J. Appl. Numer. Optim. 3 (2021), No. 1, pp. 3-20
Available online at http://jano.biemdas.com
https://doi.org/10.23952/jano.3.2021.1.02

AUGMENTED LAGRANGIAN - FAST PROJECTED GRADIENT ALGORITHM WITH WORKING SET SELECTION FOR TRAINING SUPPORT VECTOR MACHINES

MAYOWA AREGBESOLA[1], IGOR GRIVA[2,*]

[1]*Department of Computational and Data Sciences, George Mason University, Fairfax, VA 22030, USA*
[2]*Department of Mathematical Sciences, George Mason University, Fairfax, VA 22030, USA*

Abstract. We present an algorithm for training Support Vector Machines (SVM) for classification based on fast projected gradient, augmented Lagrangian methods and a working set selection principle. The algorithm is capable of training SVM with tens of thousands data points within seconds on a modern personal computer system. The algorithm can be parallelized and therefore it is suitable for multi processor environment. We describe the algorithm, provide numerical results for solving medium size problems and discuss future directions of speeding up the SVM training.

Keywords. Machine learning; Support vector machines; Fast projected gradient; Augmented Lagrangian; Working set selection.

1. INTRODUCTION

A Support Vector Machine (SVM) [1, 2] is a supervised machine learning technique used for classification, regression [3], or ranking learning tasks [4]. SVM were initially developed for classification and have been extended for regression and rank learning. The original form of SVM was a binary classifier where the output of the learned function is either positive or negative. A multiclass classification can be implemented by combining multiple binary classifiers using the pairwise coupling or one class against the rest methods [5]. The main value of the support vector machines is in their ability to achieve accurate generalization and their foundation on well developed beautiful statistical learning theory [2].

SVM training requires solving a large scale convex quadratic optimization problem that has a dense Hessian of the objective function. Due to convexity of the SVM training problem, finding a global solution is guaranteed as long as the Karusch-Kuhn-Tucker (KKT) conditions are satisfied. Therefore computational efficiency and memory requirements are usually the two main factors to consider when the choice of a particular optimization algorithm is made [6].

Since the 1990s, there have been many attempts to develop efficient algorithms for SVM training. First, interior-point methods (IPM) have been used [7]. Later, exterior-point methods [8, 9] based nonlinear rescaling principle introduced by R. Polyak [10] have been shown to be competitive to IPM.

*Corresponding author.
E-mail addresses: maregbes@gmu.edu (M. Aregbesola), igriva@gmu.edu (I. Griva).
Received October 2, 2020; Accepted December 4, 2020.

Both interior- and exterior-point methods rely on solving m-dimensional linear systems of equations, where m is the number of training examples. Therefore they are efficient for training SVM only up to a few thousands data points. In the pursuit to train SVM on larger data sets, decomposition methods such as sequential minimal optimization (SMO) [11] gained popularity due to their low memory usage requirement and efficiency. They stand as state of the art methods up to date. Good tutorial introductions to SVM training methods can be found in [3, 4, 5, 6, 12, 13].

In more recent times, however, some further attempts to improve the SMO have lead to appearance of decomposition methods that use larger working sets than does the SMO (see e.g IPSMO-2 algorithm in [14]). The SMO uses a working set that consists of just one pair of data points, while IPSMO-2 allows a selection of a larger size of a working set to fit comfortably in an operating memory of a node of a high performance computing (HPC) multi-node system.

On the other hand, in the last decade, fast gradient methods related to the family of optimal first-order methods introduced by Nesterov [15], have gained their popularity. Beck and Teboulle analyzed convergence of FISTA algorithm [16]. Later R. Polyak established convergence bounds for fast projected gradient methods (FPGM) [17, 18]. In particular he analyzed the FPGM convergence for solving a nonnegative least square problem. The theory developed in [18] can be used to justify convergence of the FPGM for minimization of a general convex quadratic function for bounded variables.

Training SVM requires solving a large scale convex quadratic problem with a linear constraint and simple bounds for variables. Therefore the fast projected gradient method is a natural choice for solving the SVM problem, since the projection on the set defined by simple bounds are computationally inexpensive. The linear constraint can be enforced with the help of augmented Lagrangian method. Theoretical analysis of the corresponding augmented Lagrangian - fast projected gradient method can be found in [19].

This paper describes an efficient decomposition based algorithm that allows an efficient training of support vector machines. Unlike the methods that require solving linear systems, the algorithm employs a fast projected gradient method used for the minimization of augmented Lagrangian. Such a combination, augmented Lagrangian - fast projected gradient method (AL-FPGM) has been introduced in [20] for training SVM and analyzed for convergence in [19]. Using the first-order based method allows to gain an improvement in efficiency over the second-order methods [20]. However, the algorithm developed here uses also a modified scheme for working set selection that further significantly improves the SVM training efficiency. The implemented working set selection method is a modification of a well known principle developed in [14] and [21].

The paper is organized as follows. The next section describes the SVM training problem and provides a description of AL-FPGM for training SVM. Section 3 describes a new working set selection principle responsible for an efficient implementation of AL-FPGM. Section 4 provides numerical results. Section 5 contains some concluding remarks and future directions of further improving an efficiency of the developed algorithm.

2. Augmented Lagrangian - Fast projected gradient method for SVM

Let $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ be a set of m labeled data points where $\mathbf{x}_i \in \mathbb{R}^n$ is a vector of features and $y_i \in \{-1, 1\}$ represent the label indicating the class to which $\mathbf{x}_i$ belongs. To train

the SVM, we need to find $\alpha^* = (\alpha_1^*, \ldots, \alpha_m^*)^T$ that solve the following problem:

$$\begin{aligned} \min_{\alpha} \; & -\sum_{i=1}^{m} \alpha_i + \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) \\ \text{s.t.} \; & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & 0 \le \alpha_i \le C, \quad i = 1, \ldots, m. \end{aligned} \tag{2.1}$$

Matrix Q with elements $Q_{ij} = y_i y_j K(x_i, x_i)$ is positive semidefinite but usually dense. Therefore a large number of training points m results in significant operating memory requirements thus leading to inefficiencies for many general purpose optimization algorithms.

A relatively simple and efficient AL-FPGM method that is capable of training medium size SVM with up to a few tens of thousands of data points was proposed in [20]. The algorithm takes advantage of two methods: one is the fast projected gradient method for solving a minimization problem on the box constraints [18] and the second is an augmented Lagrangian method [22, 23] employed to satisfy the only linear equality constraint. The AL-FPGM projects the primal variables onto the "box-like" set: $0 \le \alpha_i \le C, \quad i = 1, \ldots, m$.

Using the following definitions

$$\begin{aligned} f(\alpha) &= -\sum_{i=1}^{m} \alpha_i + \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) \\ g(\alpha) &= \sum_{i=1}^{m} y_i \alpha_i \end{aligned} \tag{2.2}$$

and the bounded set:

$$Box = \{\alpha \in \mathbb{R}^m : 0 \le \alpha_i \le C, \; i = 1, \ldots, m\}, \tag{2.3}$$

the optimization problem (2.1) can be rewritten as follows:

$$\begin{aligned} \min_{\alpha} \; & f(\alpha) \\ & g(\alpha) = 0 \\ & \alpha \in Box. \end{aligned} \tag{2.4}$$

The augmented Lagrangian is defined as

$$\mathscr{L}_\mu(\alpha, \lambda) = f(\alpha) - \lambda g(\alpha) + 0.5 \mu g(\alpha)^2, \tag{2.5}$$

where $\lambda \in \mathbb{R}$ the Lagrange multiplier that corresponds to the only equality constraint and $\mu > 0$ is the scaling parameter.

