k + l_1\mathsf{B}\boldsymbol{w}^k + l_2\mathsf{B}\boldsymbol{w}^{k-1}\right). \end{aligned} \tag{3.14}$$

If we choose $h := -1$ or $l := 1$, then the above iterative scheme would be an explicit iteration.

We also remark that both $\boldsymbol{w}^{k+1}$ and $\boldsymbol{y}^{k+1}$ use the previous two-step information, which speed up the overall convergence, as we shall see later in the numerical experiments. Iteration scheme (3.14) is different from the two-step scheme in [14], where $\boldsymbol{y}^{k+1}$ uses the previous two-step information and $\boldsymbol{w}^{k+1}$ uses only one-step information there. We will also show later that both $\boldsymbol{w}^{k+1}$ and $\boldsymbol{y}^{k+1}$ in the corresponding ADMM scheme use only one-step information of previous iterations. That is, the proposed two-step iteration scheme in this paper would be more efficient in computation. This will be confirmed by numerical experiments in Sect. 6.

The rest of this section is devoted to convergence analysis of the proposed two-step iteration scheme.

Theorem 3.1 *If $C, \lambda, \beta, h, l, h_1, h_2, l_1, l_2 \in \mathbb{R}$ satisfy*

(i) $h = h_1 + h_2$, $l = l_1 + l_2$,
(ii) $h + h_2 = l + l_2$,
(iii) $\sqrt{\frac{C\beta}{\lambda}}|h + h_2|\|B\|_2 < 1$,
(iv) $\frac{\max\{\frac{C\beta}{\lambda}, 1\}}{1 - \sqrt{\frac{C\beta}{\lambda}}|h+h_2|\|B\|_2} \max\{|h_2|, |l_2|\}\|B\|_2 < \frac{1}{2}$,

then the sequence $\{\boldsymbol{w}^k\}$ generated by the two-step iteration scheme (3.14) converges to a solution of problem (2.1) and has $O(\frac{1}{k})$ convergence rate in the ergodic sense.

Proof We employ Theorem 5.2 in [14] to derive the convergence result. Specifically, we need to show the matrices $(\mathsf{RM}_0, \mathsf{RM}_1, \mathsf{RM}_2)$ satisfy the Condition-M defined in Definition 4.2 in [14], where $\mathsf{R} := \begin{pmatrix} \frac{\lambda}{C\beta}\mathsf{I} & 0 \\ 0 & \mathsf{I} \end{pmatrix}$. That is, we need to show that

(a) $\mathsf{RM}_0 = \mathsf{RM}_1 + \mathsf{RM}_2$,
(b) $\mathsf{H} := \mathsf{RM}_0 + \mathsf{RM}_2$ is symmetric and positive definite,
(c) $\left\|\mathsf{H}^{-1/2}\mathsf{RM}_2\mathsf{H}^{-1/2}\right\|_2 < \frac{1}{2}$.

We shall check the above three conditions separately. The first condition (a) follows immediately from the definition of M_0, M_1, and M_2 in (3.12) and Assumption (i).

We next show the condition (b) also holds. It follows from a direct computation that

$$\mathsf{H} = \mathsf{R}(\mathsf{M}_0 + \mathsf{M}_2) = \begin{pmatrix} \frac{\lambda}{C\beta}\mathsf{I} & (h+h_2)\mathsf{B}^\top \\ (l+l_2)\mathsf{B} & \mathsf{I} \end{pmatrix}.$$

By Assumption (ii), we see H is symmetric. It follows from Assumption (iii) and Lemma 6.2 (i) in [13] that H is positive definite.

It remains to check the validity of condition (c). We shall prove it by a direct computation. It follows from Lemma 6.2 (ii) in [13] that

$$\|\mathsf{H}^{-1}\|_2 \le \frac{\max\{\frac{C\beta}{\lambda}, 1\}}{1 - \sqrt{\frac{C\beta}{\lambda}}|h+h_2|\|\mathsf{B}\|_2}.$$

Moreover, since

$$(\mathsf{RM}_2)^T(\mathsf{RM}_2) = \begin{pmatrix} l_2^2\mathsf{B}^T\mathsf{B} & 0 \\ 0 & h_2^2\mathsf{B}\mathsf{B}^T \end{pmatrix},$$

we have that

$$\|\mathsf{RM}_2\|_2 = \max\{|h_2|, |l_2|\}\|\mathsf{B}\|_2.$$

Condition (c) follows immediately from the above equality and equation combined with Assumption (iv). Therefore, the desired convergence results follows from Theorem 5.2 in [14]. □

4 Specific Two-Step Algorithms

In this section, we introduce two specific classes of two-step algorithms with different choices of the parameters h_1, h_2, l_1, l_2. We also provide explicit formulas of the proximity operators for the three machine learning models discussed in Sect. 2.

As we pointed out in Sect. 3, to obtain an explicit iteration from the general scheme (3.14), we need to either have $h = -1$ or $l = 1$. We next discuss these two cases separately.

