Saddle Points and Accelerated Perceptron Algorithms

Adams Wei Yu[†] WEIYU@CS.CMU.EDU
Fatma Kılınç-Karzan[‡] FKILINC@ANDREW.CMU.EDU
Jaime G. Carbonell[†] JGC@CS.CMU.EDU
[†]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
[‡]Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

In this paper, we consider the problem of finding a linear (binary) classifier or providing a near-infeasibility certificate if there is none. We bring a new perspective to addressing these two problems simultaneously in a single efficient process, by investigating a related Bilinear Saddle Point Problem (BSPP). More specifically, we show that a BSPP-based approach provides either a linear classifier or an ϵ-infeasibility certificate. We show that the accelerated primal-dual algorithm, Mirror Prox, can be used for this purpose and achieves the best known convergence rate of $O(\frac{\sqrt{\log n}}{\rho(A)})(O(\frac{\sqrt{\log n}}{\epsilon}))$, which is *almost independent of the problem size*, n. Our framework also solves kernelized and conic versions of the problem, with the same rate of convergence. We support our theoretical findings with an empirical study on synthetic and real data, highlighting the efficiency and numerical stability of our algorithm, especially on large-scale instances.

1. Introduction

One of the central tasks in supervised learning is to train a binary classifier: Given a training data set $A \in \mathbb{R}^{m\times n}$ of size n, where each column $A_j \in \mathbb{R}^m$ is an instance with label either 1 or -1; find a binary (linear) classifier that separates data into two groups based on their labels. Without loss of generality, we assume that A does not contain a zero column. After reversing the sign of each negative instance (Blum & Dunagan, 2002) (for simplicity, we still denote the data matrix A), training a classifier is equivalent to finding a feasible solution to the following system of homogeneous linear inequalities w.r.t. $y \in \mathbb{R}^m$:

$$A^T y > 0. \tag{1}$$

We refer (1) as *Linear Dual Feasibility Problem* (LDFP), and its solutions as *linear classifiers* or *feasibility (separability) certificates*. When the training data is not linearly separable, an *infeasibility (inseparability) certificate* is given by a solution to the *Linear Alternative Problem* (LAP) of (1):

$$Ax = 0, \mathbf{1}^T x = 1, x \geq 0, \tag{2}$$

where $\mathbf{1} \in \mathbb{R}^n$ denotes the vector with all coordinates equal to 1. LDFP and LAP are *Linear Feasibility Problems* (LFP) that are dual to each other: (1) is feasible if and only if (2) is infeasible (by Gordon's Theorem (Chvatal, 1983)). In particular, a feasible solution of one problem is an infeasibility certificate of the other. Often, instead of seeking an exact infeasibility certificate, it is more practical to opt for an ϵ-*infeasibility certificate* for LDFP, i.e., finding an ϵ-*solution* for LAP, x^ϵ, such that $\|Ax^\epsilon\|_2 \leq \epsilon, \mathbf{1}^T x^\epsilon = 1, x^\epsilon \geq 0$. Without loss of generality, we assume $\|A_j\|_2 = 1$, for all j. Note that such a transformation does not change the feasibility status of either LDFP or LAP but simplifies our analysis.

Given that (1) and (2) are convex (in fact just linear) optimization problems, quite a few algorithms including polynomial-time *Interior Point methods* (IPMs) can be used to solve them. Nonetheless, in real-life classification scenarios, these problems, especially the ones with "dense" data, e.g., "dense" A matrices, remain challenging for IPMs. In these cases, each iteration of IPMs requires $O(n^3)$ *arithmetic operations* (a.o.), resulting in unacceptable overall runtimes. As a result, algorithms with computationally cheap iterations, i.e., first-order methods (FOMs) like gradient descent, are more attractive despite their inferior rate of convergences. In fact, the first algorithm suggested to solve (1), the *perceptron algorithm* by (Rosenblatt, 1958) is precisely from this class. This class of FOM algorithms, including perceptron and its variants, involves only *elementary operations* in each iteration.

Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

The time complexity per iteration is dominated by simple matrix-vector multiplication, and thus it is $O(mn)$. Due to their scalability, our focus in this paper is also limited to this class of FOMs.

Most of the FOMs directly address either (1) or (2) separately, or both simultaneously. Assuming that a.o. involved in their iterations are of the same order, one can compare the efficiency of these algorithms based on their rate of convergence, i.e., the number of iterations needed to find a feasible solution for LDFP or an ϵ-solution for LAP, The convergence rates of these algorithms are usually measured in terms of the parameters n, m ϵ, and the *margin*, $\rho(A)$:

$$\rho(A) := \max_{\|u\|_2=1} \min_{j=1,\dots,n} \frac{u^T A_j}{\|A_j\|_2}. \tag{3}$$

For example, (Novikoff, 1962) established that the rate of convergence of perceptron algorithm is $O(\frac{1}{\rho(A)^2})$. Among these quantities, $\rho(A)$, in fact, provides a measure of the difficulty of solving LDFP or LAP, or equivalently of determining the separability of data, A. LDFP is feasible if $\rho(A) > 0$, and LAP is feasible if $\rho(A) < 0$ (see (Li & Terlaky, 2013)). The smaller its magnitude, $|\rho(A)|$, the harder is to solve the corresponding problem. With our normalization ($\|A_j\|_2 = 1$ for all j), we have $|\rho(A)| \leq 1$. Unfortunately, in real classification scenarios, $|\rho(A)|$ is rather tiny, so the complexity $O(\frac{1}{\rho(A)^2})$ of the original perceptron algorithm seems not so promising, despite that it is completely independent of the size (n) of the problem.

In this paper, we suggest a single algorithm, which solves (1) or (2) simultaneously by providing either a fesibility (separability) certificate or an almost infeasibility (inseparability) certificate. While doing so, our algorithm, not only enjoys the best rate of convergence in terms of its dependence on $\rho(A)$, but also retains the simplicity of each iteration. To achieve this, we circumvent dealing directly with (1) or (2), and thus, we avoid making any assumptions on the feasibility status of either one of them. Instead, bringing a new perspective and a simplified analysis, we show that by solving a related *Bilinear Saddle Point Problem* (BSPP) via a primal-dual algorithm, one can obtain either a feasible solution for LDFP or an ϵ-solution for LAP depending on the feasibility status of the respective problem. To solve BSPP, we adopt the *Mirror Prox* (MP) algorithm, an accelerated primal-dual FOM introduced by (Nemirovski, 2004). While our suggested framework *Mirror Prox for Feasibility Problems* (MPFP) maintains the same iteration efficiency as perceptron, i.e., $O(mn)$ per iteration, we establish that MP achieves a convergence rate of $O(\frac{\sqrt{\log(n)}}{|\rho(A)|})$ for LDFP and $O(\frac{\sqrt{\log(n)}}{\epsilon})$ for LAP. To the best of our knowledge, this is the first algorithm that simultaneously solves both of these problems at these rates. Unlike perceptron algorithm, the overall rate of convergence of the MP algorithm has a very mild dependence, $O(\sqrt{\log(n)})$, on the number of training data. Nonetheless, $O(\sqrt{\log(n)})$ is in fact almost a constant factor even for extremely large n, such as $n \in [10^{10}, 10^{20}]$, and thus we claim (and also empirically show) that MP has quite competitive performance in the case of large scale classification problems. Note that such dependency is consistent with the existing literature in the case of LDFP. It is also strictly better (in terms of convergence rates) than the recent results for LAP, because MP has a factor of only $O(\sqrt{\log(n)})$, whereas the competing algorithms have a factor of $O(\sqrt{n})$, significantly limiting their computational performance. We further confirm the efficiency and scalability of our algorithms via a numerical study on both synthetic and real data. In addition to their excellent theoretical and practical performance, our study also revealed that MP-based algorithms are numerically more stable than the other state-of-the-art methods.

MPFP also offers great flexibility in adjusting to the geometry of the problem. In particular, we show that by proper customization, our framework can easily be extended to handle two important generalizations: the kernelized feasibility problems and the general conic feasibility problems. In both of these generalizations, we maintain the same rate of convergence as the original version while also retaining a cheap iteration cost. The connections of classification with saddle point problems, and the efficiency and flexibility of the MP framework discussed in this paper, open up further strikingly important possibilities for acceleration based on randomization. Particularly, the sublinear time behavior achieved by MP variants introduced in (Juditsky et al., 2013) is of great interest for future research.

2. Related Work

There is an extensive literature on deterministic and stochastic FOMs for solving classification problems (Cristianini & Shawe-Taylor, 2000; Schölkopf & Smola, 2002; Cotter et al., 2012) as well as the perceptron algorithm and its variants. In this paper, we limit our focus to deterministic FOMs and the most relevant literature. We categorize the representative literature into two parts in terms of the types of certificates, e.g., solutions to LDFP and LAP, the corresponding algorithms provide. Table 1 summarizes the rate of convergence of these algorithms. We note that the overall number of a.o. involved in each iteration of any one of these algorithms is $O(mn)$ and thus the comparison presented in Table 1 is in fact meaningful.

The first category of algorithms in Table 1 assumes that LDFP is feasible and only provides solutions for LDFP, e.g., feasibility (separability) certificates. The original perceptron (PCT) algorithm (Rosenblatt, 1958) with $O(\frac{1}{\rho(A)^2})$ rate of convergence is an example for this type of algorithms. As observed in (Saha et al., 2011), acceleration

	LDFP	LAP ϵ-solution
PCT	$O(\frac{1}{\rho(A)^2})$	N/A
SPCT	$O(\frac{\sqrt{\log(n)}}{\lvert\rho(A)\rvert})$	N/A
VN	$O(\frac{1}{\rho(A)^2})$	$O(\min(\frac{1}{\epsilon^2}, \frac{1}{\rho(A)^2}\log\frac{1}{\epsilon}))$
ISPVN	$O(\frac{\sqrt{n}}{\lvert\rho(A)\rvert}\log\frac{1}{\lvert\rho(A)\rvert})$	$O(\frac{\sqrt{n}}{\max\{\lvert\rho(A)\rvert,\epsilon\}}\log\frac{1}{\epsilon})$
MPFP	$\boldsymbol{O(\frac{\sqrt{\log(n)}}{\lvert\rho(A)\rvert})}$	$\boldsymbol{O(\frac{\sqrt{\log(n)}}{\epsilon})}$

Table 1. Summary of convergence rate of different algorithms.

of perceptron algorithm via (Nesterov, 2005)'s extra gradient based smoothing technique is possible. This technique underlies the *Smooth Perceptron* (SPCT) algorithm suggested by (Soheili & Peña, 2012), which terminates in at most $\frac{2\sqrt{2\log(n)}}{\rho(A)} - 1$ iterations. Under the assumption that LDFP is feasible, SPCT achieves the same theoretical iteration complexity as MPFP. However, SPCT does not provide infeasibility (inseparability) certificates, and verifying the feasibility status of LDFP is equivalent to solving LDFP.