The augmented Lagrangian method is a sequence of inexact minimizations of $\mathscr{L}_\mu(\alpha, \lambda)$ in α on the *Box* set

$$\hat{\alpha} \approx \alpha(\lambda) = \operatorname*{argmin}_{\alpha \in Box} \mathscr{L}_\mu(\alpha, \lambda) \tag{2.6}$$

followed by updating the Lagrange multiplier λ :

$$\hat{\lambda} = \lambda - \mu g(\hat{\alpha}). \tag{2.7}$$

For the stopping criteria in (2.6), we use the following function that measures the violation of the first order optimality conditions for (2.6):

$$v(\alpha,\lambda) = \max_{1\le i\le m} v_i(\alpha,\lambda), \tag{2.8}$$

where

$$v_i(\alpha,\lambda) = \begin{cases} \left|\nabla_\alpha \mathscr{L}_\mu(\alpha,\lambda)_i\right|, & 0<\alpha_i<C, \\ \max\left\{0,-\nabla_\alpha \mathscr{L}_\mu(\alpha,\lambda)_i\right\}, & \alpha_i = 0, \\ \max\left\{0,\nabla_\alpha \mathscr{L}_\mu(\alpha,\lambda)_i\right\}, & \alpha_i = C. \end{cases} \tag{2.9}$$

For the final stopping criteria for the augmented Lagrangian method, we use $\mu(\alpha,\lambda) = \max\{v(\alpha,\lambda), |g(\alpha)|\}$, which measures the violation of the optimality conditions for (2.4).

Algorithm 1 Augmented Lagrangian-Fast Projected Gradient Method

Initialization

Initialization
$\alpha^{[0]} \in \mathbb{R}^m$ (starting point, not necessarily feasible) $\alpha^{[0]} = 0$
$\lambda^{[0]} \in \mathbb{R}$ (initial "guestimate" of Lagrange multiplier vector) $\lambda^{[0]} = 0$
$\mu_0 > 0$ (initial value of scaling parameter)
$\varepsilon > 0$ (Required accuracy)
$0 < \theta < 1$
$\delta \ge 1$
rec = $\max\{v(\alpha,\lambda), |g(\alpha)|\}$

Run the Augmented Lagragian Algorithm

while $rec > \varepsilon$ **do**
    find $\alpha : v(\alpha,\lambda) \le \theta * rec$ using FPGM
    $\lambda = \lambda - \mu g(\alpha)$
    $rec = \min(rec, v(\alpha,\lambda))$
    $k = \delta k$
end while

Run the FPGM

procedure FPGM
    Input (α,λ)
    Set $\bar{\alpha} = \alpha, t = 1$ Select $L > 0$
    while $v(\alpha,\lambda) > \theta * rec$ **do**
        Set $\hat{\alpha} = P_{Box}\left(\alpha - \frac{1}{L}\nabla_\alpha \mathscr{L}_\mu(\alpha,\lambda)\right)$.
        Set $\bar{t} = 0.5\left(1+\sqrt{1+4t^2}\right)$.
        Set $\alpha = \hat{\alpha} + \frac{t-1}{\bar{t}}(\hat{\alpha} - \bar{\alpha})$.
        $\bar{\alpha} = \hat{\alpha}, t = \bar{t}$.
    end while
    Output $\hat{\alpha}$.
end procedure

procedure P_{Box}
    for $i = 1,\ldots,m$ **do**
        if $\alpha_i < 0$ **then**
            $\alpha_i = 0$
        end if
        if $\alpha_i > C$ **then**
            $\alpha_i = C$
        end if
    end for
end procedure

The fast projected gradient method (FPGM) requires estimation of the Lipschitz constant $L > 0$ of the gradient $\nabla_\alpha \mathscr{L}_\mu$ so that the inequality

$$\left\|\nabla_\alpha \mathscr{L}_\mu(\alpha_1,\lambda) - \nabla_\alpha \mathscr{L}_\mu(\alpha_2,\lambda)\right\| \le L\|\alpha_1-\alpha_2\| \tag{2.10}$$

holds for any α_1, $\alpha_2 \in \mathbb{R}^m$ [18, 9]. The gradient and the Hessian of $\mathscr{L}_\mu(\alpha,\lambda)$ are as follows:

$$\begin{aligned} \nabla_\alpha \mathscr{L}_\mu(\alpha,\lambda) &= Q\alpha - e - \left(\lambda - \mu \sum_{i=1}^{m} y_i\alpha_i\right) y \\ \nabla^2{}_{\alpha\alpha} \mathscr{L}_\mu(\alpha,\lambda) &= Q + \mu y y^T \end{aligned} \tag{2.11}$$

where Q is an m by m matrix with the elements $Q_{ij} = y_i y_j K(x_i,x_i)$, $i = 1,\ldots,m$, $j = 1,\ldots,m$, $y = (y_1,\ldots,y_m)^T$, $e = (1,\ldots,1)^T$.

Since $\mathscr{L}_\mu$ is of quadratic form with respect to α, $L = \left\|Q + \mu y y^T\right\|$, where the matrix spectral norm is the largest singular value of a matrix.

Algorithm 1 describes the AL-FPGM.This algorithm does not use any decomposition or working set selection technique. It uses all the data points and serves as a reference for comparison with a more advance algorithm that takes advantage of working sets. The convergence of Algorithm 1 was established in [19].

3. Decomposition

Since the matrix Q is dense, often a memory shortage prohibits forming Q, keeping it in the memory, and solving the SVM problem when the number of training data points is large. Therefore decomposition methods based on a working set selection play an important role for training a large scale SVM. These methods consider only a small subset of variables per iteration [24, 25] and thus require less memory usage.

Suppose B is a subset of considered variables called the working set. Since each iteration involves only B columns of the matrix Q, the decompositions methods use the operating memory economically [24]. A subset B of the variables is chosen to be optimized while the rest of the variables remain fixed. Therefore operating memory requirements can be substantially reduced. Moreover, for large problems the overall training efficiency can be attained. The algorithm repeats the "select working set then optimize" process until the global optimality conditions are satisfied. While B denotes the working set with l variables, N denotes the non-working set with $(m-l)$ variables. Then, α, y and Q can be correspondingly written as:

$$\alpha = \begin{bmatrix} \alpha_B \\ \alpha_N \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_B \\ \mathbf{y}_N \end{bmatrix}, \quad Q = \begin{bmatrix} \mathbf{Q}_{BB} & \mathbf{Q}_{BN} \\ \mathbf{Q}_{NB} & \mathbf{Q}_{NN} \end{bmatrix}$$