We first consider the case that $h = -1$. It follows from the conditions in Theorem 3.1 that

$$\begin{cases} h_1 = -2 - l_1 - 2l_2 \\ h_2 = 1 + l_1 + 2l_2 \\ \quad l = l_1 + l_2. \end{cases}$$

Substituting these relations into the general scheme (3.14), we obtain that

$$\begin{aligned} \boldsymbol{w}^{k+1} &= \operatorname{prox}_{\frac{1}{\lambda}\varphi}\left(\boldsymbol{w}^k - \frac{C\beta}{\lambda}\mathsf{B}^T\left(\boldsymbol{y}^k + (1 + l_1 + 2l_2)(\boldsymbol{y}^k - \boldsymbol{y}^{k-1})\right)\right) \\ \boldsymbol{y}^{k+1} &= \left(\mathsf{I} - \operatorname{prox}_{\frac{1}{C\beta}\psi}\right)\left(\boldsymbol{y}^k + \mathsf{B}\left(\boldsymbol{w}^{k+1} - l_1(\boldsymbol{w}^{k+1} - \boldsymbol{w}^k) - l_2(\boldsymbol{w}^{k+1} - \boldsymbol{w}^{k-1})\right)\right). \end{aligned} \tag{4.15}$$

It is direct to observe that the vectors $\boldsymbol{w}^{k+1}$ and $\boldsymbol{y}^{k+1}$ can be explicitly solved from (4.15).

Likewise, for the case $l = 1$, we need the following conditions on parameters h_1, h_2, l_1, l_2:

$$\begin{cases} l_1 = 2 - h_1 - 2h_2 \\ l_2 = -1 + h_1 + 2h_2 \\ h = h_1 + h_2. \end{cases}$$

The corresponding explicit algorithm is

$$\begin{aligned} \boldsymbol{y}^{k+1} &= \left(\mathsf{I} - \operatorname{prox}_{\frac{1}{C\beta}\psi}\right)\left(\boldsymbol{y}^k + \mathsf{B}\left(\boldsymbol{w}^k + (1 - h_1 - 2h_2)(\boldsymbol{w}^k - \boldsymbol{w}^{k-1})\right)\right) \\ \boldsymbol{w}^{k+1} &= \operatorname{prox}_{\frac{1}{\lambda}\varphi}\left(\boldsymbol{w}^k - \frac{C\beta}{\lambda}\mathsf{B}^\top\left(\boldsymbol{y}^{k+1} + h_1(\boldsymbol{y}^{k+1} - \boldsymbol{y}^k) + h_2(\boldsymbol{y}^{k+1} - \boldsymbol{y}^{k-1})\right)\right). \end{aligned} \tag{4.16}$$

These two classes of algorithms are both two-step iterative algorithms, which make each iteration considerably efficient. Moreover, the parameters l_1, l_2 or h_1, h_2 may be used to control the amount of the two-step information to be employed.

The proposed two-step iterative schemes depend on the proximity operators of the functions φ and ψ. Efficiency of these schemes pretty much depends on how these operators are computed. We next provide closed form formulas for the proximity operators of φ and ψ in all the three models introduced in Sect. 2. They lead to fast solvers of the machine learning problems under consideration.

We first compute the proximity operators for the ℓ^1-SVM classification model in Sect. 2.1.

Proposition 4.1 *If $\varphi(\boldsymbol{w}) := \|A\boldsymbol{w}\|_1$, where A is an $n \times n$ diagonal matrix having the diagonal entries $a_j \geq 0$, then for any $z \in \mathbb{R}^n$ and $\lambda > 0$,*

$$\left(\operatorname{prox}_{\frac{1}{\lambda}\varphi}(z)\right)_j = \max\left\{|z_j| - \frac{a_j}{\lambda}, 0\right\}\operatorname{sgn}(z_j), \quad j \in \mathbb{N}_n. \tag{4.17}$$

Proof Since $\varphi(z) = \|\mathsf{A}z\|_1$ and A is a diagonal matrix with all entries $a_j \geq 0$, we have that

$$\frac{1}{\lambda}\varphi(z) = \sum_{j=1}^{n} \frac{a_j}{\lambda}|z_j|.$$

We modify a result in [22] to prove this proposition. It has been shown in [22] that for any $z \in \mathbb{R}^n$ and $\lambda > 0$, the proximity operator of $\frac{1}{\lambda}\|z\|_1$ is

$$\operatorname{prox}_{\frac{1}{\lambda}\|\cdot\|_1}(z) = \left(\operatorname{prox}_{\frac{1}{\lambda}|z_1|}, \ldots, \operatorname{prox}_{\frac{1}{\lambda}|z_n|}\right)^T,$$

where

$$\operatorname{prox}_{\frac{1}{\lambda}|\cdot|}(z_j) = \max\left\{|z_j| - \frac{1}{\lambda}, 0\right\}\operatorname{sgn}(z_j).$$

The proximity operator of $\frac{1}{\lambda}\varphi$ in (4.17) follows from replacing $\frac{1}{\lambda}$ by $\frac{a_j}{\lambda}$. □

The function φ_l defined by (2.4) for the ℓ^1-SVM classification model in Sect. 2.1 is a special case of the function φ discussed in the last proposition. Below, we consider computing the proximity operator of ψ defined by (2.5).

