Survey Paper: Distributed Classification based on ADMM

CSDS 433 Spring 2021

Randolph Zhao (yxz1648)

***Abstract*—As increasing attention has been paid to distributed classification algorithms, many researchers focus on the Alternating Direction Method of Multipliers (ADMM) since ADMM is easy to adapt to distributed calculation and easy to implement. In this survey paper, I analyze and compare six algorithms related to ADMM, including ADMM, CS-GADMM, SDCA-ADMM, SVRG-ADMM, A2DM2, D-A2DM2. I also propose several potential improvements that could be applied to these ADMM-related algorithms.**

Keywords—Distributed Classification, ADMM, CS-GADMM, SDCA-ADMM, SVRG-ADMM, A2DM2, D-A2DM2

I. Introduction

Nowadays, as the classification problem becomes more large scale with the increasing size of datasets, distributed classification problems become a hot topic in academic fields, which takes advantage of the computational power of multiple workers to solve the optimization problems. Many algorithms have been proposed for solving problems in the distribution method. For example, the Alternating Direction Method of Multipliers (ADMM) provides an optimization method that is flexible to adapt to distributed algorithms with different classifiers. Due to the slow convergence rate and high computation cost of ADMM, many researchers focus on optimizing the iteration update method to converge faster and also have lower calculation costs. In this paper, I analyze and compare these ADMM-related algorithms in a time sequence that provides improvement motivations and corresponding solutions.

II. Background

A. Alternating Direction Method of Multipliers (ADMM)

The Alternating Direction Method of Multipliers (ADMM) is a computation framework that provides an optimization method. The optimized calculation is easy to adapt to parallelized or distributed algorithms. The decomposition-coordination of ADMM divides a large global problem into several small and easy-to-calculate sub-problems and then coordinates the solutions of these sub-problems to get a global consensus as to the solution of the original function.

$$\min \sum_{i=1}^{N} f_i(x) + g(z) \tag{1}$$

$$x_i^{k+1} = \underset{x_i}{\operatorname{argmin}}\left(f_i(x_i) + \frac{\rho}{2}\left\|x_i - z^k + u_i^k\right\|_2^2\right) \tag{2.1}$$

$$z^{k+1} = \underset{x_i}{\operatorname{argmin}}\left(g(z) + \frac{N\rho}{2}\left\|z - x^{1-k} - u_i^{-k}\right\|_2^2\right) \tag{2.2}$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1} \tag{2.3}$$

ADMM is good at solving the problem as Eq. (1) by using its augmented Lagrangian to iteratively update variables as Eq. (2). It is obvious that variable x and u is easy to distribute calculate and only need a global update of z. In practice, it is better to use the reducer to update the penalty parameter ρ as Eq. (3) [1].

$$\rho^{k+1} := \begin{cases} \tau^{\text{incr}}\rho^k & \text{if } \|r^k\|_2 > \mu\|s^k\|_2 \\ \rho^k/\tau^{\text{decr}} & \text{if } \|s^k\|_2 > \mu\|r^k\|_2 \\ \rho^k & \text{otherwise,} \end{cases} \tag{3}$$

Since ADMM is a computation framework, it is easy to plug in different classifiers, such as Logistic Regression or SVM [1]. ADMM is also easy to implement and modify. However, the convergence rate of ADMM is proved to be $O(\frac{1}{\varepsilon})$, where ε is the precision threshold [5]. As $O(\frac{1}{\varepsilon})$ means it is a sublinear convergence rate, it is very slow when the precision requirement is high.

B. Cost-Sensitive classification algorithm via Group-based ADMM (CS-GADMM)

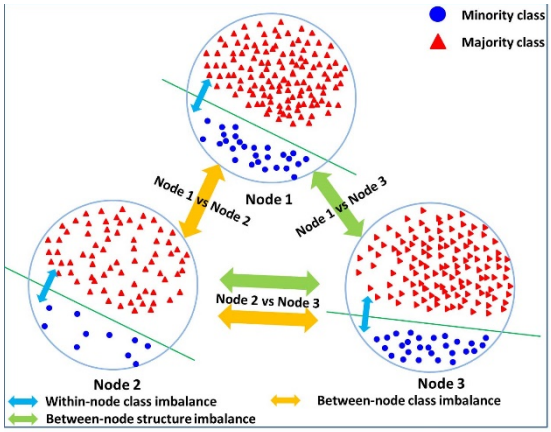

Fig. 1. Three class imbalance issues

One of the ADMM variants is the Cost-Sensitive classification algorithm via Group-based ADMM (CS-GADMM). The authors propose CS-GADMM (Algorithm 1) to address three imbalance issues: with-in node class imbalance, between-node class imbalance, and between-node structure imbalance (Fig. 1.) [2].