The optimization subproblem can be rewritten

$$\begin{aligned} \min_{\alpha_B} \ & \frac{1}{2}\begin{bmatrix} \alpha_{\mathrm{B}}^T & \alpha_{\mathrm{N}}^T \end{bmatrix} \begin{bmatrix} \mathbf{Q}_{BB} & \mathbf{Q}_{BN} \\ \mathbf{Q}_{NB} & \mathbf{Q}_{NN} \end{bmatrix} \begin{bmatrix} \alpha_{\mathrm{B}} \\ \alpha_{\mathrm{N}} \end{bmatrix} - \begin{bmatrix} \mathbf{e}_{\mathrm{B}}^T & \mathbf{e}_{\mathrm{N}}^T \end{bmatrix} \begin{bmatrix} \alpha_{\mathrm{B}} \\ \alpha_{\mathrm{N}} \end{bmatrix} \\ \text{s.t.} \ & \alpha_{\mathrm{B}}^T y_{\mathrm{B}} + \alpha_N^T y_N = 0 \\ & 0 \le \alpha_{\mathrm{B}} \le C \end{aligned} \tag{3.1}$$

with a fixed α_{N}, this is the same as

$$\begin{aligned} \min_{\alpha_B} \ & \frac{1}{2}\alpha_{\mathrm{B}}^T \mathbf{Q}_{\mathrm{BB}}\alpha_{\mathrm{B}} + \alpha_{\mathrm{B}}^T\left(\mathbf{Q}_{\mathrm{BN}}\alpha_N - \mathbf{e}_B\right) \\ \text{s.t.} \ & \alpha_{\mathrm{B}}^T y_{\mathrm{B}} + \alpha_N^T y_N = 0 \\ & 0 \le \alpha_{\mathrm{B}} \le C. \end{aligned} \tag{3.2}$$

The reduced problem (3.2) can be solved much faster than the original problem (2.1). The resulting algorithm is shown as Algorithm 2.

Algorithm 2 Decomposition method

1: **while** global minimum not reached **do**
2: find working set B
3: find α_B for working set B using a quadratic problem solver.
4: Update α with α_B
5: **end while**

As one can see from the description of Algorithm 2, there are three main questions that need to be answered. The first question is if such a decomposition scheme will converge to the solution of the original SVM problem (2.1), the second questions is how to select a working set. Both questions have been addressed in [21]. The third question is what quadratic problem solver to be used to solve the subproblem (3.2). We suggest to use the AL-FPGM. This section also proposes below a *pWSS* scheme, which is a modification of existing working set selection scheme developed as a part of *IPSMO-2* algorithm [14], that we found efficient.

Below we describe the working set selection (*WSS*) scheme developed as a part of *IPSMO-2* algorithm alongside with some general working set selection rules.

3.1. **Working Set Selection.** An important issue of any decomposition method is how to select the working set B. Below we describe a working set selection (WSS) method that uses the second order information [21]. The WSS methods show faster convergence than a previously developed a maximum violating pair method [27].

The KKT condition of (3.1) shows that there is a scalar b such that

$$\begin{array}{llllll} y_t = 1, \alpha_t < C & \Rightarrow & (Q\alpha - \mathbf{e})_t + b \ge 0 & \Rightarrow & b \ge -(Q\alpha + \mathbf{e})_t = -\nabla f(\alpha)_t, \\ y_t = -1, \alpha_t > 0 & \Rightarrow & (Q\alpha - \mathbf{e})_t - b \le 0 & \Rightarrow & b \ge (Q\alpha + \mathbf{e})_t = \nabla f(\alpha)_t, \\ y_t = -1, \alpha_t < C & \Rightarrow & (Q\alpha - \mathbf{e})_t - b \ge 0 & \Rightarrow & b \le (Q\alpha + \mathbf{e})_t = \nabla f(\alpha)_t, \\ y_t = 1, \alpha_t > 0 & \Rightarrow & (Q\alpha - \mathbf{e})_t + b \le 0 & \Rightarrow & b \le -(Q\alpha + \mathbf{e})_t = -\nabla f(\alpha)_t, \end{array} \tag{3.3}$$

where $f(\alpha) \equiv \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$ and $\nabla f(\alpha)$ is the gradient of $f(\alpha)$.

This can be rewritten as

$$\begin{aligned} \nabla f(\alpha) + by_t \ge 0, \quad & \text{if } \alpha_t < C, \\ \nabla f(\alpha) + by_t \le 0, \quad & \text{if } \alpha_t > 0. \end{aligned}$$

Alternatively, it can be stated that a vector α is a stationary point of (3.1) if and only if there is a number b and two non-negative vectors ρ and θ such that

$$\begin{aligned} & \nabla f(\alpha) + by_t = \rho - \theta, \\ & \rho_i\alpha_i = 0, \ \theta_i(C - \alpha_i) = 0, \ \rho_i \ge 0, \ \theta_i \ge 0, \ i = 1, \ldots, m. \end{aligned}$$

Consider the sets:

$$I_{up}(\alpha) \equiv \{t \mid y_t = 1, \alpha_t < C \text{ or } y_t = -1, \alpha_t > 0\},$$
$$I_{low}(\alpha) \equiv \{t \mid y_t = -1, \alpha_t < C \text{ or } y_t = 1, \alpha_t > 0\}.$$

Then, we have

$$i \in \underset{t \in I_{up}(\alpha^k)}{\arg\max} \{-y_t \nabla f(\alpha)_t\}, \tag{3.4}$$

$$j \in \underset{t \in I_{low}(\alpha^k)}{\arg\min} \{-y_t \nabla f(\alpha)_t\}. \tag{3.5}$$

Define

$$q_t(\alpha) = -y_t \nabla f(\alpha)_t.$$

From (3.3), we see that the inequality

$$q_i(\alpha) \leq q_j(\alpha) \tag{3.6}$$

implies that α is a feasible and optimal solution of (3.1) at a stationary point of (3.1). The largest difference $q_i - q_j$ for the pair that violates (3.6) the most is compared against requested accuracy of the solution. It is used in the stopping criteria of Algorithm 2.

Let us define the quantities:

$$a_{ij} = K_{ii} + K_{jj} - 2K_{ij} > 0,$$
$$\widehat{a}_{ij} = \begin{cases} a_{ij}, & \text{if } a_{ij} > 0, \\ \tau \end{cases}$$
$$b_{ij} = q_i(\alpha) - q_j(\alpha) > 0,$$

where τ is a small positive number. Then, one can select

$$i \in \underset{t \in I_{up}(\alpha)}{\arg\max} \{q_t(\alpha)\}, \tag{3.7}$$

and

$$j \in \underset{t \in I_{low}(\alpha)}{\arg\min} \left\{ \frac{b_{it}^2}{\widehat{a}_{it}} : q_t(\alpha) < q_i(\alpha) \right\}. \tag{3.8}$$

If a working set is selected as a single pair $B = i, j$, then that leads to a sequential minimal optimization algorithm (*SMO*).

3.2. **p-Pairs Working set Selection (*pWSS*).** To obtain p working pairs, p indices i are chosen from I_{up} and the corresponding j are selected from I_{low}. This is a generalization of the *SMO* algorithm [11, 26] described as *IPSMO-2* algorithm in [14] . The *SMO* algorithm selects a single pair of data points as a working set, while *IPSMO-2* selects larger working sets containing p pairs in general. The rule how the pairs are selected is based on the maximum **KKT** violation, similar to that of the *SMO*.

In the attempt to keep an important information about how often a particular data point is selected in a working set, we have modified the *WSS* described above. We use that information to rank data points according to their frequencies in working sets. In this section we present a modified working set selection *IPSMO-2*, which we call ***pWSS***. Unlike *IPSMO-2*, which for each i finds only a single j using (3.8), ***pWSS*** finds for each i a set of indices J_i as described below. Then the sets J_i are used to select p pairs. As demonstrated in the next section, *pWSS* scheme allows to train SVM efficiently.