Proposition 4.2 *If ψ is defined by (2.5), then for any $z \in \mathbb{R}^n$ and $\beta > 0$,*

$$\left(\operatorname{prox}_{\frac{1}{C\beta}\psi}(z)\right)_j = \begin{cases} z_j, & \text{if } z_j \geq 1, \\ 1, & \text{if } 1 - \frac{1}{\beta} \leq z_j < 1, \\ z_j + \frac{1}{\beta}, & \text{if } z_j < 1 - \frac{1}{\beta}, \end{cases} \quad j \in \mathbb{N}_n.$$

Proof The proximity operator of $\frac{1}{C\beta}\psi$ can be computed component-wise. For each $j \in \mathbb{N}_n$, we have that

$$\left(\operatorname{prox}_{\frac{1}{C\beta}\psi}(z)\right)_j = \operatorname{argmin}\left\{\frac{1}{2}(x_j - z_j)^2 + \frac{1}{\beta}(1 - x_j)_+ : x_j \in \mathbb{R}\right\}.$$

We compute the above minimizer in three cases $z_j \geq 1$, $1 - \frac{1}{\beta} \leq z_j < 1$, and $z_j < 1 - \frac{1}{\beta}$ separately. Suppose that $z_j \geq 1$. When $x_j \leq 1$, we have that the minimal value is $\frac{1}{2}(1 - z_j)^2$ taking at $x_j = 1$. When $x_j > 1$, the minimal value is 0 taking at $x_j = z_j$. It is clear that when $z_j \geq 1$, the minimizer is z_j. The other two cases follow in a similar proof. □

We next compute the proximity operators for the ℓ^1-SVM regression model in Section 2.2. Note that φ_l is the same as that in the ℓ^1-SVM classification model and its proximity operator is given in (4.17). The proximity operator of $\psi_{\epsilon,y}$ in (2.7) is given below:

(i) If $\epsilon \geq \frac{C}{2\beta}$, then for $j \in \mathbb{N}_n$

$$\left(\operatorname{prox}_{\frac{1}{\beta}\psi_{\epsilon,y}}(z)\right)_j = \begin{cases} z_j - \frac{C}{\beta}, & \text{if } z_j \geq \epsilon + \frac{C}{\beta} + y_i \\ \epsilon + y_i, & \text{if } \epsilon + y_i \leq z_j < \epsilon + \frac{C}{\beta} + y_i \\ z_j, & \text{if } \epsilon - \frac{C}{\beta} + y_i \leq z_j < \epsilon + y_i \\ z_j + \frac{C}{\beta}, & \text{if } -\epsilon + y_i \leq z_j < \epsilon - \frac{C}{\beta} + y_i \\ -\epsilon + y_i, & \text{if } -\epsilon - \frac{C}{\beta} + y_i \leq z_j < -\epsilon + y_i \\ z_j + \frac{C}{\beta}, & \text{if } z_j < -\epsilon - \frac{C}{\beta} + y_i. \end{cases}$$

(ii) If $\epsilon < \frac{C}{2\beta}$, then for $j \in \mathbb{N}_n$

$$\left(\operatorname{prox}_{\frac{1}{\beta}\psi_{\epsilon,y}}(z)\right)_j = \begin{cases} z_j - \frac{C}{\beta}, & \text{if } z_j \geq \epsilon + \frac{C}{\beta} + y_i \\ \epsilon + y_i, & \text{if } \epsilon + y_i \leq z_j < \epsilon + \frac{C}{\beta} + y_i \\ z_j, & \text{if } -\epsilon + y_i \leq z_j < \epsilon + y_i \\ -\epsilon + y_i, & \text{if } -\epsilon - \frac{C}{\beta} + y_i \leq z_j < -\epsilon + y_i \\ z_j + \frac{C}{\beta}, & \text{if } z_j < -\epsilon - \frac{C}{\beta} + y_i. \end{cases}$$

For the group LASSO regularized-SVM classification model in Section 2.3, ψ_h is the same as that in the ℓ^1-SVM classification model and the proximity operator of φ_g is:

$$\left(\operatorname{prox}_{\frac{1}{\lambda}\varphi_g}(z)\right)_{G_j} = \max\left\{\|z_{G_j}\|_2 - \frac{\delta_j}{\lambda}, 0\right\} \frac{z_{G_j}}{\|z_{G_j}\|_2}.$$

With the formulas developed above for the proximity operators of functions that appear in the three machine learning models studied in this paper, algorithms (4.15) and (4.16) lead to fast solvers for these models.