Another vein of algorithms aims to either solve LDFP or provide an ϵ-solution to LAP simultaneously, without any assumption on their feasibility status beforehand. The *von Neumann* (VN) algorithm (Dantzig, 1992), is a well-known example of such a method. Combining the analysis of (Dantzig, 1992) and (Epelman & Freund, 2000), one can conclude that *i)* if LAP is feasible, then in at most $O(\min(\frac{1}{\epsilon^2}, \frac{1}{\rho(A)^2}\log\frac{1}{\epsilon}))$ iterations VN returns an ϵ-solution; *ii)* if LAP is infeasible, then VN provides a feasible solution to LDFP within $O(\frac{1}{\rho(A)^2})$ iterations. Recently, based on SPCT and VN algorithms, (Soheili & Peña, 2013) suggested the *Iterated Smooth Perceptron-Von Neumann* (ISPVN) algorithm. ISPVN finds a feasible solution for LDFP within $O(\frac{\sqrt{n}}{\lvert\rho(A)\rvert}\log(\frac{1}{\lvert\rho(A)\rvert}))$ iterations provided that LDFP is feasible, or otherwise, gives an ϵ-solution to LAP within $O(\frac{\sqrt{n}}{\lvert\rho(A)\rvert}\log(\frac{1}{\epsilon}))$ iterations. Also, an extention of ISPVN to the kernelized setting, with the same convergence rate, (c.f., Theorem 4 in (Ramdas & Peña, 2014)) is possible. We note that compared to ISPVN, MPFP achieves a significantly better performance in terms of its dependence on both the dimension, n, of the problem ($O(\sqrt{\log(n)})$ as compared to $O(\sqrt{n})$), and the condition number $\rho(A)$. On the other hand, the complexities of MPFP and ISPVN to find an ϵ-solution of LAP indicates a trade-off between n, $\rho(A)$ and ϵ so that they are not comparable. Moreover, the improved complexity of MPFP in terms of its dependence on n, also gives an affirmative answer to the conjecture of (Ramdas & Peña, 2014).

3. Notation and Preliminaries

We first introduce some notation and key concepts related to our setup and analysis. Throughout this paper, we use Matlab notation to denote vector and matrices, i.e., $[x; y]$ denotes the concatenation of two column vectors x, y.

Bilinear Saddle Point Problem (BSPP) forms the backbone of our analysis. In its most general form a BSPP is defined as

$$\max_{y\in Y}\min_{x\in X}\phi(x,y) \qquad (\mathcal{S})$$

where $\phi(x,y) = \upsilon + \langle a_1, x\rangle + \langle a_2, y\rangle + \langle y, Bx\rangle$; X, Y are nonempty convex compact sets in Euclidean spaces E_x, E_y and $Z := X \times Y$, hence $\phi(x,y) : Z \to \mathbb{R}$. Note that $(\mathcal{S})$ gives rise to two convex optimization problems that are dual to each other:

$$\begin{array}{rcll} \mathrm{Opt}(P) & = & \min_{x\in X}[\overline{\phi}(x) := \max_{y\in Y}\phi(x,y)] & (P) \\ \mathrm{Opt}(D) & = & \max_{y\in Y}[\underline{\phi}(y) := \min_{x\in X}\phi(x,y)] & (D) \end{array}$$

with $\mathrm{Opt}(P) = \mathrm{Opt}(D) = \mathrm{Opt}$, and to the *variational inequality* (v.i.), i.e., find $z_* \in Z$, such that

$$\langle F(z), z - z_*\rangle \geq 0 \ \text{ for all } z \in Z, \qquad (4)$$

where $F : Z \mapsto E_x \times E_y$ is the affine monotone operator defined by

$$F(x,y) = \left[F_x(y) = \frac{\partial\phi(x,y)}{\partial x}; F_y(x) = -\frac{\partial\phi(x,y)}{\partial y}\right].$$

It is well known that the solutions to $(\mathcal{S})$ — the saddle points of ϕ on $X \times Y$ — are exactly the pairs $z = [x; y]$ comprised of optimal solutions to problems (P) and (D). They are also solutions to the v.i. (4). For BSPP $(\mathcal{S})$, the accuracy of a candidate solution $z = [x; y]$ is quantified by the *saddle point residual*

$$\begin{array}{rcl} \epsilon_{\text{sad}}(z) & = & \overline{\phi}(x) - \underline{\phi}(y) \\ & = & \underbrace{[\overline{\phi}(x) - \mathrm{Opt}(P)]}_{\geq 0} + \underbrace{[\mathrm{Opt}(D) - \underline{\phi}(y)]}_{\geq 0}. \end{array}$$

4. General Mirror Prox Framework

MP algorithm is quite flexible in terms of adjusting to the geometry of the problem characterized by the domain of BSPP $(\mathcal{S})$, e.g., X, Y. The following components are standard in forming the MP setup for given domains X, Y, and analyzing its convergence rate:

- *Norm*: $\|\cdot\|$ on the Euclidean space E where the domain $Z = X \times Y$ of $(\mathcal{S})$ lives, along with the dual norm $\|\zeta\|_* = \max_{\|z\|\leq 1}\langle\zeta, z\rangle$.
- *Distance-Generating Function* (d.g.f.): $\omega(z)$, which is convex and continuous on Z, admits continuous on the set $Z^o = \{z \in Z : \partial\omega(z) \neq \emptyset\}$ selection $\omega'(z)$ of subgradient (here $\partial\omega(z)$ is a subdifferential of ω taken at z), and is strictly convex with modulus 1 w.r.t. $\|\cdot\|$:

$$\forall z', z'' \in Z^o : \ \langle\omega'(z') - \omega'(z''), z' - z''\rangle \geq \|z' - z''\|^2.$$

- *Bregman distance*: $V_z(u) = \omega(u) - \omega(z) - \langle \omega'(z), u - z \rangle$, where $z \in Z^o$ and $u \in Z$.
- *Prox-mapping*: Given a *prox center* $z \in Z^o$, $\text{Prox}_z(\xi) = \underset{w \in Z}{\text{argmin}} \{\langle \xi, w \rangle + V_z(w)\} : E \to Z^o$.
- *ω-center*: $z_\omega = \underset{z \in Z}{\text{argmin}}\, \omega(z) \in Z^o$ of Z.
- $\Omega = \Omega_z := \max_{z \in Z} V_{z_\omega}(z) \le \max_{z \in Z} \omega(z) - \min_{z \in Z} \omega(z)$.
- *Lipschitz constant*: $\mathcal{L}$ of F from $\|\cdot\|$ to $\|\cdot\|_*$, satisfying $\|F(z) - F(z')\|_* \le \mathcal{L}\|z - z'\|, \forall z, z'$.

Based on this setup, the general template of *Mirror Prox algorithm for Feasibility Problems* (MPFP) is given in Algorithm 1. We refer the customizations of Algorithm 1 to handle linear, kernelized, and conic feasibility problems as MPLFP, MPKFP, and MPCFP, respectively.

Algorithm 1 MPFP

Input: ω-center z_ω, step size $\{\gamma_t\}$ and ϵ.
Output: $z_t (= [x_t; y_t])$.
$t = 1; v_1 = z_\omega$;
while $\underline{\phi}(y_t) \le 0$ and $\epsilon_{\text{sad}}(z_t) > \epsilon$ **do**
  $w_t = \text{Prox}_{v_t}(\gamma_t F(v_t))$;
  $v_{t+1} = \text{Prox}_{v_t}(\gamma_t F(w_t))$;
  $z_t = \left[\sum_{s=1}^t \gamma_s\right]^{-1} \sum_{s=1}^t \gamma_s w_s$;
  $t = t + 1$;
end while

The standard customization of MPFP is based on associating a norm, $\|\cdot\|_x$, and a d.g.f., $\omega_x(\cdot)$, with domain X, and similarly $\|\cdot\|_y, \omega_y(\cdot)$ with domain Y. Then, given two scalars $\alpha_x, \alpha_y > 0$, we build the d.g.f. and ω-center, z_ω, for $Z = X \times Y$ as:

$$\omega(z) = \alpha_x \omega_x(x) + \alpha_y \omega_y(y) \quad \text{and} \quad z_\omega = [x_{\omega_x}; y_{\omega_y}],$$

where $\omega_x(\cdot)$ and $\omega_y(\cdot)$ as well as x_{ω_x} and y_{ω_y} are customized based on the geometry of the domains X, Y. Also, by letting $\xi = [\xi_x; \xi_y]$, $z = [x; y]$, our prox mapping becomes decomposable as

$$\text{Prox}_z(\xi) = \left[\text{Prox}_x^{\omega_x}\left(\frac{\xi_x}{\alpha_x}\right); \text{Prox}_y^{\omega_y}\left(\frac{\xi_y}{\alpha_y}\right)\right],$$

where $\text{Prox}_x^{\omega_x}(\cdot)$ and $\text{Prox}_y^{\omega_y}(\cdot)$ are respectively prox mappings w.r.t. $\omega_x(x)$ in domain X and $\omega_y(y)$ in domain Y. Because of space limitation, we provide the detailed derivation of the prox-mapping operators and the rationale behind our parameter choices, α_x, α_y, in the appendix.