Suppose p pairs working set are chosen from the data as the working set for each decomposition iteration. The data points $i_s \in I_{up},\ s = 1,\ldots,p$ are selected the same way as WSS using (3.7). For each $i \in I_{up}$, select $J_i \subset I_{low}$.

$$J_i = \{t \in I_{low}(\alpha),\ q_t(\alpha) < q_i(\alpha)\},$$

where J_i are multiple violating pairs and $1 \leq |J_i| \leq p$.

The p most frequently occurring indices in all J_i for all the $i_s, s = 1,\ldots,p$ are selected to make p pairs in the working set. This differs from the IPSMO-2 method, where for each i, the corresponding j is selected according to (3.8). In our approach, multiple violating pairs of each i are selected and added to a pool of "*violators*".

The selection of J_i for each i can be done in parallel and it is not dependent on other choices of J_i for a different i. The *pWSS* rule is described as Algorithm 3.

Algorithm 3 *pWSS*

p = number of pairs
for $k = 0,1,2,\ldots,p$ **do**
    find $J_i \in I_{low}$ for every $i_k \in I_{up}$ using WSS second order. This can be done in parallel. J_i is an ordered list of violating pairs.
end for
$B_J = \emptyset$
for $i_k, k = 0,1,2,\ldots,p$ **do**
    Select $J_i = \left\{t \in I_{low}\left(\alpha^k\right),\ q_t(\alpha^k) < q_i(\alpha^k)\right\}$
    if $J_i \neq \emptyset$ **then**
        $B = B \cup i_k$
        $B_J = B_J \cup J_i$
    end if
end for
select top p occurring elements in B_J.
$B = B \cup B_{J_k}, k = 0,1,2,\ldots,p$

TABLE 1. Training results without decomposition

			AL-FPGM				MATLAB			
test	nData	nFeatures	Total iterations	Training time	Time per data size	objective value	Total iterations	Training time	Time per data size	objective value
Arcene	100	10,000	18	0.00612	6.12E-05	-4.93E+01	5	0.00675	6.75E-05	-4.93E+01
Seeds	210	7	158	0.01268	6.04E-05	-5.85E+01	10	0.01235	5.88E-05	-5.85E+01
Dexter	300	20,000	24	0.03422	1.14E-04	-3.00E+02	5	0.06174	2.06E-04	-3.00E+02
HabermanSurvival	306	3	98	0.01215	3.97E-05	-6.92E+02	7	0.02194	7.17E-05	-6.92E+02
BreastCancerWisconsin	569	30	46	0.01061	1.86E-05	-2.62E+02	5	0.06524	1.15E-04	-2.62E+02
Balance Scale	625	4	463	0.03776	6.04E-05	-3.60E+02	10	0.08682	1.39E-04	-3.60E+02
Dorothea	800	100,000	55	0.30581	3.82E-04	-1.41E+02	5	0.36854	4.61E-04	-1.41E+02
CNAE9DataSet	1,080	856	63	0.03668	3.40E-05	-2.00E+02	5	0.34231	3.17E-04	-2.00E+02
BanknoteAuthentication	1,372	4	140	0.05971	4.35E-05	-6.85E+01	9	0.33467	2.44E-04	-6.85E+01
HCVEgyptian	1,385	29	63	0.03378	2.44E-05	-1.53E+02	5	0.21885	1.58E-04	-1.53E+02
ContraceptiveMethodChoice	1,473	9	39	0.04227	2.87E-05	-6.69E+02	5	0.46053	3.13E-04	-6.69E+02
MadelonDataSet	2,600	500	46	0.05195	2.00E-05	-1.00E+03	5	0.45312	1.74E-04	-1.00E+03
StatSatLog	4,435	36	73	0.38119	8.60E-05	-1.51E+03	5	2.93383	6.62E-04	-1.51E+03
PageBlocksClassification	5,473	10	252	1.2543	2.29E-04	-9.22E+02	5	4.06224	7.42E-04	-9.22E+02
ZeroPenDigits	7,494	16	66	0.22247	2.97E-05	-6.51E+02	5	1.47892	1.97E-04	-6.51E+02
HumanActivityRecognition	10,299	561	121	0.567	5.51E-05	-2.11E+03	5	11.32824	1.10E-03	-2.11E+03
EpilepticSeizureRecognition	11,500	179	119	3.51063	3.05E-04	-3.68E+03	5	78.46099	6.82E-03	-3.68E+03
GasSensorArrayDrift	13,910	128	128	5.9015	4.24E-04	-6.65E+03	5	38.29408	2.75E-03	-6.65E+03
EegEyeState	14,980	14	132	6.98412	4.66E-04	-7.41E+03	5	49.49192	3.30E-03	-7.41E+03
Magictelescope	19,020	11	506	31.20835	1.64E-03	-8.58E+03	5	194.86633	1.02E-02	-8.58E+03
				50.6733	**2.1E-04**			**383.3494**	**1.4E-03**	
				Total Time	**Average**			**Total Time**	**Average**	

TABLE 2. Results comparison with decomposition pWSS

			ALFGM					QuadProg				
test	nData	nFeatures	nDecomp	total_ite	time	Time per data size	obj_val	nDecomp	total_ite	time	Time per data size	obj_val
Arcene	100	10000	3	43	0.0463	4.63E-04	-4.93E+01	3	14	0.0277	2.77E-04	-4.93E+01
Seeds	210	7	6	447	0.0742	3.53E-04	-5.85E+01	4	57	0.0506	2.41E-04	-5.85E+01
Dexter	300	20000	4	109	0.1158	3.86E-04	-3.00E+02	3	15	0.1201	4.00E-04	-3.00E+02
HabermanSurvival	306	3	7	298	0.0608	1.99E-04	-6.92E+02	11	70	0.1464	4.78E-04	-6.92E+02
BreastCancerWisconsin	569	30	10	105	0.0684	1.20E-04	-2.62E+02	11	55	0.1235	2.17E-04	-2.62E+02
Balance Scale	625	4	12	2454	0.3236	5.18E-04	-3.60E+02	4	26	0.0949	1.52E-04	-2.22E+02
Dorothea	800	100000	12	123	1.0223	1.28E-03	-1.41E+02	12	57	0.9943	1.24E-03	-1.41E+02
CNAE9DataSet	1080	856	11	168	0.1732	1.60E-04	-2.00E+02	11	53	0.6206	5.75E-04	-2.00E+02
BanknoteAuthentication	1372	4	4	15364	0.3158	2.30E-04	-6.33E+01	11	122	0.5533	4.03E-04	-6.78E+01
HCVEgyptian	1385	29	13	188	0.1497	1.08E-04	-1.53E+02	14	110	0.3383	2.44E-04	-1.53E+02
ContraceptiveMethodChoice	1473	9	13	164	0.2190	1.49E-04	-6.69E+02	14	70	0.6147	4.17E-04	-6.69E+02
MadelonDataSet	2600	500	4	206	0.1355	5.21E-05	-1.00E+03	3	15	0.2705	1.04E-04	-1.00E+03
StatSatLog	4435	36	11	303	1.0218	2.30E-04	-1.51E+03	10	47	3.8312	8.64E-04	-1.51E+03
PageBlocksClassification	5473	10	12	451	1.2003	2.19E-04	-9.22E+02	11	55	5.5628	1.02E-03	-9.22E+02
ZeroPenDigits	7494	16	10	269	0.7249	9.67E-05	-6.51E+02	11	55	2.8959	3.86E-04	-6.51E+02
HumanActivityRecognition	10299	561	8	344	0.8727	8.47E-05	-2.11E+03	10	46	9.3495	9.08E-04	-2.11E+03
EpilepticSeizureRecognition	11500	179	21	493	5.6446	4.91E-04	-3.68E+03	21	103	24.2059	2.10E-03	-3.68E+03
GasSensorArrayDrift	13910	128	29	541	8.7746	6.31E-04	-6.65E+03	33	164	22.7575	1.64E-03	-6.65E+03
EegEyeState	14980	14	27	561	9.0059	6.01E-04	-7.41E+03	27	135	21.2971	1.42E-03	-7.41E+03
Magictelescope	19020	11	41	1521	17.2744	9.08E-04	-8.58E+03	42	211	39.7710	2.09E-03	-8.58E+03
					47.2238	**3.64E-04**				**133.6257**	**7.59E-04**	
					Total Time	**Average**				**Total Time**	**Average**	