5 Comparison with ADMM

We identify in this section the ADMM method as a special case of the proposed two-step iteration scheme to understand the insight on what makes an iterative scheme converges fast.

ADMM is a powerful method in solving convex optimization problems [4,25]. It can also be applied to solve the general optimization model (2.1). In this section, we compare the proposed two-step iteration scheme with the ADMM scheme. Specifically, we identify ADMM as a special case of the proposed scheme which allows us to see that in updating $\boldsymbol{y}^{k+1}$ and $\boldsymbol{w}^{k+1}$, ADMM only uses one-step information, while the proposed scheme in general uses two-step information.

We first recall the ADMM scheme for solving the optimization model (2.1). The ADMM scheme for solving (2.1) follows from the augmented Lagrangian and a linearized technique [25]. For some given positive constants μ, γ, and some initial points $\boldsymbol{w}^0$, $\boldsymbol{z}^0$, $\boldsymbol{u}^0$, the standard ADMM iteration scheme is given by

$$\begin{aligned}
\boldsymbol{z}^{k+1} &= \operatorname{prox}_{\gamma\psi}\left(\mathsf{B}\boldsymbol{w}^k + \boldsymbol{u}^k\right)\\
\boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + \mathsf{B}\boldsymbol{w}^k - \boldsymbol{z}^{k+1}\\
\boldsymbol{w}^{k+1} &= \operatorname{prox}_{\mu\varphi}\left(\boldsymbol{w}^k - \frac{\mu}{\gamma}\mathsf{B}^T\left(\mathsf{B}\boldsymbol{w}^k - \boldsymbol{z}^{k+1} + \boldsymbol{u}^{k+1}\right)\right).
\end{aligned}$$

For convenience of comparing ADMM with the proposed two-step iteration scheme, we identify ADMM as a special case of (3.14). Let $\boldsymbol{y}^k = \boldsymbol{u}^k$, $\gamma = \frac{1}{C\beta}$ and $\mu = \frac{1}{\lambda}$. It follows from a direct computation that the above linearized ADMM scheme is equivalent to

$$\begin{aligned}
\boldsymbol{y}^{k+1} &= \left(\mathsf{I} - \operatorname{prox}_{\frac{1}{C\beta}\psi}\right)\left(\mathsf{B}\boldsymbol{w}^k + \boldsymbol{y}^k\right)\\
\boldsymbol{w}^{k+1} &= \operatorname{prox}_{\frac{1}{\lambda}\varphi}\left(\boldsymbol{w}^k - \frac{C\beta}{\lambda}\mathsf{B}\left(2\boldsymbol{y}^{k+1} - \boldsymbol{y}^k\right)\right).
\end{aligned} \tag{5.18}$$

Clearly, we observe that the above iteration scheme is equivalent to (3.14) with the following choice of parameters:

(i) $l = 1, l_1 = 1, l_2 = 0$;
(ii) $h = 1, h_1 = 1, h_2 = 0$.

Moreover, since both l_2 and h_2 are taking 0 in the ADMM scheme, only one-step information is used in updating $\boldsymbol{y}^{k+1}$ and $\boldsymbol{w}^{k+1}$. On the contrary, the proposed two-step iterative scheme (3.14) in general not only allows more flexible choices of such parameters, but also improves the convergence through using two-step information and choosing appropriate parameters. The advantages of the proposed two-step iterative scheme (3.14) over the ADMM scheme will be confirmed by numerical results on some benchmark datasets in Sect. 6.

6 Numerical Experiments

In this section, we present numerical results to confirm the efficiency of the proposed two-step iteration scheme. Specifically, we implement it to solve all the three non-differentiable models mentioned in Sect. 2: the ℓ^1-SVM classification model, the ℓ^1- SVM regression model, and the group LASSO regularized SVM classification model, on some benchmark datasets. We compare it with the linear programming method (LPM) and ADMM. For all models tested in this section, we use the Gaussian kernel due to its universality [23,31]. The comparison shows the proposed iteration scheme outperforms significantly LPM and ADMM in solving these three machine learning models in terms of computational time.

The numerical experiments presented in this section except the extremely large dataset are implemented in a personal computer with an 2.6 GHz Intel Core i5 CPU and 8G RAM memory. We use a server with an 3.33 GHz Intel Xeon Core 2 CPU and a 96G RAM memory to work with the extremely large dataset.

6.1 ℓ^1-SVM Classification

We first implement numerical experiments for solving the ℓ^1-SVM classification. As we mentioned in Sect. 2.1, the ℓ^1-norm SVM would generate less support vectors than the standard ℓ^2-norm SVM. The ℓ^1-SVM is advantageous when there are redundant noise features [36] and usually has shorter training time than the standard ℓ^2-SVM [10]. We compare these two models on a synthetic dataset and a real-world dataset below.