The convergence rate of MP algorithm for solving BSPP was established by (Nemirovski, 2004) as follows:

Theorem 1. *(Nemirovski, 2004) Suppose the step sizes in the Mirror Prox algorithm satisfy $\gamma_t = \mathcal{L}^{-1}$. Then at every iteration $t \ge 1$, the corresponding solution, $z_t = [x_t; y_t]$ satisfies $x_t \in X$, $y_t \in Y$, and we have*

$$\overline{\phi}(x_t) - \underline{\phi}(y_t) = \epsilon_{\text{sad}}(z_t) \le \frac{\Omega \mathcal{L}}{t}.$$

5. Linear Feasibility Problems

In this section, we first associate a BSPP for the linear feasibility problems LDFP and LAP, and establish close connections between the solutions of these problems. Then, we discuss customization of general MPFP framework, e.g. Section 4, for simultaneously solving LDFP and LAP, and the computational complexity of the resulting MPLFP. We provide customization of MPFP to the kernelized and conic feasibility problems in Section 6. Because of space limitation, all of the proofs are provided in appendix.

5.1. BSPP Formulation for Linear Feasibility Problems

To address the linear feasibility problems, LDFP (1) or LAP (2) simultaneously, we consider the BSPP:

$$\text{Opt} = \max_{y \in \mathcal{B}_m} \min_{x \in \Delta_n} y^T A x, \tag{5}$$

where the domains X, Y of the variables x, y are $\Delta_n := \{x \in \mathbb{R}^n : \sum_{i=1}^n x_i = 1, x \ge 0\}$, i.e., a standard n-dimensional simplex, and $\mathcal{B}_m := \{y \in \mathbb{R}^m : \|y\|_2 \le 1\}$, a unit Euclidean ball in $\mathbb{R}^m$, respectively. Then $Z = \Delta_n \times \mathcal{B}_m$. Also, $\phi(x, y) = y^T A x$, $\overline{\phi}(x) = \max_{y \in \mathcal{B}_m} y^T A x$, $\underline{\phi}(y) = \min_{x \in \Delta_n} y^T A x$, and $F(x, y) = [A^T y; -Ax]$.

The connections between LDFP, LAP and BSPP, and their solutions, play an important role in our main result given in Theorem 3. These are explored in the next section.

5.2. Connections between LDFP/LAP and BSPP

We first establish a close connection between $\rho(A)$ and the objective value of BSPP (5) in Lemma 1, and then relate solutions of BSPP (5) and LDFP/LAP in Theorem 2.

Lemma 1. *(a) $\rho(A) = \max_{\|y\|_2 = 1} \min_{x \in \Delta_n} y^T A x$; (b) Whenever $\rho(A) > 0$, then $\rho(A) = \max_{\|y\|_2 \le 1} \min_{x \in \Delta_n} y^T A x = \text{Opt}$.*

Moreover, for any primal-dual algorithm solving BSPP (5), we have the following relations:

Theorem 2. *Let $z_t = [x_t; y_t]$ be the solution at iteration t of a primal-dual algorithm for solving problem (5). Suppose that the algorithm terminates at step t, when either $\underline{\phi}(y_t) > 0$ or $\epsilon_{\text{sad}}(z_t) \le \epsilon$. Then we have*

(a) If $\underline{\phi}(y_t) > 0$, then $A^T y_t > 0$; otherwise,

(b) If $\epsilon_{\text{sad}}(z_t) \le \epsilon$, then $\|A x_t\|_2 \le \epsilon$.

Theorem 2 reveals that, depending on the sign of $\rho(A)$, the primal-dual solution pair $z_t = [x_t; y_t]$ for BSPP (5) indeed corresponds to that of either LDFP or LAP. In fact, part (a) of the theorem corresponds to the case when LDFP is feasible, (since $\rho(A) > \underline{\phi}(y_t) > 0$), and y_t is a feasible solution of LDFP; part (b) is essentially the case when x_t is an ϵ-solution of LAP. The merit of such a primal-dual algorithm stated in Theorem 2 is that it can automatically identify whether LDFP or LAP is feasible during the execution and provide the corresponding solution. As a result, we do not need to assess the feasibility of either of LDFP or LAP upfront, which is already as difficult as solving the original feasibility problems.

Any primal-dual algorithm capable of solving BSPP (5) can be customized to fit the conditions of Theorem 2, and thus, is suitable for simultaneously solving LDFP/LAP. Yet the convergence rates of these algorithms can significantly differ in terms of their dependence on $\rho(A)$. In Section 5.4, we show that Mirror Prox algorithm achieves the best known performance in terms of $\rho(A)$. Further generalizations of the MP algorithm to handle kernelized and conic feasibility problems are possible as well. These generalizations retain the same rate of convergence given in Theorem 1, yet they may differ in terms of the a.o. needed for their iterations. We discuss these in Section 6.

5.3. MPLFP: Customization of MPFP for LDFP/LAP

For LDFP and LAP, we have $Z = X \times Y = \Delta_n \times \mathcal{B}_m$, and hence we pick $\omega_x(x) = \sum_{i=1}^n x_i \ln(x_i)$ and $\omega_y(y) = \frac{1}{2} y^T y$, which leads to, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$

$$[\text{Prox}_x^{\omega_x}(\zeta)]_i = \frac{x_i \exp\{-\zeta_i\}}{\sum_{k=1}^n x_k \exp\{-\zeta_k\}}, \text{ and}$$

$$[\text{Prox}_y^{\omega_y}(\zeta)]_j = \begin{cases} y_j - \zeta_j, & \text{if } \|y - \zeta\|_2 \le 1 \\ \frac{y_j - \zeta_j}{\|y-\zeta\|_2}, & \text{otherwise} \end{cases}.$$

Therefore, each iteration of Algorithm 1, involves only the computation of $F(\xi) = [A^T \xi_y; -A\xi_x]$, and

$$\text{Prox}_z(\gamma F(\xi)) = \left[\text{Prox}_x^{\omega_x}(\frac{\gamma A^T \xi_y}{\alpha_x}); \text{Prox}_y^{\omega_y}(\frac{-\gamma A \xi_x}{\alpha_y})\right].$$

This choice of d.g.f. leads to $z_\omega = \left[x_{\omega_x}; y_{\omega_y}\right]$ where $x_{\omega_x} = \frac{1}{n}\mathbf{1} \in \mathbb{R}^n$, and $y_{\omega_y} = \mathbf{0}_m$, the zero vector in $\mathbb{R}^m$. We compute the associated $\Omega = \Omega_z$ and $\mathcal{L}$ following the derivations in the appendix, and set $\gamma_t = \frac{1}{\mathcal{L}}$.

5.4. Convergence Analysis for MPLFP

In the specific case of linear feasibility problems LDFP and LAP, following the derivation of α_x and α_y presented in appendix, one can optimally select α_x and α_y to achieve $\Omega\mathcal{L} \le \sqrt{\log(n)} + \sqrt{1/2}$. Hence by combining Theorems 1 and 2, we arrive at the following main result:

Theorem 3. *Let the step sizes of MPFP be $\gamma_t = \mathcal{L}^{-1}$. Then*

(a) When $\rho(A) > 0$, MPFP terminates in at most $N = \frac{\Omega\mathcal{L}}{\rho(A)} + 1 \le \frac{\sqrt{\log(n)}+\sqrt{1/2}}{\rho(A)} + 1$ iterations with a feasible solution to LDFP given by y_N.

(b) When $\rho(A) < 0$, MPFP terminates in at most $N = \frac{\Omega\mathcal{L}}{\epsilon} + 1 \le \frac{\sqrt{\log(n)}+\sqrt{1/2}}{\epsilon} + 1$ iterations with an ϵ-solution for LAP given by x_N.

Complexity per Iteration of MPLFP: Each iteration t involves the following elementary computations with the given a.o. complexity: ***(1)*** $Ax_t, A^T y_t$: $O(mn)$. ***(2)*** Prox mapping: $O(m+n)$. ***(3)*** $\overline{\phi}(x_t) = \|Ax_t\|_2$: $O(m)$. ***(4)*** $\underline{\phi}(y_t) = \min_{i\in\{1,\ldots,n\}}(A_i^T y_t) : O(n)$. Therefore, the overall a.o. involved in each iteration of MPLFP is $O(mn)$, which is the same as that of the perceptron algorithm.

6. Generalized Feasibility Problems

There are two important extensions of the basic MPFP framework, namely the *Kernelized Feasibility Problems* (KDFP, KAP) and the *Conic Feasibility Problems* (CDFP, CAP). Both of these extensions merely require customization of the MPFP framework to the geometry of the problem by using proper representation or proper selection of prox mappings. Hence, these can easily be handled within the existing MPFP framework while still enjoying the same convergence rate of MPLFP. To the best of our knowledge, the extension of MP to the KDFP achieving $O(\frac{\sqrt{\log(n)}}{|\rho(\Psi)|})$ (or $O(\frac{\sqrt{\log(n)}}{\epsilon})$ in the case of KAP) performance is novel and the same performance of MP for the conic feasibility problem (in terms of its dependence on n and $\rho(A)$) is far superior to all of the other competing algorithms.

6.1. Kernelized Feasibility Problems

In the *kernelized* classification problems, along with the data A, we are given, a *feature map* $\Phi(\cdot) : \mathbb{R}^m \to \mathbb{F}$, which maps each data point, A_i, to a new feature in the *Reproducing Kernel Hilbert Space*, $\mathbb{F} = \mathbb{R}^d$. The feature map is specified implicitly via a *kernel*, $K_\Phi(a, a') := \langle \Phi(a), \Phi(a') \rangle$, with the motivation that explicitly computing $\Phi(a)$ can be rather time consuming, but instead the inner product computation, $\langle \Phi(a), \Phi(a') \rangle$, via the kernel $K_\Phi(a, a')$ is much easier to obtain (see (Schölkopf & Smola, 2002)). Therefore, kernelized algorithms are designed under the assumption that only "black box" access to the kernel is available (i.e., our methods work for any kernel, as long as we can compute $K_\Phi(a, a')$ efficiently). By normalization, we assume that $K_\Phi(a, a) = 1$, and $K_\Phi(a, a') \le 1$ for all $a, a' \in \mathbb{R}^m$, and in our runtime analysis, we assume kernel matrix associated with data A, i.e.,

G_Φ where $[G_\Phi]_{i,j} = K_\Phi(A_i, A_j)$, is known.