4. Implementation and Numerical Results

This section presents results of numerical experiments with the augmented Lagrangian-fast projected gradient method (AL-FPGM) implemented in Matlab© 2020b and the interior-point method (IPM) that comes with Matlab's QuadProg subroutine. For both AL-FPGM and the IPM, we compare the SVM training efficiency with and without *pWSS* decomposition method.

We used a desktop with Intel® Xeon® 2.50GHz, 12 Cores, 64 GB RAM.

For testing, we have selected 20 binary classification training data sets with a number of training examples ranging from 100 to 19,020. In all cases, the data was normalized. $C = 100$ was used for all tests. $\gamma = 0.5$ was used for the RBF kernel. The scaling parameter μ used for all the experiments is $\mu = 0.1$. The accuracy of solution was selected $\varepsilon = 10^{-2}$. SVM training times using AL-FPGM were compared with results obtained using Matlab's *QuadProg* with and without *pWSS* decompoisiton scheme.

The fast projected gradient method (FPGM) requires estimation of the Lipschitz constant $L \geq 0$ in (2.10) for the gradient of $\mathscr{L}_k$. To estimate L, the maximum eigenvalue of $Q + \mu yy^T$ is estimated using the power method. This estimate is updated in step 3 of procedure FPGM of Algorithm 1.

Tables 1-2 provide numerical results for training SVM with the FPGM and an interior-point method (IPM) implemented in Matlab QP solver *QuadProg*. Table 1 shows the results without the use of the decomposition method i.e. all the data points are used simultaneously. In this case, the full $m \times m$ matrix Q is computed. Table 2 shows the results for training the SVM with the new working set selection method *pWSS* with both AL-FPGM and IPM.

For each data set the tables show the number of training data points (*nData*) and the number of features (*nFeatures*), the number of iterations (*nIter*), training time, the training time divided by the number of training points, i.e. the training time per training data point, and a final objective function value. In addition, *nDecomp* columns in Table 2 are the number of decomposition iterations i.e. the number of times a working set is selected and optimization subproblem solved. Each simulation scenario was averaged over 10 runs.

Figures 1-2 show the ratios of times for the IPM and AL-FPGM. The values larger than 1.0, shown in blue indicate that the AL-FPGM was faster, i.e. it took more time of the IPM to train an SVM for a particular data set, while the values less than 1.0 shown in yellow indicate that the IPM was faster. Figures 1-2 suggest that the AL-FPGM outperforms the QuadProg IPM for the most training sets with and without the *pWSS* decomposition scheme. The IPM can train the SVM for no fewer than 5 iterations, with each iteration solving a dense linear system of equations. Assuming that Cholesky factorization of the solved system is used, we could roughly estimate that it takes about $\frac{1}{3}m^3 * 5 = \frac{5}{3}m^3$. Each iteration of FPGM can be done in about $2m^2$ flops as the most computationally expensive part is a matrix vector multiplication to form the gradient of the augmented Lagrangian. Therefore, the total number of flops is about $2km^2$, where k is the total number of AL-FPGM iterations. Solving the inequality $2km^2 \leq \frac{5}{3}m^3$ for k yields $k \leq \frac{5}{6}m$, which took place for all the solved problems according to Table 1.Therefore all the solved SVM problems from the test set should be solved faster by AL-FPGM, which is confirmed in Figure 1.

https://www.mathworks.com/help/optim/ug/quadprog.html

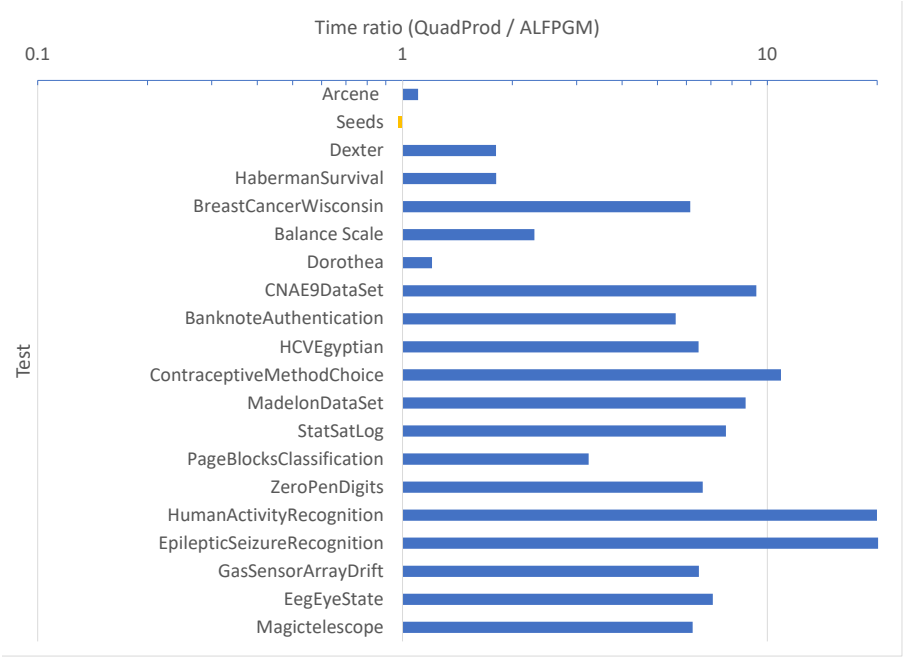

FIGURE 1. Comparison of IPM and AL-FPGM without decomposition. The blue bars correspond to the sets with a faster training by AL-FPGM