We first generate a synthetic dataset in the following way. We generate a random vector $\boldsymbol{x} \in \mathbb{R}^{30}$ while each component is uniformly selected from $[-6, 6]$. We then assign its label: if $x_1 \geq x_2 + \cdots + x_{30} + 1$, then we set its label to be $\{1\}$; if $x_1 \leq x_2 + \cdots + x_{30} - 1$, we set its label to be $\{-1\}$. We repeat the process until obtaining 1905 instances with labels. We randomly choose 948 instances as training data and the other 957 instances as test data.

We also use a real-world dataset "Australian" from the benchmark datasets [5], it has 699 instances and each instance has 14 features. We use 400 instances as training data and the other 299 as testing data.

While algorithms of solving the ℓ^2-SVM have been well studied in the literature [3], the ℓ^1-SVM solvers are not fully developed. We point out that the linear programming method [26,30,36] (LPM) is a popular method of solving the ℓ^1-SVM classification model. However, LPM has relatively higher demands both on the memory storage and computational time for large size problems. ADMM is another popular method for solving the general optimization model (2.1). We compare the proposed two-step iteration method with LPM and ADMM on solving the ℓ^1-SVM classification model. In particular, we compare all of these three methods in terms of computational cost, classification accuracy, and storage cost with different sizes of several benchmark datasets.

Following [26], LPM applied to solving the ℓ^1-SVM model (2.2) has the form:

$$
\begin{aligned}
\min_{\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \xi \in \mathbb{R}^m, b \in \mathbb{R}} \quad & \left\{ \sum_{i=1}^{m} \left(\boldsymbol{\alpha}_i^+ + \boldsymbol{\alpha}_i^-\right) + C \sum_{i=1}^{m} \boldsymbol{\xi}_i \right\} \\
\text{subject to} \quad & y_i \left(\sum_{j=1}^{m} \left(\boldsymbol{\alpha}_j^+ - \boldsymbol{\alpha}_j^-\right) K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b \right) \geq 1 - \boldsymbol{\xi}_i, \\
& \boldsymbol{\alpha}_i^+, \boldsymbol{\alpha}_i^-, \boldsymbol{\xi}_i \geq 0, \quad i = 1, 2, \ldots, m.
\end{aligned}
$$

The standard package **linprog** in the optimization toolbox of the MATLAB could be employed to solve the above linear programming problem.

We now compare the computational time and classification accuracy of the proposed two-step fixed-point proximity algorithm (TFP^{2}A), ADMM, and LPM. We consider four benchmark datasets [5] of different sizes. The first dataset is "Australian" with 699 instances and each instance has 14 features. We use 400 instances as training data and the other 299 as testing data. The second dataset is "Breast Cancer" with 683 instances and each instance has 10 features. We set 500 instances as training data and 183 as testing data. The third one is "diabetes" with 768 instances and each instance has 8 features. We set 500 instances as training data and 268 as testing data. The last one is "Splice" with 3175 instances and 60 features for each instance. We use 1000 instances as training data and 2175 as testing data. We use the same Gaussian kernel $K(\boldsymbol{s}, \boldsymbol{t}) = \mathrm{e}^{-0.01\|\boldsymbol{s}-\boldsymbol{t}\|_2^2}$ and the same regularization parameter $C = 3$ for all these three algorithms. We remark that the regularization parameter plays an important role in the prediction accuracy. Since all three methods are solving the same model, we thus focus on the efficiency comparison and use the same regularization parameter for all

Table 1 Comparison of LPM, ADMM, and the proposed two-step method for solving ℓ^1-SVM classification in training time (in seconds), the prediction accuracy, and the minimal objective function value

	Australian			Breast cancer		
	Time (s)	Accuracy (%)	Obj. fun.	Time (s)	Accuracy (%)	Obj. fun.
LPSVM	6.34	84.83	118.28	12.15	99.45	35.91
ADMM	1.75	85.86	120.67	0.72	99.45	36.88
TFP^{2}A	0.54	86.90	119.25	0.11	99.45	36.15
	Diabetes			**Splice**		
	Time (s)	Accuracy (%)	Obj. fun.	Time (s)	Accuracy (%)	Obj. fun.
LPSVM	11.24	80.97	1.203E4	61.07	88.69	730.46
ADMM	10.56	82.09	1.226E4	14.65	86.80	738.85
TFP^{2}A	1.06	82.09	1.218E4	6.52	86.94	736.50

Table 2 Numbers of iterations needed for different stopping tolerance levels

tolerance	Australian		Breast cancer		Diabetes		Splice	
	ADMM	TFP^{2}A	ADMM	TFP^{2}A	ADMM	TFP^{2}A	ADMM	TFP^{2}A
10^{-1}	5	4	4	4	449	4	5	4
10^{-2}	11	7	7	10	1990	6	12	7
10^{-3}	17	9	30	21	1992	27	33	17
10^{-4}	63	49	45	31	1992	67	99	45
10^{-5}	293	136	79	127	1993	161	256	120
10^{-6}	979	426	152	184	3615	355	577	256
10^{-7}	1784	620	419	287	3912	615	1410	606
10^{-8}	2261	759	3929	560	7586	2800	3931	1699
10^{-9}	3665	2337	10, 000	1996	10, 000	7965	10, 000	7733

of them. For the proposed two-step method and ADMM, the stopping criterion is set to be the relative error between the successive iterations less than a given tolerance 10^{-7}. We present the comparison of computational time for training and classification accuracy in Table 1.