Let Q be a diagonal matrix, where $Q_{ii} = 1$ or -1 is the label of the i^{th} data point, A_i. In order to simplify the notation in our analysis, we work with the feature map $\Psi(A_i) = Q_{ii}\Phi(A_i)$ and the resulting kernel is given by $K(A_i, A_j) = \langle \Psi(A_i), \Psi(A_j)\rangle = \langle Q_{ii}\Phi(A_i), Q_{jj}\Phi(A_j)\rangle$ which is the (i,j)-th element of the kernel matrix $G := Q^T G_\Phi Q$. For notational convenience, we also let $\Psi := [\Psi(A_1), \ldots, \Psi(A_n)]$.

Thus the kernelized classification problem is equivalent to finding a feasible solution to the following Kernelized Dual Feasibility Problem (KDFP): $y^T\Psi > 0$.When no such solution exists, an inseparability certificate is provided by an ϵ-solution for the Kernelized Alternative Problem (KAP): $\Psi x = 0, \mathbf{1}^T x = 1, x \geq 0$.Therefore, one can equivalently consider the following Kernelized BSPP:

$$\max_{y\in\mathcal{B}_d} \min_{x\in\Delta^n} y^T\Psi x.$$

By a simple customization, the MPFP framework can be generalized to solve kernelized BSPP using only black box kernel oracle. Throughout the algorithm, in order avoid any explicit computation involving Ψ, we keep track of two vectors $x_t, g_t \in \mathbb{R}^n$. While the role of x_t in MPKFP remains the same, we use g_t as a surrogate for y_t. In particular, we let $g_0 = \mathbf{0}_n$ implying $y_0 = \Psi g_0$, and initialize the algorithm with $z_\omega = \left[\frac{1}{n}\mathbf{1}; \mathbf{0}_d\right] = [x_0; y_0]$. And, in all subsequent iterations, we always maintain the relation $y_t = \Psi g_t$ implicitly. This is the key modification in MPKFP. The benefit of this modification is that $\Psi^T y_t = \Psi^T\Psi g_t = \sum_{i=1}^n K(A_i, A_i)(g_t)_i$, hence we can avoid explicit computations involving Ψ. In particular, at each iteration t, the corresponding kernelized BSPP solution is given by $z_t = \left[\sum_{s=1}^t \gamma_s\right]^{-1}\sum_{s=1}^t \gamma_s[x_t; \Psi g_t]$. Based on z_t, the corresponding lower and upper bounds are given by $\underline{\phi}(y_t) = \min_{i=1,\ldots,n}(y_t)^T\Psi(A_i) = \min_{i=1,\ldots,n}\sum_{j=1}^n (g_t)_j K(A_j, A_i)$, and $\overline{\phi}(x_t) = \|\Psi x_t\|_2 = \sqrt{K(x_t, x_t)}$, using only the kernel K.

Moreover, for $y = \Psi g$, the prox mapping computations are given by

$$\text{Prox}_x^{\omega_x}\left(\frac{\gamma\Psi^T\xi_y}{\alpha_x}\right) = \frac{x\circ\eta}{x^T\eta},$$

where $\eta = \left[\exp\left\{-\frac{\gamma(Gg)_1}{\alpha_x}\right\}, \ldots, \exp\left\{-\frac{\gamma(Gg)_n}{\alpha_x}\right\}\right]$, and $\circ$ is the elementwise product, and,

$$\text{Prox}_y^{\omega_y}\left(\frac{-\gamma\Psi\xi_x}{\alpha_y}\right) = \begin{cases} \Psi(g + \frac{\gamma\xi_x}{\alpha_y}), & \text{if} \|\Psi(g + \frac{\gamma\xi_x}{\alpha_y})\|_2 \leq 1 \\ \frac{\Psi(g+\frac{\gamma\xi_x}{\alpha_y})}{\|\Psi(g+\frac{\gamma\xi_x}{\alpha_y})\|_2}, & \text{otherwise} \end{cases}.$$

Note that $\text{Prox}_x^{\omega_x}(\cdot)$ computation does not use Ψ, and in $\text{Prox}_y^{\omega_y}(\cdot)$, $\|\Psi(g + \frac{\gamma\xi_x}{\alpha_y})\|_2 = \sqrt{K\left(g + \frac{\gamma\xi_x}{\alpha_y}, g + \frac{\gamma\xi_x}{\alpha_y}\right)}$, also relies only on the kernel function K. Hence it is sufficient to keep only the (perhaps normalized) term, $g + \frac{\gamma\xi_x}{\alpha_y}$, after each $\text{Prox}_y^{\omega_y}(\cdot)$ operation.

Complexity: ***(1)*** **Per Iteration**: Given K, the complexity within each iteration is $O(n^2)$ due to Gg_t, $K\left(g + \frac{\gamma\xi_x}{\alpha_y}, g + \frac{\gamma\xi_x}{\alpha_y}\right)$, $\underline{\phi}(y_t)$ and $\overline{\phi}(x_t)$. ***(2)*** **Convergence Rate:** As the domains $X = \Delta_n$ and $Y = \mathcal{B}_d$ remain the same as MPLFP, MPKFP shares the same convergence rate as the former, except that $\rho(A)$ is replaced by $\rho(\Psi)$.

Remark: Let (x^*, g^*) be the output of MPKFP. Then, if the data is separable in the feature space, we use g^* for classification as follows: For a new testing data point, A_0, its class label is determined by the sign of $(\Phi(A_0))^T\Psi g^* = \sum_{i=1}^n Q_{ii}\langle\Phi(A_0), \Phi(A_i)\rangle g_i^* = \sum_{i=1}^n Q_{ii}K_\Phi(A_0, A_i)g_i^*$. Otherwise, the ϵ-inseparability certificate, i.e., an ϵ-solution for KAP, is given by Qx^*.

6.2. Conic Feasibility Problems

The second extension we discuss is the *Conic Dual Feasible Problem* (CDFP) of the form

$$A^*y \in \text{int}(\mathcal{K}^*) \tag{6}$$

where A^* is the conjugate linear map of a linear map A, and $\mathcal{K} \in \mathbb{R}^n$ is a proper (closed, convex, pointed with nonempty interior) cone with its dual cone $\mathcal{K}^*$.

Note that $A^*y \in \text{int}(\mathcal{K}^*)$ is equivalent to requiring $\langle x, A^*y\rangle > 0$ for all $x \in \mathcal{K}\setminus\{0\}$. Let $\boldsymbol{e}$ be a vector from $\text{int}(\mathcal{K}^*)$, as the selection of $\boldsymbol{e}$ depends on $\mathcal{K}$, below we will specify it separately for each $\mathcal{K}$. Define $\Delta(\mathcal{K}) = \{x \in \mathcal{K} : \langle \boldsymbol{e}, x\rangle = 1\}$, which is the *base* of the cone $\mathcal{K}$ given by the vector $\boldsymbol{e}$. Hence $\langle x, A^*y\rangle > 0$ for all $x \in \mathcal{K}\setminus\{0\}$ is equivalent to $\min_x\{\langle x, A^*y\rangle : x \in \Delta(\mathcal{K})\} > 0$. Moreover, the *Conic Alternative Problem* (CAP) is given by:

$$Ax = 0,\ \langle \boldsymbol{e}, x\rangle = 1,\ x \in \mathcal{K}. \tag{7}$$

Based on the same rationale of previous sections, we consider the following conic version of BSPP:

$$\max_{y\in\mathcal{B}_m} \min_{x\in\Delta(\mathcal{K})} y^T Ax. \tag{8}$$

Once again, (8), analogous to (5), is a BSPP albeit with different domains, and hence it can still be solved within the MPFP framework, achieving the same iteration complexity given in Theorem 3. The main customization of the MPFP applied to (8) as opposed to (5) is in selecting various algorithm parameters, and prox-mapping operators for the corresponding domains. In particular, one needs to specify an efficient prox-mapping operator customized to the domain $\Delta(\mathcal{K})$. We present several interesting cases of $\mathcal{K}$, where such efficient prox-mapping onto $\Delta(\mathcal{K})$ exists. These include $\mathcal{K} = \mathbb{R}^n_+$, the nonnegative orthant,

$\mathcal{K} = \mathbb{L}^n = \{x \in \mathbb{R}^n : x_n \geq \sqrt{x_1^2 + \ldots + x_{n-1}^2}\}$, the second order cone, and $\mathcal{K} = \mathbb{S}_+^n = \{X \in \mathbb{R}^{n\times n} : X = X^T, a^T X a \geq 0 \, \forall a \in \mathbb{R}^n\}$, the positive semidefinite cone. The basic settings of these cases as suggested in (Nemirovski, 2004; Juditsky et al., 2013) are as follow:

Nonnegative orthant: $\mathcal{K} = \mathcal{K}^* = \mathbb{R}^n_+$. This is precisely the case we have addressed for linear feasibility problems (see Section 5.3).

Second order cone: $\mathcal{K} = \mathcal{K}^* = \mathbb{L}^n$.

- $\boldsymbol{e} = [0, ..., 0, 1]^T$, $\Delta(\mathcal{K}) = \mathcal{B}_{n-1} \times \{1\}$.
- d.g.f: $\omega(x) = \frac{1}{2}\sum_{i=1}^{n-1} x_i^2$; $\Omega_x \leq \frac{1}{2}$ with $x_{\omega_x} = [0, ..., 0, 1]^T$.
- Given $x \in \Delta(\mathcal{K})$, $\xi \in \mathbb{R}^n$, for $i = 1, \ldots, n-1$,

$$[\text{Prox}_x(\xi)]_i = \begin{cases} x_i - \xi_i, & \text{if } \sum_{i=1}^{n-1}(x_i - \xi_i)^2 \leq 1 \\ \frac{x_i - \xi_i}{\sum_{i=1}^{n-1}(x_i - \xi_i)^2}, & \text{otherwise} \end{cases}$$

and $[\text{Prox}_x(\xi)]_n = 1$. Note that the computational complexity of this prox mapping is $O(n)$ as discussed in the MPLFP setup (see Section 5.3).