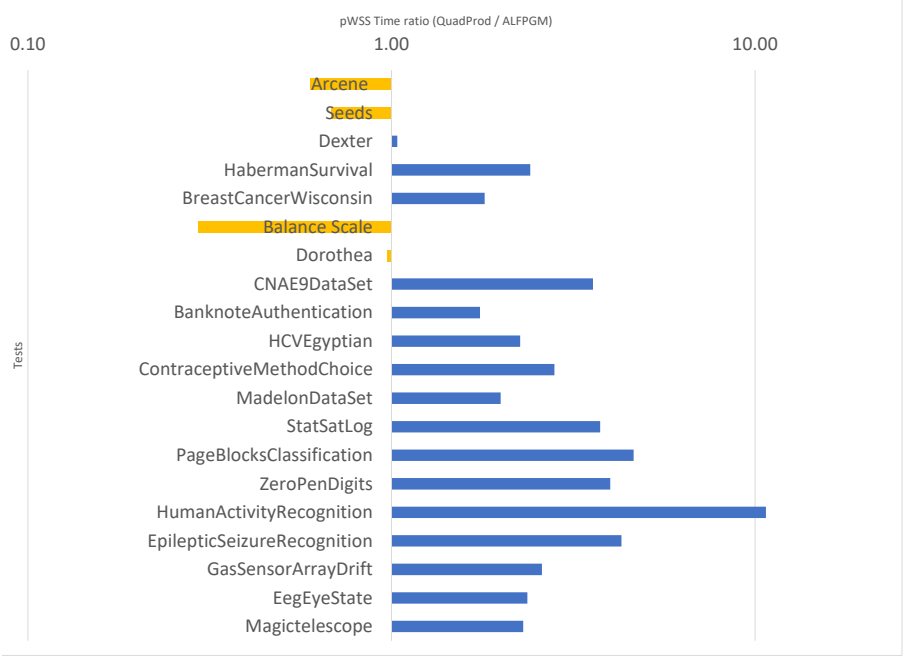

FIGURE 2. Comparison of IPM and AL-FPGM with decomposition pWSS. The blue bars correspond to the sets with a faster training by AL-FPGM

Figure 2 shows that the AL-FPGM is faster than the IPM for the largest problems when both use *pWSS* decomposition scheme.

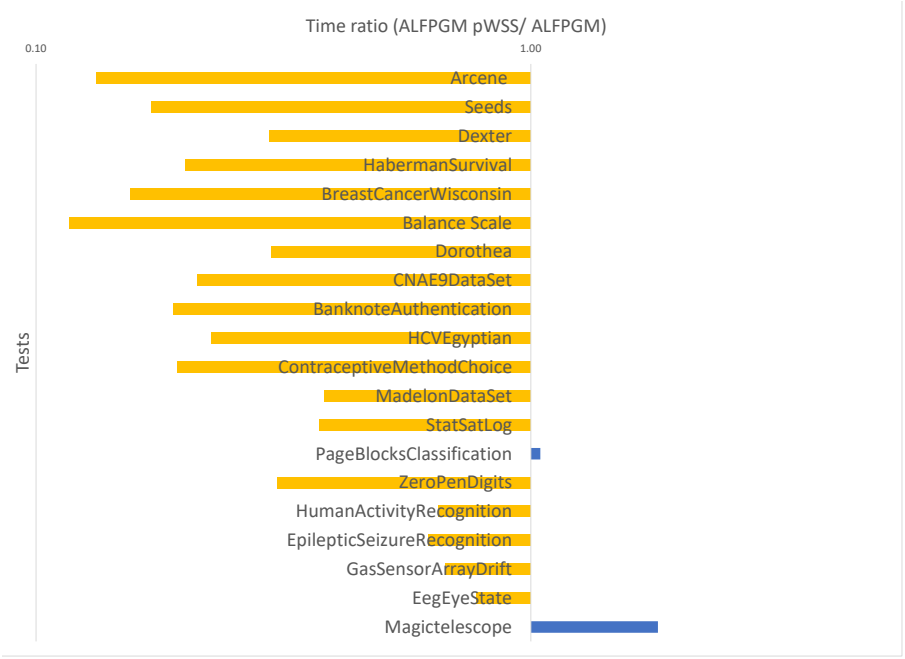

FIGURE 3. Comparison of no decomposition and pWSS decomposition schemes for AL-FPGM. The blue bars correspond to the sets with a faster training with pWSS scheme.

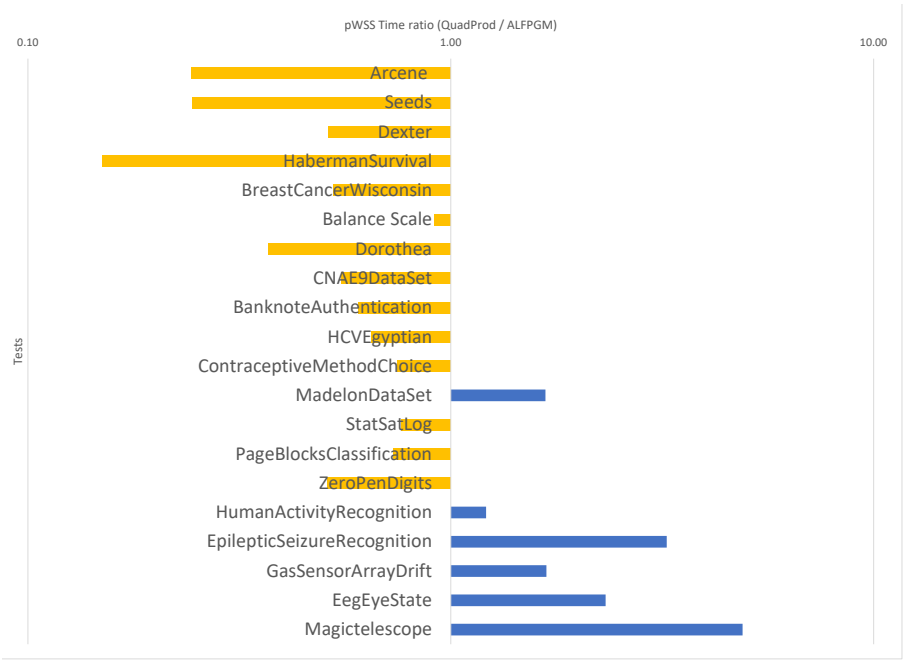

FIGURE 4. Comparison of no decomposition and pWSS decomposition schemes for IPM. The blue bars correspond to the sets with a faster training with pWSS scheme.

Figure 3 shows the relative efficiency of the *pWSS* decomposition scheme over no decomposition for the AL-FPGM. Similarly, Figure 4 shows that the relative efficiency of the *pWSS*

decomposition scheme over no decomposition for the IPM. The figures suggest that the relative efficiency of the *pWSS* decomposition scheme over no decomposition scheme grows with the size of the training data for both methods even on a stand alone desktop. The figures also suggest that for training data sets within a dozen of thousands points using using no decomposition method is justified, while for larger problems using decomposition methods is preferable.

Figure 5 shows the logarithmic plot of the dependence of the training times from the size of the training set for both AL-FPGM and IPM with and without decomposition. The training times are taken from Tables 1-2.

Due to non-homogeneity of the data sets (variability of features, scale, etc.), scalability trends are not clearly seen in Figure 5 due to zigzagging lines. Therefore we ran an additional experiment by taking a portion of data sets from the largest two training data sets. Figures 6 and 7 show the dependence of training times on the number of training examples for the EegEyeState and Magictelescope problems respectively, where one can see a linear trend in the logarithmic plot, suggesting a polynomial scaling of the training times. The blue and orange lines that correspond to the *pWSS* decomposition scheme have the smaller slope than the grey and yellow lines that correspond to no decomposition approach. The difference in slopes indicates that the *pWSS* scheme based on both the AL-FPGM and IPM may scale up as a polynomial of a smaller degree than do those methods without the decomposition.

Finally, Figure 8 shows the training times for the 4 largest problems indicating that the AL-FPGM with or without *pWSS* decomposition is faster than the runs for the IPM.