With-node class imbalance can result in poor performance for common classifiers and the authors introduce soft-margin cost-sensitive support vector machine (CSSVM) to address it. Between-node class imbalance can cause time delay since the training speed is limited by the slowest computing node. Inspired by Group-based GADMM (GADMM), the authors group workers with similar datasets and update intra-group to alleviate the between-node class imbalance. Between-node

structure imbalance will result in a difference in values of variables, which requires more iterations to reach a global consensus. To address this, the authors utilize a mini-batch extension of DCD (MDCD). Thus, CS-GADMM can adapt to imbalanced data and also have a faster convergence rate compared to ADMM.

Algorithm 1 Distributed classification via CS-GADMM

Input: datasets: $\mathbf{D} = \{D_1, D_2, \ldots, D_N\}$, parameters ($C_j$, ρ, α, ϵ^{pri}, ϵ^{dual}, $\mathbf{r}^0$, $\mathbf{s}^0$, $\mathbf{z}^0$ and $\mathbf{u}_i^0$)
Output: the global variable $\mathbf{z}^{k+1}$

$\beta_i = \frac{B_i(X)*N_i(X)}{\sum_{F_i \in G_j} B_i(X)*N_i(X)}$
$k \leftarrow 0$
while $\|\mathbf{r}^k\|_2 > \epsilon^{\text{pri}}$ or $\|\mathbf{s}^k\|_2 > \epsilon^{\text{dual}}$ **do**
  $\mathbf{w}_i^{k+\frac{1}{2}}$ can be obtained by MDCD in parallel
  if $k == 0$ **then**
    local functions are grouped into several groups
  end if
  $\mathbf{w}_i^{k+1} = \sum_{f_i \in G_j} \beta_i * \mathbf{w}_i^{k+\frac{1}{2}}$
  $\mathbf{z}^{k+1} = \frac{\rho}{1+N\rho} \sum_{i=1}^{N} (\mathbf{w}_i^{k+1} + \mathbf{u}_i^k)$
  $\mathbf{u}_i^{k+1} \leftarrow \mathbf{u}_i^k + \mathbf{w}_i^{k+1} - \mathbf{z}^{k+1}$
  $\mathbf{r}^{k+1} \leftarrow \mathbf{w}^{k+1} - \mathbf{z}^{k+1}, \quad \mathbf{s}^{k+1} = -\rho(\mathbf{z}^{k+1} - \mathbf{z}^k)$
  $k \leftarrow k + 1$
end while

C. Convergence Rate Problem

There are three kinds of convergence rate-sublinear, linear, and superlinear-and two kinds of representations-using k or using ε. Superlinear convergence has the fastest speed to converge to the required precision, while sublinear convergence is the lowest one. The precision threshold is the mechanism that tells the algorithm to stop, often represented by $\frac{1}{k}$ or ε. In this paper, I use ε for every convergence rate. For example, $O(\frac{1}{\varepsilon})$ is sublinear, $O(\frac{1}{\sqrt{\varepsilon}})$ is also sublinear but faster, and $O(\frac{1}{\log(\varepsilon)})$ is linear, which is the fastest among the three examples.

Since ADMM is proved to have $O(\frac{1}{\varepsilon})$ convergence rate, which is a sublinear convergence rate, many trails have been proposed to accelerate the convergence of ADMM. There are two major categories of methods: Variance Reduction (VR) and Nesterov's Acceleration (NA) [6]. In this paper, I introduce two representative algorithms for each category: SDCA-ADMM and SVRG-ADMM for variance reduction; A2DM2 and D-A2DM2 for Nesterov's Acceleration.

III. Improved Algorithms

A. Stochastic Dual Coordinate Ascent with Alternating Direction Method of Multipliers (SDCA-ADMM)

Stochastic Dual Coordinate Ascent with Alternating Direction Method of Multipliers (SDCA-ADMM) integrates the idea of SDCA in order to converge faster. SDCA-ADMM first uses Fenchel's duality theorem to get the dual problem of Eq. (1) and try to solve the dual problem instead. The reason is that the dual loss function becomes strongly convex in most cases. Then, for each iteration, SDCA-ADMM does not need to train on all data, but a mini-batch of data uniformly at random.