We remark that all the three algorithms have similar prediction accuracy since they are solving the same model. However, the proposed two-step method has a much less training time in all the four datasets. Moreover, we present a more detailed comparison between TFP^{2}A and ADMM. We observe from the iteration scheme (5.18) that both of them have almost the same computational cost in each iteration. Therefore, we compare the number of iterations needed to obtain tolerance levels ranging from 10^{-1} to 10^{-9} on these four datasets described above in Table 2. We further visualize the comparison in Fig. 1, from which we observe that under the same stopping criteria the proposed two-step method converges much faster than ADMM.

We next compare the memory performance of these three algorithms. We use a large scale data set “ala” from the UCI machine learning repository [17]. We consider the training data size N ranging from 800, 1000, 1100, to 3000. The results are reported in Table 3.

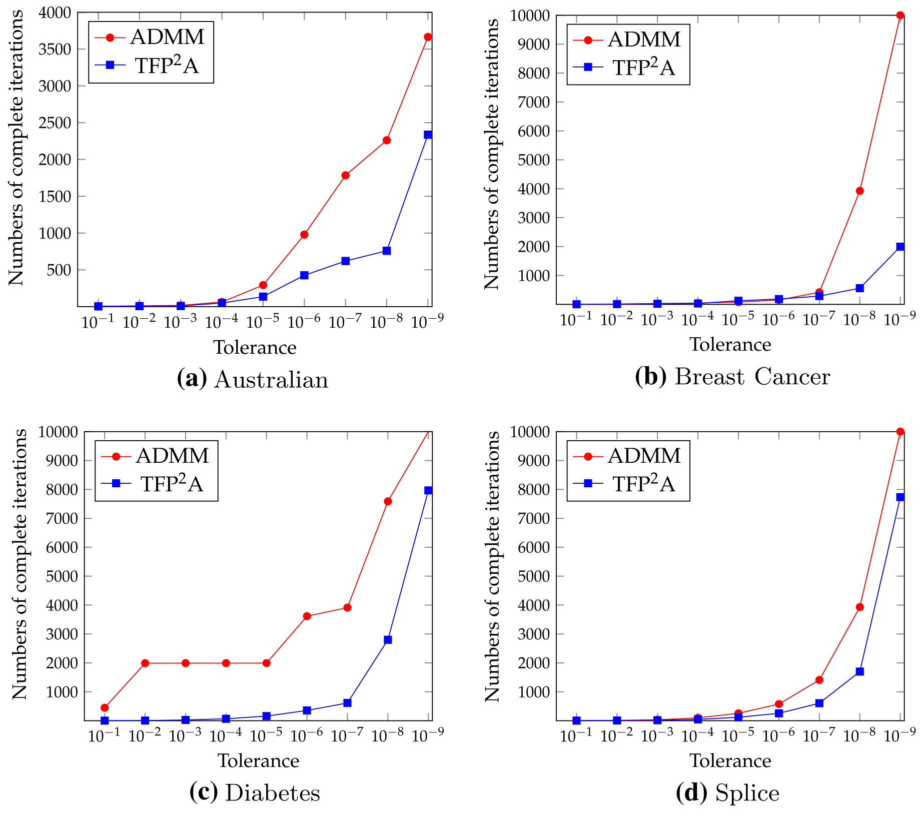

Fig. 1 The number of iterations used by ADMM and TFP^{2}A under different tolerance levels

Table 3 Memory performance of the proposed two-step method, ADMM, and LPSVM. “No” means out of memory

Data size	800	1000	1100	3000
LPM	Yes	Yes	No	No
ADMM	Yes	Yes	Yes	Yes
TFP^{2}A	Yes	Yes	Yes	Yes

We observe from the above Table 3 that the proposed two-step method and ADMM require less memory storage in the training process. In other words, these two algorithms would be able to handle larger datasets than LPM.

We further compare the proposed two-step method and ADMM with two larger datasets. The first dataset “covtype” is from [5] and contains 10,000 instances as training data and 5000 instances as testing data. The second dataset is handwriting digits which is from the MNIST database [11]. The original MNIST database of handwriting digits consists of 60,000 training examples and 10,000 testing instances of “0” through “9”. We only use 2 digits of this dataset (“0” and “1”) to make classification, which results in a dataset containing 12665 training instances and 2115 testing instances of “0” and “1”. We point out that it is beyond the memory storage limit of our personal computer. We use a server with an 3.33 GHz Intel Xeon Core 2 CPU and a 96G RAM memory to implement this experiment. The numerical results are reported in Table 4.