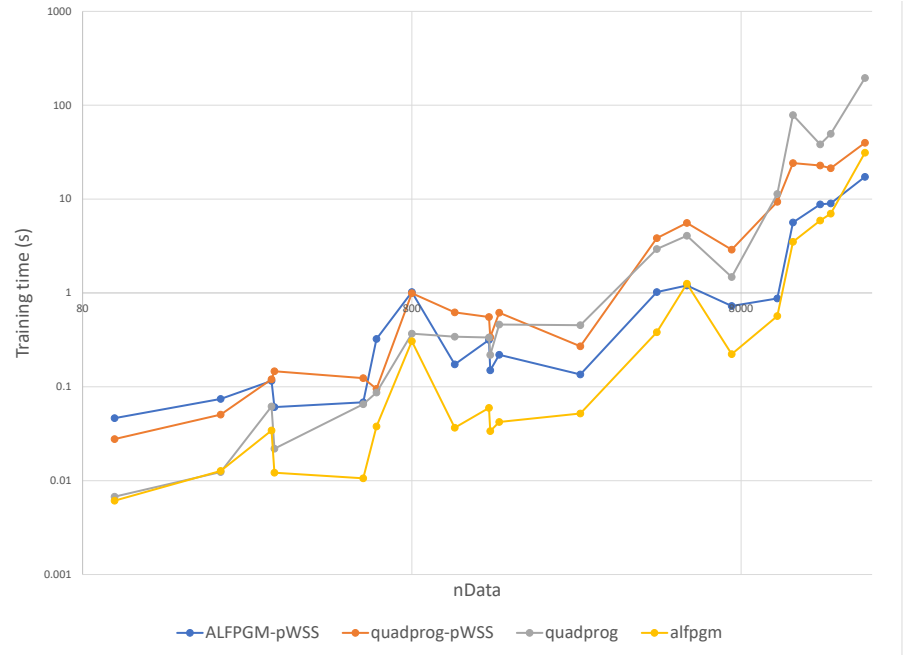

FIGURE 5. Dependence of SVM training times on the number of training points *nData* for AL-FPGM and IPM with and without *pWSS* decomposition (a logarithmic plot).

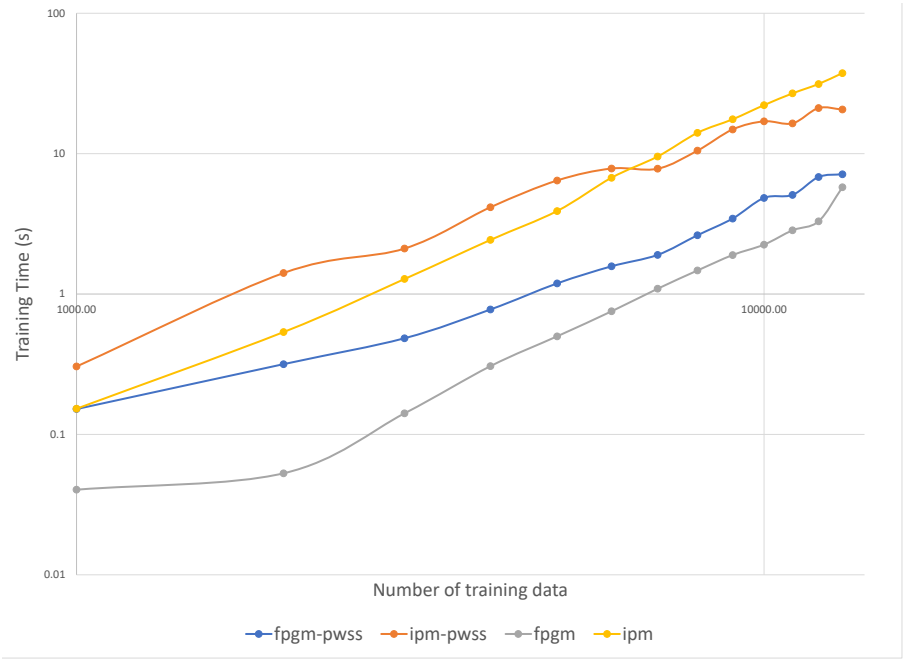

FIGURE 6. Training times for EegEyeStatetest example with varying number of training data subset.

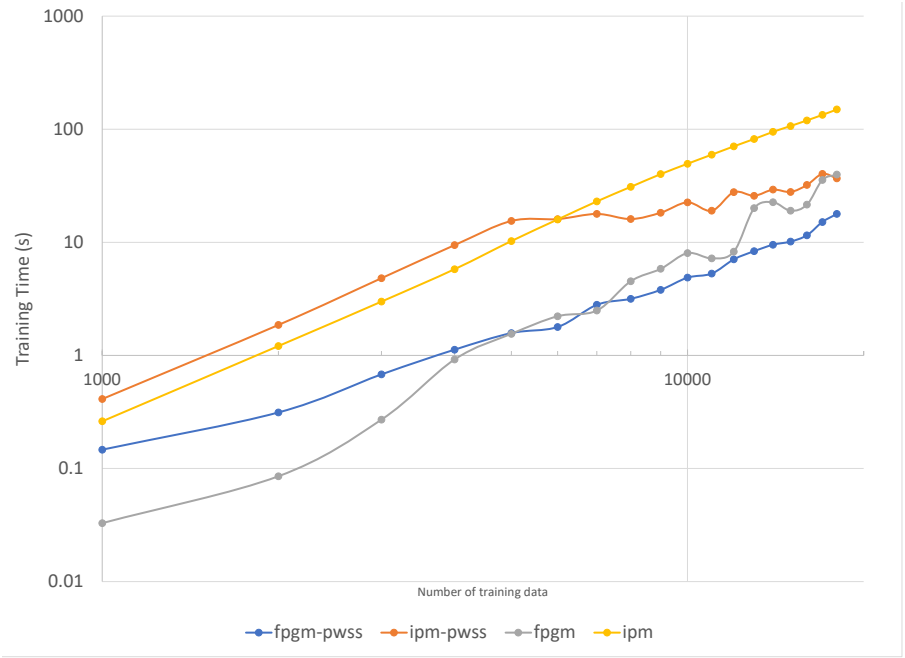

FIGURE 7. Training times for Magictelescope test example with varying number of training data subset.

5. CONCLUDING REMARKS AND FUTURE WORK

Support vector machines are popular for learning classification and regression rules on training data sets of modest size (up to a few thousands data points). Factors that contributed to the

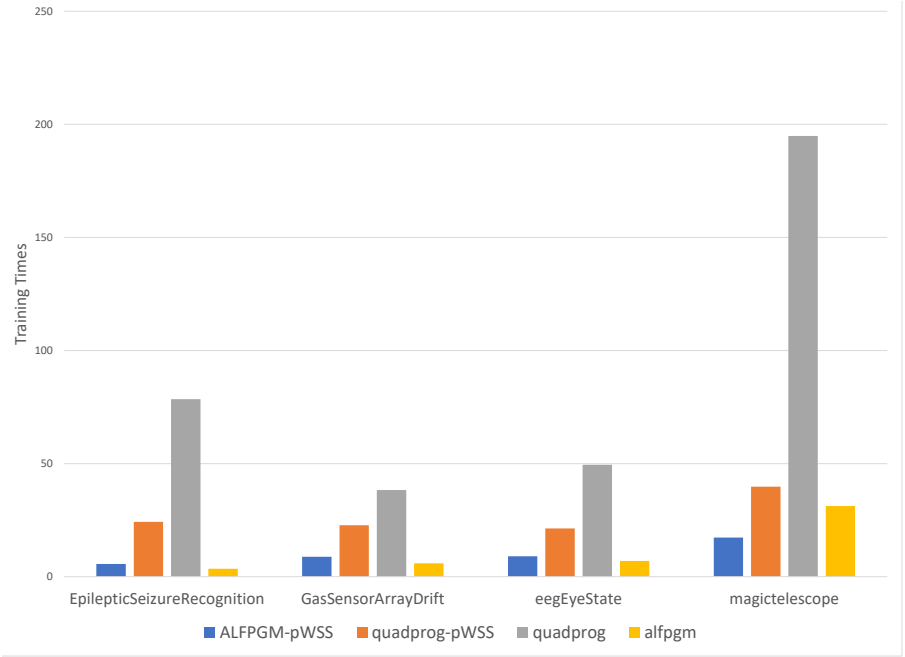

FIGURE 8. Training times for four largest training sets with AL-FPGM and IPM, with and without *pWSS* decomposition (a logarithmic plot).