Positive semi-definite cone: $\mathcal{K} = \mathcal{K}^* = \mathbb{S}_+^n$.
For any two symmetric matrices A, B, we let $\langle A, B\rangle = \text{Tr}(AB)$ as the corresponding inner product.

- $\boldsymbol{e} = I_n$, $\Delta(\mathcal{K}) = \{X \in \mathbb{S}_+^n : X \succeq 0, \text{Tr}(X) = 1\}$(Flat Spectrahedron).
- d.g.f: matrix entropy $\omega(X) = \text{Entr}(\lambda(X)) = \sum_i \lambda_i(X)\log(\lambda_i(X))$, where $\lambda_i(X)$ is the i-th eigenvalue of X and resulting $\Omega_X \leq 4\log(n)$ with $x_{\omega_x} = \text{Diag}(\frac{1}{n})$.
- Given $X \in \mathbb{S}_+^n$, $\Xi \in \mathbb{S}^n$, computing $\text{Prox}_X(\Xi)$ reduces to computing the eigenvalue decomposition of the matrix X, which also leads to the computation of $\omega'(X)$, and subsequent eigenvalue decomposition of the matrix $H = \Xi - \omega'(X)$. Let $H = U\text{Diag}(h)U^T$ be the eigenvalue decomposition of H, where $\text{Diag}(h)$ stands for the diagonal matrix with diagonal elements from h. Then computing $\text{Prox}_X(\Xi)$ reduces to constructing the matrix $W = U\text{Diag}(w)U^T$ where $w = \text{argmin}_{z\in\mathbb{R}^n}\{\langle \text{Diag}(h), \text{Diag}(z)\rangle + \omega(\text{Diag}(z)) : \text{Diag}(z) \in \Delta(\mathcal{K})\}$, which is identical to the computation of the Prox function in the simplex setup. The computational complexity of this prox mapping is $O(n^3)$ per iteration due to the two singular value decompositions ($O(n^3)$) and simple a.o. of $O(n)$.

Discussion of MPCFP and conic ISPVN: The MPFP framework is quite flexible: in particular, it can be easily adjusted to handle a variety of domains Z by properly selecting $\omega(\cdot)$ and thus $\text{Prox}_z(\cdot)$. Besides, for the aforementioned cones, the computational cost of each iteration of MPCFP is quite low. While ISPVN is capable of handling the same conic feasibility problems, the analysis of (Soheili & Peña, 2013; Ramdas & Peña, 2014) is based on Euclidean d.g.f.'s, and thus the respective operations involved for projections onto these domains are much more expensive (involving sorting operations), and hence they are inferior to the ones presented above.

7. Numerical Experiments

In this section, we conduct an empirical study on the MPFP framework and compare it with other baseline algorithms, using both synthetic data (for MPLFP and LAP) and real data (for MPKFP). All the codes are written in Matlab 2011b, and are ran in a single-threaded fashion on a Linux server with 6 dual core 2.8GHz CPU and 64 GB memory.

In the implementation of MPLFP or MPKFP, we select α_x, α_y as described in the appendix, for a normalized matrix A, i.e., $\|A_i\|_2 = 1$ for all i. We implemented the Smooth Perceptron (SPCT) (Soheili & Peña, 2012), normalized Perceptron (PCT) (Ramdas & Peña, 2014) and the normalized Von-Neumann (VN) algorithm (Soheili & Peña, 2012) for comparison. We also implemented the ISPVN (Soheili & Peña, 2013) and ISNKPVN (Ramdas & Peña, 2014), but as the authors do not provide a parameter selection strategy, we did a search on the space and choose the one with the best performance, i.e., the parameter denoted by γ in (Soheili & Peña, 2013) (Theorem 1) was set to 2. We note that our numerical study is based on the comparison of the basic implementations of all of these algorithms including ours as well. There may be ways for small improvement leading better computational performance for any one of them. Nevertheless, guided by our theoretical findings, we do not expect any major changes in the overall conclusions drawn.

7.1. Synthetic data

We first compare the performance of different algorithms (non-kernelized versions) on synthetic data. In our instance generation for LDFP, κ serves as a proxy for the condition number $\rho(A)$, i.e., a larger κ corresponds to a larger $\rho(A)$ in (3). We provide details of our instance generation for both LDFP and LAP in the appendix.

Experimental Setup. We test the methods on a wide spectrum of (m, n) combinations. Given a combination, we generate 20 instances for LDFP and 20 for LAP on which all the methods are tested. For each method, the iteration limit is 10^6 and runtime limit is 5×10^4s. When either one of these limits is reached, we report a "TO (Timeout)". We empirically observe that, occasionally, SPCT suffers

from numerical overflow, and ISPVN, frequently, falls into a dead loop. We report a "Fail" in the table if a method fails in all of the 20 instances. Table 2 summarizes the average runtime for each algorithm over the instances that the algorithm successfully solved. Because of space limitations, we present the number of iterations for LDFP and for the effect of $\rho(A)$, and our results on LAP instances in the appendix.

Performance on LDFP. We set $\kappa = 1$ for the case LDFP, and, by varying m, n, we generate normalized instances of $A \in \mathbb{R}^{m\times n}$ such that $\|A_j\|_2 = 1$ for all j. In all of the instances generated, ISPVN algorithm failed, and hence we omitted it from comparison. As indicated by Table 2, MPLFP outperforms all the other methods in terms of runtime for any pair of (m, n). The runner-up algorithm SPCT is obviously much slower while PCT and VN perform even equally worse. Moreover, as (m, n) increases, the time savings achieved by MPLFP become more significant. More importantly, even on the very large size of high-dimensional data ($m = 10^3, n = 5 \times 10^6$), MPLFP finds a feasible solution for LDFP within 10 hours, while the baseline methods either run out of time (PCT and VN) or fail (SPCT and ISPVN) on large scale problems ($m = 10^2, n = 5 \times 10^5$). This highlights the suitability of MPLFP for large-scale learning.

The Effect of $\rho(A)$. We tested the performance of algorithms by varying κ as a proxy for $\rho(A)$ on instances with $(m, n) = (100, 5000)$ and $\|A_j\|_2 = 1$ for all j. For various values of $\kappa \in [1, 10]$ (i.e., for feasible LDFP), the runtime of each algorithm is reported in Figure 4. It is not surprising that as $\kappa(\rho(A))$ becomes larger, which means that the data become "more separable," the runtime is shorter for all of the methods. Nonetheless, MPLFP is still considerably faster than its closest competitor in all cases.

	With column-normalized matrix, A			
(m, n)	MPLFP	SPCT	PCT	VN
$(10^2, 5 \times 10^3)$	**1.3**	4.3	98.5	127.3
$(10^3, 5 \times 10^3)$	**5.5**	16.2	142.2	141.5
$(10^2, 5 \times 10^4)$	**112.7**	455.0	TO	TO
$(10^3, 5 \times 10^4)$	**318.1**	1301.0	TO	TO
$(10^2, 5 \times 10^5)$	**25506**	Fail	TO	TO
$(10^3, 5 \times 10^5)$	**22471**	Fail	TO	TO

Table 2. Runtime (second) of all methods for LDFP ($\kappa = 1$). Iteration limit is 10^6 and runtime limit is 5×10^4s.

7.2. Real Data: CIFAR-10 Image Classification

We test the efficiency of MPKFP in training classifiers with kernel method. For comparison, we also implement the Kernelized Perceptron (KPCT), Kernelized Von-Neumann (KVN), and ISNKVPN (Ramdas & Peña, 2014). Our testbed is the CIFAT-10 (Krizhevsky, 2009) dataset[1], which contains 60,000 color images in 10 mutually exclusive classes, with 6,000 images per class. Each image A_i is encoded as a $m = 3072$ dimensional vector.[2] We evenly pick around 1,000 images in each class to construct a training set of $n = 10,000$ in total, and repeat this process to form a testing set of the same size. We train one-vs-rest binary classifier for each class, and hence obtain 10 classifiers in total. After that, each test image is run on by all the classifiers and its predicted label corresponds to the one that gives the largest output value. We choose Radial Basis Function (RBF) kernel given by $K_\Phi(A_i, A_j) = \exp\{-5.5(\|A_i - A_j\|_2)\}$.

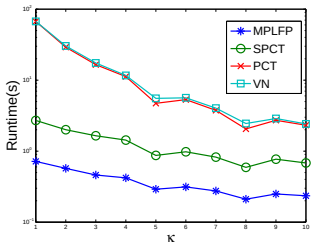

Figure 1. κ ($\rho(A)$) v.s. runtime.

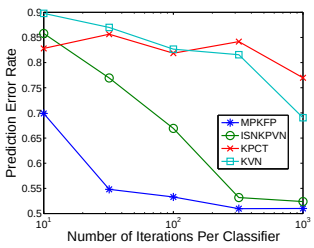

Figure 2. Iteration v.s. error.

We run all of the algorithms, and at different pre-specified iteration limits of $N \in \{10, 32, 100, 320, 1000\}$, we collect their respective solutions. We treat these solutions as the corresponding classifiers, and benchmark their prediction quality on the test set. Figure 2 shows how the prediction error rate drops as the training iteration number per classifier increases. It can be seen that, within the same iteration limit, MPKFP outperforms all the other methods in terms of the prediction error rate. In particular, the error rate obtained by perceptron after 1,000 iterations is still higher than that achieved within merely 10 iterations by MPKFP, which highlights the suitability of MPKFP for fast classification training scenarios.

8. Conclusions

We build a framework based on BSPP to either find a binary classifier (DFP) or return a near infeasibility certificate if there is none (AP). We adopt the accelerated primal-dual algorithm, Mirror Prox, to solve the BSPP and achieve the rate of convergence $O(\frac{\sqrt{\log n}}{|\rho(A)|})$ for DFP and $O(\frac{\sqrt{\log n}}{\epsilon})$ for AP. Our results also extend to the general kernelized and conic problems, all of which share the same rate of convergence. We further confirm the efficiency and numerical stability of our approach by a numerical study on both synthetic and real data. Future work includes exploring whether these methods can be modified to achieve sublinear time behavior, or allow for rapid incremental retraining in active learning scenarios.