Table 4 Comparison of ADMM and the proposed two-step method in training time and the prediction accuracy

	Covtype			Handwriting		
	Time (s)	Accuracy (%)	Iteration	Time (s)	Accuracy (%)	Iteration
ADMM	680.28	81.04	452	1700.50	99.72	656
TFP^{2}A	88.18	81.04	60	896.39	99.72	270

Table 5 Comparison of ADMM and TFP^{2}A in training time, MSE, and iteration numbers

	Housing			Mg			Mg		
	Time (s)	MSE	Iteration	Time (s)	MSE	Iteration	Time (s)	MSE	Iteration
ADMM	0.71	50.98	1293	7.61	0.04	716	17.56	13.57	456
TFP^{2}A	0.29	50.91	570	2.42	0.04	238	6.77	13.39	184

We observe from the above table that the proposed method consumes much less computing time and requires much fewer iteration numbers than ADMM, while maintaining the same prediction accuracy.

6.2 SVM Regression with ℓ^1-norm Regularization

We next compare the proposed TFP^{2}A with ADMM for solving the ℓ^1-SVM regression problem.

In this computation, we consider three benchmark datasets [5] of different sizes. The first dataset is "Housing" with 506 instances and each instance has 13 features. We use 300 instances as training data and the other 206 as testing data. The second dataset is "Mg" with 1385 instances and each instance has 6 features. We set 1000 instances as training data and 385 as testing data. The third one is "Abalone" with 4177 instances and each instance has 8 features. We set 2000 instances as training data and 2177 as testing data.

For both the proposed two-step method and ADMM, we use the same Gaussian kernel, the same regularized parameter C, and the same stopping criterion with the relative error between the successive iterations less than 10^{-7}. We compare the two competing methods in terms of the computational time, the mean squared error (MSE), and the iteration numbers. The numerical results are reported in Table 5.

We remark that both algorithms have similar MSE since they are solving the same model. However, the proposed two-step method consumes much less training time and requires less iterations for all the three datasets.

6.3 SVM Classification with Group LASSO Regularization

In this subsection, we present numerical comparison of the proposed two-step method and ADMM for solving SVM classification with group LASSO regularization.

We use the same datasets "Australian"', "Breast Cancer", "Diabetes", and "Splice" as in Sect. 6.1. We divide the variables into 10 groups for all these datasets. We use the same Gaussian kernel and the same regularized parameter C for both algorithms. The stopping criterion is also set to be the relative error between the successive iterations less than 10^{-7}.

Table 6 Comparison of ADMM and the proposed two-step method for solving group LASSO regularized SVM in training time (in seconds) and the prediction accuracy

	Australian		Breast cancer		Diabetes		Splice	
	Time (s)	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)	Accuracy (%)	Time (s)	Accuracy (%)
ADMM	1.86	86.21	1.66	99.45	4.81	81.72	16.65	86.62
TFP^{2}A	0.61	86.21	0.51	99.45	1.40	81.72	8.72	89.79

We compare the proposed two-step method and ADMM in computing time for training and learning accuracy. The numerical results are reported in Table 6.

We observe from Table 6 that the proposed two-step method consumes much less training time than ADMM for all the four datasets.

7 Conclusion

We have developed a two-step fixed-point proximity algorithm for solving a class of non-differentiable optimization models motivated from sparse learning. We prove the convergence results of the proposed method. Moreover, we identify the well-known ADMM as a special case of the proposed method and show that ADMM uses only one-step information and the proposed method uses two-step information in each iteration. Numerical experiments with some benchmark datasets show that the proposed algorithm has advantages in both convergence speed and memory storage compared to either LPM or ADMM. The proposed method outperforms LPM or ADMM significantly in terms of computational time.