popularity of suport vector machines have always been their strong generalization ability with solid theoretical bounds, and their interpretability of classification and regression performance using the similarity of new testing cases to existing training support vectors.

During the last decade we have seen a dramatic increase of data sets sizes that need to be processed for training machine learning algorithms. It became obvious that some widely used machine learning methods that simultaneously use all training data examples cannot cope with the need to process large training data sizes. Therefore the attention of the machine learning community has turned to the algorithm that use batches or subsets of training data.

Some machine learning methods such as artificial neural networks (ANN) have undergone a revolutionary development by shifting to a stochastic paradigm that allowed to train the ANN on batches, or parts of data sequentially. As the effect of training on earlier batches could fade when training later batches, ANN rely on going through cycles or epochs, each containing all the batches of the data set. As a result, there is no notion of a well defined stopping rule. At the same time, it is difficult to train ANN on all the batches in parallel, since it is going to be problematic to reconcile discrepancies among the neural networks trained on different batches.

Support vector machines, on the other hand, enjoy the presence of well defined optimization problem that needs to be solved to train an SVM. Therefore as long as we have an algorithm that can solve the SVM problem in a reasonable time, a good quality classification or regression rule is built. While it became apparent that such successful SVM training algorithms can be built only if it uses batches of data rather than whole training data set simultaneously, the information collected from training SVM on different batches can be efficiently brought together. Therefore training SVM on different batches can be done in parallel. Moreover, there is simple stopping rule that verifies if the training is finished or not.

We believe that this manuscript demonstrates how to turn the dream of training SVM on a large data points to the reality. Having a successful decomposition algorithm with each subproblem being solved efficiently allows using high performance (HPC) multi-node computers that take advantage of the parallel computing environment. Even though we have not demonstrate this point with HPC experiments in this manuscript, we plan to conduct such experiments in the future. The fact that the developed algorithm can train an SVM problem with more than 19,000 training examples for less than a half a minute even without HPC on a stand alone desktop demonstrates new opportunities for using support vector machines on large data sets.

REFERENCES

[1] V. Vapnik, Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics), Springer-Verlag, New York, 1982.

[2] V. N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.

[3] A. J. Smola, B. Schölkopf, A tutorial on support vector regression, Statist. Comput. 14 (2004), 199-222.

[4] H. Yu, S. Kim, SVM Tutorial — Classification, Regression and Ranking, In: G. Rozenberg, T. Bäck, J.N. Kok, (eds) Handbook of Natural Computing, Springer, Berlin, Heidelberg, 2012.

[5] T. Hastie, R. Tibshirani, Classification by pairwise coupling, Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10, ser. NIPS '97, pp. 507–513, 1998.

[6] L. Bottou, C. Lin, Support vector machine solvers, in Large Scale Kernel Machines, L. Bottou, O. Chapelle, D. DeCoste, J. Weston, (eds) pp. 301–320, MIT Press, Cambridge, MA, 2007.

[7] M. C. Ferris, T. S. Munson, Interior-point methods for massive support vector machines, SIAM J. Optim. 13 (2002), 783-804.

[8] I. Griva, R. A. Polyak, 1.5-q-superlinear convergence of an exterior-point method for constrained optimization, J. Global Optim. 40 (2008), 679-695.

[9] V. Bloom, I. Griva, B. Kwon, A. R. Wolff, Exterior-point method for support vector machines, IEEE Trans. Neural Networks Learn. Sys. 25 (2014), 1390–1393.

[10] R. Polyak, M. Teboulle, Nonlinear rescaling and proximal-like methods in convex optimization, Math. Program. 76 (1997), 265–284.

[11] S. Keerthi, E. Gilbert, Convergence of a generalized smo algorithm for svm classifier design, Mach. Learn. 46 (2002), 351–360.

[12] C. Campbell, Y. Ying, Learning with support vector machines, Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 5, pp. 1–95, 2011.

[13] C.-C. Chang, C. Lin, Libsvm: A library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011), 1–27. software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[14] W. Wei, C. Li, J. Guo, Improved parallel algorithms for sequential minimal optimization of classification problems, Proceedings - 20th International Conference on High Performance Computing and Communications, 16th International Conference on Smart City and 4th International Conference on Data Science and Systems, pp. 6-13, 2018

[15] Y. Nesterov, Introductory Lectures on Convex Optimization: a basic coursem Kluwer Academic Publishers, 2004.

[16] A. Beck, M. Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems, SIAM J. Imaging Sci. 2 (2009), 183–202.

[17] R. A. Polyak, J. Costa, S. Neyshabouri, Dual fast projected gradient method for quadratic programming, Optim. Lett. 7 (2013), 631–645.

[18] R. A. Polyak, Projected gradient method for non-negative least square, in infinite products of operators and their applications, A research workshop of the Israel Science Foundation, Haifa, Israel, 2012. Providence, RI: American Mathematical Society, pp. 167–179, 2015.

[19] I. Griva, Convergence analysis of augmented lagrangian - fast projected gradient method for convex quadratic problems, Pure Appl. Funct. Anal. 3 (2018), 417–428.

[20] V. Bloom, I. Griva, F. Quijada, Fast projected gradient method for support vector machines, Optim. Eng. 17 (2016), 651-662.
[21] R.-E. Fan, P.-H. Chen, C.-J. Lin, Working set selection using second order information for training support vector machines, J. Mach. Learn. Res. 6 (2005), 1889-1918.
[22] M. R. Hestenes, Multiplier and gradient methods, J. Optimization Theory Appl. 4 (1969), 303–320.
[23] M. J. D. Powell, Algorithms for nonlinear constraints that use lagrangian functions, Math. Program. 14 (1978), 224–248.
[24] P.-H. Chen, R.-E. Fan, C.-J. Lin, A study on smo-type decomposition methods for support vector machines, Trans. Neur. Netw. 17 (2006), 893908. https://doi.org/10.1109/TNN.2006.875973.
[25] Q. Zhang, J. Wang, A. Lu, S. Wang, J. Ma, An improved smo algorithm for financial credit risk assessment evidence from chinas banking, Neurocomputing, 272 (2018), 314–325.
[26] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines, Microsoft Corporation, Tech. Rep., April 1998. [Online]. Available: https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines/
[27] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, K. R. K. Murthy, Improvements to platt's smo algorithm for svm classifier design, Neural Comput. 13 (2001), 637-649.