[1] http://www.cs.utoronto.ca/~kriz/cifar.html

[2] To achieve a lower prediction error rate, one can use other advanced features,which is not this paper's focus.

References

Blum, Avrim and Dunagan, John. Smoothed analysis of the perceptron algorithm for linear programming. In *SODA*, pp. 905–914, 2002.

Chvatal, Vasek. *Linear Programming*. Macmillan, 1983.

Cotter, Andrew, Shalev-Shwartz, Shai, and Srebro, Nathan. The kernelized stochastic batch perceptron. In *ICML*, 2012.

Cristianini, Nello and Shawe-Taylor, John. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

Dantzig, George Bernard. An ϵ-precise feasible solution to a linear program with a convexity constraint in $1/\epsilon^2$ iterations independent of problem size. Technical Report 92-5, Stanford University, 1992.

Epelman, Marina and Freund, Robert M. Condition number complexity of an elementary algorithm for computing a reliable solution of a conic linear system. *Math. Program.*, 88(3):451–485, 2000.

Juditsky, Anatoli, Kılınç-Karzan, Fatma, and Nemirovski, Arkadi. Randomized first order algorithms with applications to ℓ_1-minimization. *Math. Program.*, 142(1-2): 269–310, 2013.

Krizhevsky, Alex. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

Li, Dan and Terlaky, Tamás. The duality between the perceptron algorithm and the von neumann algorithm. In *Modeling and Optimization: Theory and Applications*, volume 62, pp. 113–136. 2013.

Nemirovski, Arkadi. Prox-method with rate of convergence $o(1/t)$ for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

Nesterov, Yurii. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal on Optimization*, 16(1):235–249, 2005.

Novikoff, Albert B. J. On convergence proofs for perceptrons. Technical report, 1962.

Ramdas, Aaditya and Peña, Javier. Margins, kernels and non-linear smoothed perceptrons. In *ICML*, 2014.

Rosenblatt, Frank. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

Saha, Ankan, Vishwanathan, S. V. N., and Zhang, Xinhua. New approximation algorithms for minimum enclosing convex shapes. In *SODA*, pp. 1146–1160, 2011.

Schölkopf, Bernhard and Smola, Alex J. *Learning with kernels*. The MIT Press, 2002.

Soheili, Negar and Peña, Javier. A smooth perceptron algorithm. *SIAM Journal on Optimization*, 22(2):728–737, 2012.

Soheili, Negar and Peña, Javier. A primal-dual smooth perceptron-von neumann algorithm. In *Discrete Geometry and Optimization*, volume 69, pp. 303–320. 2013.

Appendix: Saddle Points and Accelerated Perceptron Algorithms

This appendix provides the omitted proofs from the main text, detailed descriptions and derivations for mirror prox framework, and supplement for numerical experiments.

9. Proofs Omitted from the Main Text

Proof of Lemma 1.

Proof. (a) Definition of $\rho(A)$ and the fact $\min_{j=1,\dots,n} y^T A^j = \min_{x\in\Delta_n} y^T Ax$ implies $\rho(A) = \max_{\|y\|_2=1} \min_{x\in\Delta_n} y^T Ax$.

(b) Let $f(y) := \min_{x\in\Delta_n} y^T Ax$, and note that $f(y)$ is concave in y. Using (a), $\rho(A) = \max_{\|y\|_2=1} f(y)$, i.e., $\exists\, y^*$ s.t. $\|y^*\|_2 = 1$ and $\rho(A) = f(y^*) = \max_{\|y\|_2=1} f(y) \le \max_{\|y\|_2\le 1} f(y)$ where the last inequality follows from the obvious relaxation. Whenever $\rho(A) > 0$, we claim that this relaxation is tight. If not, then $\exists\ \bar{y}$ s.t. $\|\bar{y}\|_2 < 1$, and $f(\bar{y}) > f(y^*) = \rho(A) > 0 = f(0)$, and thus $\bar{y} \neq 0$. Now consider $\hat{y} := \frac{\bar{y}}{\|\bar{y}\|_2}$. Note that $\|\hat{y}\|_2 = 1$ and $f(\hat{y}) = \frac{1}{\|\bar{y}\|_2} f(\bar{y}) > f(\bar{y}) > f(y^*)$, where the first inequality is due to $\|\bar{y}\|_2 < 1$. But this contradicts to the optimality of y^*, and hence we conclude $\rho(A) = \max_{\|y\|_2\le 1} f(y) = \text{Opt}$. □

Proof of Theorem 2.

Proof. (a) $0 < \underline{\phi}(y_t) = \min_{x\in\Delta_n} y_t^T Ax \le y_t^T Ae^i = (A^T y_t)_i$ for all $i = 1, \dots, n$, and hence $A^T y_t > 0$.

(b) In this case, $\underline{\phi}(y_t) \le 0$ and $\epsilon_{\text{sad}}(z_t) = \overline{\phi}(x_t) - \underline{\phi}(y_t) \le \epsilon$ implying $\overline{\phi}(x_t) \le \underline{\phi}(y_t) + \epsilon \le \epsilon$. Moreover, $\overline{\phi}(x_t) = \max_{\|y\|_2\le 1} y^T Ax = \|Ax_t\|_2$, and hence we have $\|Ax_t\|_2 \le \epsilon$. □

Proof of Theorem 3.

Proof. From the definition of Opt, $\overline{\phi}(\cdot)$, and $\underline{\phi}(\cdot)$, for any $x \in X$ and $y \in Y$, we have $\underline{\phi}(y) \le \text{Opt} \le \overline{\phi}(x)$.

(a) If $\rho(A) > 0$, by Lemma 1, we have $\rho(A) = \text{Opt}$. Combining the above relations with Theorem 1, for every iteration $t \ge 1$, we obtain

$$\overline{\phi}(x_t) - \frac{\Omega\mathcal{L}}{t} \le \underline{\phi}(y_t) \le \rho(A) \le \overline{\phi}(x_t) \le \underline{\phi}(y_t) + \frac{\Omega\mathcal{L}}{t}.$$

Note that by rearranging terms in this inequality, we get

$$\rho(A) - \frac{\Omega\mathcal{L}}{t} \le \underline{\phi}(y_t) \le \rho(A),$$

By setting $N = \frac{\Omega\mathcal{L}}{\rho(A)} + 1$, we get $\underline{\phi}(y_N) \ge \rho(A) - \frac{\Omega\mathcal{L}}{N} > 0$. By Theorem 2, we know y_N is a feasible solution of LDFP.

(b) If $\rho(A) < 0$, then $\underline{\phi}(y_t) < 0$. Let $N = \frac{\Omega\mathcal{L}}{\epsilon} + 1$. By Theorem 1, $\epsilon_{\text{sad}}(z_N) \le \frac{\Omega\mathcal{L}}{N} < \epsilon$. Then by Theorem 2, x_N is an ϵ-solution of LAP. □

10. Mirror Prox Setup for Feasibility Problems

Let $Z = X \times Y$ where $X,\ Y$ are two given domains with respective norms $\|\cdot\|_x$ and $\|\cdot\|_y$, their dual norms $\|\cdot\|_{(x,*)}$ and $\|\cdot\|_{(y,*)}$, and d.g.f.'s $\omega_x(\cdot)$ and $\omega_y(\cdot)$. Given two scalars $\alpha_x, \alpha_y > 0$, we build the setup for the Mirror Prox algorithm, i.e., $\|\cdot\|$ and d.g.f. $\omega(\cdot)$ for the domain Z, where $z = [x; y]$, as follows:

$$\|z\| = \sqrt{\alpha_x \|x\|_x^2 + \alpha_y \|y\|_y^2}$$

with the dual norm

$$\|\zeta\|_* = \sqrt{\frac{1}{\alpha_x}\|\zeta_x\|_{(x,*)}^2 + \frac{1}{\alpha_y}\|\zeta_y\|_{(y,*)}^2}$$

and corresponding d.g.f. given by

$$\omega(z) = \alpha_x \omega_x(x) + \alpha_y \omega_y(y).$$

With these choices, we arrive at $\Omega = \Omega_z \le \alpha_x \Omega_x + \alpha_y \Omega_y$ where $\Omega_x := \max_{x\in X} V_{x_\omega}(x) \le \max_{x\in X} \omega(x) - \min_{x\in X} \omega(x)$ and Ω_y is defined similarly. Moreover, our prox mapping $\text{Prox}_z(\xi)$ becomes decomposable, i.e., by letting $\xi =$

$[\xi_x; \xi_y]$, we have

$$\begin{aligned}
&\text{Prox}_z^{\omega}(\xi)\\
&= \underset{w\in Z}{\text{argmin}}\,\{\langle \xi, w\rangle + V_z(w)\}\\
&= \underset{w\in Z}{\text{argmin}}\,\{\langle \xi, w\rangle + (\omega(w) - \omega(z) - \langle \omega'(z), w - z\rangle)\}\\
&= \underset{(w_x;w_y)\in Z}{\text{argmin}}\,\{\langle \xi_x, w_x\rangle + \langle \xi_y, w_y\rangle + \alpha_x\omega_x(w_x) + \alpha_y\omega_y(w_y)\\
&\qquad - \alpha_x\langle \omega_x'(x), w_x - x\rangle - \alpha_y\langle \omega_y'(y), w_y - y\rangle\}\\
&= \underset{(w_x;w_y)\in Z}{\text{argmin}}\,\{\langle \xi_x, w_x\rangle + \langle \xi_y, w_y\rangle + \alpha_x\omega_x(w_x) + \alpha_y\omega_y(w_y)\\
&\qquad - \alpha_x\langle \omega_x'(x), w_x\rangle - \alpha_y\langle \omega_y'(y), w_y\rangle\}\\
&= \left[\underset{w_x\in X}{\text{argmin}}\{\langle \xi_x - \alpha_x\omega_x'(x), w_x\rangle + \alpha_x\omega_x(w_x)\}\ ;\right.\\
&\qquad \left.\underset{w_y\in Y}{\text{argmin}}\{\langle \xi_y - \alpha_y\omega_y'(y), w_y\rangle + \alpha_y\omega_y(w_y)\}\right]\\
&= \left[\text{Prox}_x^{\omega_x}\left(\frac{\xi_x}{\alpha_x}\right);\ \text{Prox}_y^{\omega_y}\left(\frac{\xi_y}{\alpha_y}\right)\right].
\end{aligned}$$