References

1. Argyriou, A., Micchelli, C.A., Pontil, M., Shen, L., Xu, Y.: Efficient first order methods for linear composite regularizers. arXiv:1104.1436 (2011)
2. Bach, F., Jenatton, R., Mairal, J., Obozinski, G., et al.: Optimization with sparsity-inducing penalties. Found. Trends® Mach. Learn. **4**, 1–106 (2012)
3. Bottou, L., Lin, C.-J.: Support vector machine solvers. Large Scale Kernel Mach. 301–320 (2007)
4. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends® Mach. Learn. **3**, 1–122 (2011)
5. Chang, C., Lin, C.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**, 27:1–27:27 (2011)
6. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**, 273–297 (1995)
7. Fung, G.M., Mangasarian, O.L.: A feature selection Newton method for support vector machine classification. Comput. Optim. Appl. **28**, 185–202 (2004)
8. Hastie, T., Tibshirani, R., Wainwright, M.: Statistical Learning with Sparsity: The Lasso and Generalizations. CRC Press, Boca Raton (2015)
9. Jacob, L., Obozinski, G., Vert, J.P.: Group lasso with overlap and graph lasso. In: International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, pp. 433–440 (2009)
10. Koshiba, Y., Abe, S.: Comparison of L1 and L2 support vector machines. In: Proceedings of the International Joint Conference On Neural Networks, 2003, vol. 3. IEEE, pp. 2054–2059 (2003)
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**, 2278–2324 (1998)
12. Li, Q., Micchelli, C.A., Shen, L., Xu, Y.: A proximity algorithm accelerated by Gauss–Seidel iterations for L1/TV denoising models. Inverse Probl. **28**, 095003 (2012)
13. Li, Q., Shen, L., Xu, Y., Zhang, N.: Multi-step fixed-point proximity algorithms for solving a class of optimization problems arising from image processing. Adv. Comput. Math. **41**, 387–422 (2015)

14. Li, Q., Xu, Y., Zhang, N.: Two-step fixed-point proximity algorithms for multi-block separable convex problems. J. Sci. Comput. **70**, 1204–1228 (2017)
15. Li, Z., Song, G., Xu, Y.: A fixed-point proximity approach to solving the support vector regression with the group Lasso regularization. Int. J. Numer. Anal. Model. **15**, 154–169 (2018)
16. Li, Z., Xu, Y., Ye, Q.: Sparse support vector machines in reproducing kernel Banach spaces. In: Dick, J., Kuo, F., Woźniakowski, H. (eds.) Contemporary Computational Mathematics: A Celebration of the 80th Birthday of Ian Sloan, pp. 869–887. Springer, Cham (2018)
17. Lichman, M.: UCI Machine Learning Repository. University of California, Irvine (2013)
18. Lin, R., Song, G., Zhang, H.: Multi-task learning in vector-valued reproducing kernel Banach spaces with the ℓ^1 norm. arXiv:1901.01036 [math] (2019)
19. Mangasarian, O.L.: Exact 1-norm support vector machines via unconstrained convex differentiable minimization. J. Mach. Learn. Res. **7**, 1517–1530 (2007)
20. Meier, L., Van De Geer, S., Bühlmann, P.: The group lasso for logistic regression. J. R. Stat. Soc. Ser. B (Stat. Methodol.) **70**, 53–71 (2008)
21. Micchelli, C.A., Shen, L., Xu, Y.: Proximity algorithms for image models: denoising. Inverse Prob. **27**, 045009 (2011)
22. Micchelli, C.A., Shen, L., Xu, Y., Zeng, X.: Proximity algorithms for the L1/TV image denoising model. Adv. Comput. Math. **38**, 401–426 (2013)
23. Micchelli, C.A., Xu, Y., Zhang, H.: Universal kernels. J. Mach. Learn. Res. **7**, 2651–2667 (2006)
24. Nesterov, Y.E.: A method for solving the convex programming problem with convergence rate O $(1/k\hat{2})$, in Dokl. Akad. Nauk Sssr **269**, 543–547 (1983)
25. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends Optim. **1**, 127–239 (2014)
26. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2002)
27. Song, G., Zhang, H.: Reproducing kernel Banach spaces with the l1 norm II: error analysis for regularized least square regression. Neural Comput. **23**, 2713–2729 (2011)
28. Song, G., Zhang, H., Hickernell, F.J.: Reproducing kernel Banach spaces with the l1 norm. Appl. Comput. Harmon. Anal. **34**, 96–116 (2013)
29. Tibshirani, R.: Regression shrinkage and selection via the lasso. J. R. Stat. Soc. Ser. B Methodol. **58**, 267–288 (1996)
30. Vapnik, V.: Statistical Learning Theory. Wiley, New York (1998)
31. Wang, R., Xu, Y.: Functional reproducing kernel Hilbert spaces for non-point-evaluation functional data. Appl. Comput. Harmon. Anal. **46**, 569–623 (2019)
32. Xu, Y., Ye, Q.: Generalized Mercer kernels and reproducing kernel Banach spaces. Mem. Am. Math. Soc. **258**, 1–122 (2019)
33. Yuan, G., Chang, K., Hsieh, C., Lin, C.: A comparison of optimization methods and software for large-scale L1-regularized linear classification. J. Mach. Learn. Res. **11**, 3183–3234 (2010)
34. Yuan, M., Lin, Y.: Model selection and estimation in regression with grouped variables. J. R. Stat. Soc. **68**, 49–67 (2006)
35. Zhang, H., Xu, Y., Zhang, J.: Reproducing kernel Banach spaces for machine learning. J. Mach. Learn. Res. **10**, 3520–3527 (2009)
36. Zhu, J., Rosset, S., Hastie, T., Tibshirani, R.: 1-norm support vector machines. Adv. Neural Inf. Process. Syst. **16**, 49–56 (2004)