Furthermore for bilinear saddle point problems, we have

$$\begin{aligned}
\|F(z) - F(z')\|_* &= \|(A^T(y-y'); A(x-x'))\|_*\\
&= \sqrt{\frac{1}{\alpha_x}\|A^T(y-y')\|_{(x,*)}^2 + \frac{1}{\alpha_y}\|A(x-x')\|_{(y,*)}^2}\\
&\le \mathcal{L}\|z - z'\|
\end{aligned}$$

with $\mathcal{L} := \sqrt{\frac{1}{\alpha_x}\mathcal{L}_{xy}^2 + \frac{1}{\alpha_y}\mathcal{L}_{yx}^2}$ where

$$\begin{aligned}
\mathcal{L}_{xy} &\ge \max_y\left\{\|A^Ty\|_{(x,*)} :\ \|y\|_y \le 1\right\} \quad \text{and}\\
\mathcal{L}_{yx} &\ge \max_x\left\{\|Ax\|_{(y,*)} :\ \|x\|_x \le 1\right\}.
\end{aligned}$$

Hence we arrive at

$$\Omega\mathcal{L} \le (\alpha_x\Omega_x + \alpha_y\Omega_y)\sqrt{\frac{1}{\alpha_x}\mathcal{L}_{xy}^2 + \frac{1}{\alpha_y}\mathcal{L}_{yx}^2}.$$

By minimizing this upper bound in terms of α_x, α_y, we get

$$\begin{aligned}
\alpha_x &= \frac{\mathcal{L}_{xy}}{\sqrt{\Omega_x}(\mathcal{L}_{xy}\sqrt{\Omega_x} + \mathcal{L}_{yx}\sqrt{\Omega_y})} \text{ and}\\
\alpha_y &= \frac{\mathcal{L}_{yx}}{\sqrt{\Omega_y}(\mathcal{L}_{xy}\sqrt{\Omega_x} + \mathcal{L}_{yx}\sqrt{\Omega_y})},
\end{aligned}$$

which leads to

$$\mathcal{L} = \mathcal{L}_{xy}\sqrt{\Omega_x} + \mathcal{L}_{yx}\sqrt{\Omega_y},$$

and

$$\Omega \le \alpha_x\Omega_x + \alpha_y\Omega_y \le 1.$$

Setup for Linear Feasibility Problems

In the particular case of LDFP, taking into account the setup $X\times Y = \Delta_n\times\mathcal{B}_m$, we select $\|\cdot\|_x = \|\cdot\|_1$ and $\|\cdot\|_y = \|\cdot\|_2$ with d.g.f.'s $\omega_x(x) = \text{Entr}(x) := \sum_{i=1}^n x_i\ln(x_i)$, i.e., the Entropy d.g.f., and $\omega_y(y) = \frac{1}{2}y^Ty$ Euclidean d.g.f. with

$$\text{Prox}_z^{\omega}(\xi) = \left[\text{Prox}_x^{\omega_x}\left(\frac{\xi_x}{\alpha_x}\right);\ \text{Prox}_y^{\omega_y}\left(\frac{\xi_y}{\alpha_y}\right)\right].$$

By considering the form of the d.g.f.s and the associated domains, we can utilize the closed form expressions for the corresponding optimal solutions in the above optimization problems for prox mapping computations and hence arrive at, for $i = 1, \ldots, n$,

$$[\text{Prox}_x^{\omega_x}(\xi_x)]_i = \frac{x_i\exp\{-[\xi_x]_i\}}{\sum_{j=1}^n x_j\exp\{-[\xi_x]_j\}} \quad \text{and}$$

$$[\text{Prox}_y^{\omega_y}(\xi_y)]_i = \begin{cases} y_i - [\xi_y]_i, & \text{if } \|y-\xi_y\|_2 \le 1\\ \frac{y_i - [\xi_y]_i}{\|y-\xi_y\|_2}, & \text{otherwise}\end{cases}.$$

Moreover by selecting $z_\omega = \left[\frac{1}{n}\mathbf{1}; \mathbf{0}_m\right]$, where $\mathbf{1}$ stands for the vector in $\mathbb{R}^n$ with all coordinates equal to 1 and $\mathbf{0}_m$ denotes the zero vector in $\mathbb{R}^m$, we get

$$\Omega_x = \log(n), \text{ and } \Omega_y = \frac{1}{2} \tag{9}$$

Also, $\mathcal{L} = \sqrt{\frac{1}{\alpha_x}\mathcal{L}_{xy}^2 + \frac{1}{\alpha_y}\mathcal{L}_{yx}^2}$ with $\mathcal{L}_{xy} \ge \max_y\{\|A^Ty\|_\infty : \|y\|_2 \le 1\}$ and $\mathcal{L}_{yx} \ge \max_x\{\|Ax\|_2 : \|x\|_1 \le 1\}$. In fact in this current setup, one can pick

$$\mathcal{L}_{xy} = \mathcal{L}_{yx} = \max_j\|A_j\|_2 = 1$$

due to our normalization. Hence by considering $\Omega \le \alpha_x\Omega_x + \alpha_y\Omega_y$, we arrive at

$$\Omega\mathcal{L} \le (\alpha_x\Omega_x + \alpha_y\Omega_y)\sqrt{\frac{1}{\alpha_x}\mathcal{L}_{xy}^2 + \frac{1}{\alpha_y}\mathcal{L}_{yx}^2}.$$

The minimization of this upper bound in α_x, α_y, as discussed above, leads to the corresponding parameters associated with Mirror Prox algorithm with $\Omega \le 1$ and $\mathcal{L} = \mathcal{L}_{xy}\sqrt{\Omega_x} + \mathcal{L}_{yx}\sqrt{\Omega_y}$. Therefore, in the case of linear feasibility problems LDFP and LAP, we have

$$\mathcal{L} = \mathcal{L}_{xy}\sqrt{\Omega_x} + \mathcal{L}_{yx}\sqrt{\Omega_y} = \sqrt{\log(n)} + \sqrt{1/2}$$

and hence

$$\Omega\mathcal{L} \le \sqrt{\log(n)} + \sqrt{1/2}.$$

Setup for Kernelized and Conic Feasibility Problems

The kernelized case is the same as the linear case. For the conic case, as mentioned before, $\Omega\mathcal{L}$ determines the convergence rate of the MPFP framework. As mentioned in the previous section, the domains involved X, Y determine both $\Omega\mathcal{L}$ and the prox mappings. For conic feasibility problems, in all cases Y domain is $\mathcal{B}_m$, always resulting in $\Omega_y = \frac{1}{2}$, yet Ω_x is case dependent, for $\mathcal{K} = \mathbb{R}^n_+$ and $\mathcal{K} = \mathbb{S}^n_+$, we have $\Omega_x = O(\log(n))$ and for $\mathcal{K} = \mathbb{L}^n$, we have $\Omega_x = \frac{1}{2}$, which leads to the resulting rate of convergences which is no worse than $O(\frac{\sqrt{\log(n)}}{\epsilon})$ for Mirror Prox algorithm.

11. Supplement for Numerical Experiments

In this section, we supplement our real data results and present our synthetic data generation procedures and the additional experimental results on the synthetic data, which could not have been incorporated to the main text because of space limitation. Recall that, in the following tables, 'TO' means 'Timeout' and 'Fail' means the algorithm fails due to running into numerical issues or falling into dead loop.

Synthetic Data for LDFP Instances. As for the LDFP instance generation, we follow the scheme suggested by (Soheili & Peña, 2012) to construct the matrix A: we generate random vectors $\bar{y} \in \mathbb{R}^m$, $v \in \mathbb{R}^n$ where $\bar{y}$ is drawn from a uniform distribution on the unit sphere $\{y \in \mathbb{R}^m : \|y\|_2 = 1\}$ and each v_i is i.i.d. from Uniform$[0, 1]$. We set $\bar{A} = B + \bar{y}(\kappa v^T - \bar{y}^T B)$ where $\kappa \geq 0$ is a parameter and $B \in \mathbb{R}^{m\times n}$ is a randomly generated matrix, with each entry independently drawn from the standard normal distribution. Note that $\bar{A}^T\bar{y} = \kappa v$; hence, if $\kappa > 0$, LDFP is feasible with $\bar{y}$ as a solution. Further, κ serves as a proxy for the condition number $\rho(\bar{A})$, i.e., a larger κ corresponds to a larger $\rho(\bar{A})$ in (3). Since normalization does not change feasibility status of LDFP or $\rho(A)$, our A matrix is obtained by normalizing $\bar{A}$ such that all columns have their Euclidean norms equal to 1. Nevertheless, we observed that normalization drastically alters the computational behavior of two algorithms, SPCT and ISPVN. In particular, we observed that in all of the normalized instances, ISPVN algorithm failed by falling into a deadloop. Therefore we compared our algorithms against SPCT, PCT, and VN for normalized instances. On the other hand, we also performed a number of tests with unnormalized data, e.g., with the matrix $\bar{A}$ described above, and observed that in these cases, SPCT algorithm failed while we were able to run the ISPVN algorithm. We discuss the corresponding comparisons below.

Iterations of the LDFP. We present results on the number of iterations for algorithms tested in the LDFP experiments (normalized instances with A) in Table 3. This complements the corresponding runtime results, i.e., Table 2 in the LDFP section of the main text. From Table 3, we observe that the MPLFP outperforms competing algorithms in terms of number of iterations, under any (m, n) combination. This behavior also supports the superior performance of MPLFP in terms of its solution time given in Table 3.

We also present the results on the unnormalized data matrix $\bar{A}$, in Tables 3 and 4, which also support the superior performance of MPLFP.

	With column-normalized matrix, A			
(m, n)	MPLFP	SPCT	PCT	VN
$(10^2, 5\times10^3)$	**204.6**	605.3	92413.4	91450.1
$(10^3, 5\times10^3)$	**146.0**	338.9	15009.3	14864.0
$(10^2, 5\times10^4)$	**1692.2**	5144.2	TO	TO
$(10^3, 5\times10^4)$	**730.8**	2409.9	TO	TO
$(10^2, 5\times10^5)$	**11580**	Fail	TO	TO
$(10^3, 5\times10^5)$	**6970**	Fail	TO	TO
	With unnormalized matrix, $\bar{A}$			
(m, n)	MPLFP	ISPVN	PCT	VN
$(10^2, 5\times10^3)$	**203.2**	1601.1	90649	90841
$(10^3, 5\times10^3)$	**30.9**	68.0	14869	14909
$(10^2, 5\times10^4)$	**2210.4**	Fail	TO	TO
$(10^3, 5\times10^4)$	**283.4**	2525.8	TO	TO
$(10^2, 5\times10^5)$	**18149**	Fail	TO	TO
$(10^3, 5\times10^5)$	**3415.4**	Fail	TO	TO

Table 3. Number of iterations of all methods for LDFP ($\kappa = 1$). Iteration limit is 10^6 and runtime limit is 5×10^4s.

	With unnormalized matrix, $\bar{A}$			
(m, n)	MPLFP	ISPVN	PCT	VN
$(10^2, 5\times10^3)$	**1.2**	19.1	98.3	110.8
$(10^3, 5\times10^3)$	**1.4**	4.9	171.3	181.2
$(10^2, 5\times10^4)$	**90.4**	Fail	TO	TO
$(10^3, 5\times10^4)$	**136.7**	1963.6	TO	TO
$(10^2, 5\times10^5)$	**43748**	Fail	TO	TO
$(10^3, 5\times10^5)$	**22753**	Fail	TO	TO

Table 4. Runtime (second) of all methods for LDFP ($\kappa = 1$). Iteration limit is 10^6 and runtime limit is 5×10^4s.

Iterations as κ ($\rho(A)$) changes. As a complementary of experiment on the effect of $\rho(A)$ (on normalized instances of A), we include Figure 3 here to show the corresponding number of iterations needed for different algorithms under different $\rho(A)$. Again, MPLFP is faster than the nearest competitor by at least one order of magnitude.

We also test the effect of $\rho(\bar{A})$ on the unnormalized instances with $\bar{A}$ matrix. The corresponding runtime and number of iteration figures are given in Figures 4 and 5.

Synthetic Data for LAP Instances. We generated difficult LAP instances as follows: Given an integer $r \geq 3$, and number $\theta > 1$, we let $n = 2^r$ and generate a square $n \times n$ matrix for A. We control the difficulty of the instance with θ. We first generate $q = \lfloor \frac{(n-1)}{2} \rfloor$ points given by

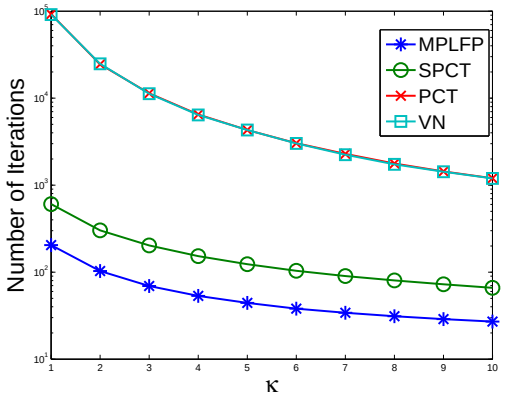

Figure 3. κ ($\rho(A)$) vs iteration with normalized matrix, A.

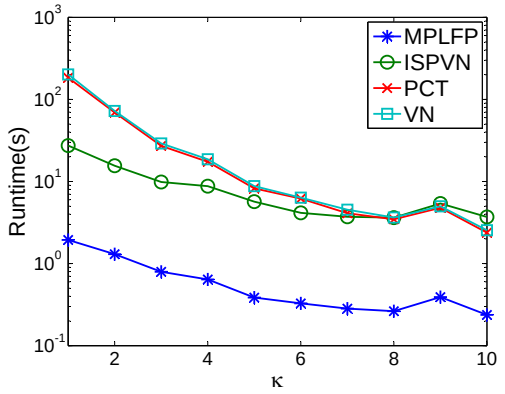

Figure 4. κ ($\rho(\bar{A})$) vs runtime with unnormalized matrix, $\bar{A}$.

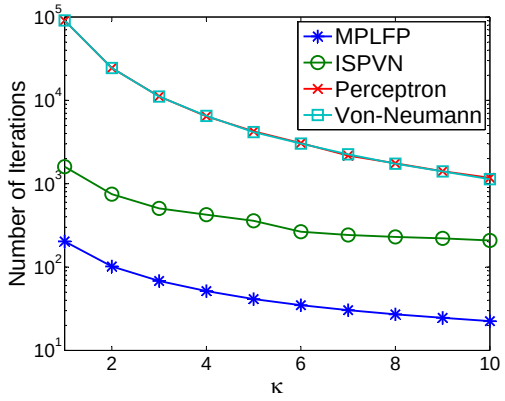

Figure 5. κ ($\rho(\bar{A})$) vs iteration with unnormalized matrix, $\bar{A}$.

$S = \left\{\cos(\pi \frac{k}{(q-1)}) : 0 \le k \le q-1\right\} := \{s_0, \ldots, s_{q-1}\}$ and then translate and scale these points to obtain $p_i = \frac{1}{\theta} + (s_i - \min_j s_j)\frac{1-\frac{1}{\theta}}{\max_j s_j - \min_j s_j}$ for $i = 0, \ldots, q-1$, and the set $P = \{p_0, \ldots, p_{q-1}\}$. With this construction we have $\min\{p : p \in P\} = \frac{1}{\theta}$ and $\max\{p : p \in P\} = 1$. We define $(n-1) \times (n-1)$ diagonal matrix B with $2q$ diagonal entries $\{\pm\sqrt{p} : p \in P\}$, and set the remaining $n-1-2q$ diagonal entries to 1. Then we let H be a Hadamard matrix of size $n \times n$ which is normalized to have all column norms equal to 1. We then generate a random $\bar{x}$ such that $\bar{x} \in \Delta_n = \{x \in \mathbb{R}^n : \sum_i x_i = 1,\ x_i \ge 0\ \forall i\}$. Then we construct $\bar{A}$ matrix of the following form

$$\bar{A} = H^T \begin{bmatrix} c & f^T \\ f & B \end{bmatrix} H,$$

where we select c, f to ensure that $\bar{A}\bar{x} = 0$. In particular, this leads to a square system of linear equations in c, f as a result of the special structure of B and H. Our A matrix is then obtained by normalizing $\bar{A}$ to have all Euclidean norms of the columns equal to 1. Note that by normalizing $\bar{x}$, we will also obtain $x \in \Delta_n$ such that $Ax = 0$. In all of our instances for LAP, we work with the normalized matrix A. In all of the generated instances, ISPVN algorithm failed, therefore the comparison is done against only VN algorithm.

Performance on LAP. We first test the performance of different methods to find an ϵ-solution for LAP by setting $\theta = 5$ and $\epsilon = 10^{-3}$. We report the performance comparison of MPLFP and VN in Tables 5 and 6. As suggested by our theoretical convergence rates, MPLFP algorithm is orders of magnitude faster than VN algorithm on the LAP instances.

	ϵ-solution of LAP ($\theta = 5$)	
(m, n)	MPLFP	VN
(1024, 1024)	**2.5**	21.6
(2048, 2048)	**10.6**	125.8
(4096, 4096)	**69.2**	948.6
(8192, 8192)	**284.5**	5465.9
(16384, 16384)	**1014.0**	25426

Table 5. Runtime (second) of all methods for LAP. Iteration limit is 10^6 and runtime limit is 5×10^4s.

The Effect of θ. We also test the effect of θ by varying θ from the set $5, 10, 100, 1000, 10000$ on instances of size $n = m = 2048$. The results on runtime and iteration numbers are provided in Tables 7 and 8.

Real Data: CIFAR-10 Image Classification. We complement the performance of the algorithms on the test data of CIFAR-10 Image Classification data set, e.g., Figure 2, with the corresponding comparison of their training errors

	ϵ-solution of LAP ($\theta = 5$)	
(m, n)	MPLFP	VN
(1024, 1024)	**265.2**	8763.6
(2048, 2048)	**283.4**	12352.4
(4096, 4096)	**291.6**	16353.8
(8192, 8192)	**297.1**	22838.1
(16384, 16384)	**293.5**	31894.3

Table 6. Number of iteration of all methods for LAP. Iteration limit is 10^6 and runtime limit is 5×10^4s.

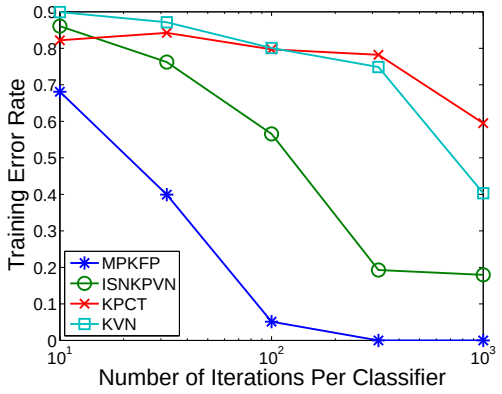

Figure 6. Iteration v.s. training error.

θ	5	10	10^2	10^3	10^4
MPLFP	10.4	10.7	14.2	13.7	13.5
VN	123.0	121.3	149.9	148.4	143.7

Table 7. Runtime (second) of MPLFP for LAP as θ varies.

θ	5	10	10^2	10^3	10^4
MPLFP	284.1	285.2	296.3	288.2	286.9
VN	12341	12149	11946	11800	11712

Table 8. Number of iterations of MPLFP for LAP as θ varies.

on the training data set, e.g., Figure 6. Figure 6 indicates that within roughly 300 iterations, MPKFP achieves a training error of almost 0, and therefore supports the exceptional performance of MPKFP that we have observed in Figure 2.