The original SDCA update of $x^{(t)}$ is given by Eq. (4). Since sub-batches have no overlap, we can construct a positive semi-definite matrix G as Eq. (5) to simplify the update rule of $x^{(t)}$ as Eq. (6). The update $y^{(t)}, w^{(t)}$ is showed in Algorithm 2 [3].

$$x^{(t)} = \underset{x_I}{\operatorname{argmin}}\{\textstyle\sum_{i\in I} f_i^*(x_i) - \langle w^{(t-1)}, Z_I x_I + B y^{(t)}\rangle + \frac{\rho}{2}\left\|Z_I x_I + Z_{\setminus I} x_{\setminus I}^{(t-1)} + B y^{(t)}\right\|^2 + \frac{1}{2}\left\|x_I - x_I^{(t-1)}\right\|_{G_{I,I}}^2\} \quad (4)$$

$$G_{I,I} = \rho(\eta_{Z,I} I_{|I|} - Z_I^T Z_I) \quad (5)$$

$$x^{(t)} = prox(x_I^{(t-1)} + \frac{Z_I^T}{\rho\eta_{Z,I}}\{w^{(t-1)} - \rho(Z x^{(t-1)} + B y^{(t)})\} | \frac{\sum_{i\in I} f_i^*(x_i)}{\rho\eta_{Z,I}}) \quad (6)$$

SDCA-ADMM has been proved to have a linear convergence rate $O\left(\frac{1}{Klog(\frac{n}{\varepsilon})}\right)$ under strong convexity in some special cases, and close to linear convergence rate under strong convexity in general. The convergence rate for general convexity remains unknown.

Algorithm 2 SDCA-ADMM

Input: $\rho, \eta > 0$
Initialize $x_0 = \mathbf{0}$, $y_0 = \mathbf{0}$, $w_0 = \mathbf{0}$ and $\{I_1, \ldots, I_K\}$.
for $t = 1$ **to** T **do**
  Choose $k \in \{1, \ldots, K\}$ uniformly at random, set $I = I_k$, and observe the training samples $\{(x_i, y_i)\}_{i\in I}$.
  Set $q^{(t)} = y^{(t-1)} + \frac{B^\top}{\rho\eta_B}\{w^{(t-1)} - \rho(Z x^{(t-1)} + B y^{(t-1)})\}$.
  Update $y^{(t)} \leftarrow q^{(t)} - \text{prox}(q^{(t)} | n\psi(\rho\eta_B \cdot)/(\rho\eta_B))$
  Update $x_I^{(t)} \leftarrow \text{prox}\left(x_I^{(t-1)} + \frac{Z_I^\top}{\rho\eta_{Z,I}}\{w^{(t-1)} - \rho(Z x^{(t-1)} + B y^{(t)})\} \middle| \frac{\sum_{i\in I} f_i^*}{\rho\eta_{Z,I}}\right)$.
  Update $w^{(t)} \leftarrow w^{(t-1)} - \gamma\rho\{n(Z x^{(t)} + B y^{(t)}) - (n - n/K)(Z x^{(t-1)} + B y^{(t-1)})\}$.
end for
Output: $w^{(T)}$.

B. ADMM with the method of stochastic variance reduced gradient (SVRG-ADMM)

Due to the slow convergence rate of ADMM and the high space cost of stochastic ADMM with variance reduction (e.g. $O(n)$ of SDCA-ADMM), the authors propose ADMM with the method of stochastic variance reduced gradient (SVRG-ADMM) by integrating SVRG, which not only has the same convergence rate as other stochastic ADMM with variance reduction but also requires less space cost.

There are two assumptions made on SVRG-ADMM: each f_i is convex, continuously differentiable, and has Li-Lipschitz-continuous gradient; g is convex but can be nonsmooth [4].

$$\hat{\nabla} f(x_{t-1}) = \frac{1}{b}\textstyle\sum_{i_t \in I_t}\left(\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x})\right) + \tilde{z} \quad (7)$$

SVRG-ADMM also computes based on a mini-batch. However, as Algorithm 3 shows, different from normal descent methods, SVRG divides iterations into stages, where each stage with m iterations uses a constant approximated

gradient given by Eq. (7). Since $\hat{\nabla} f(x_{t-1})$ is unbiased and its variance goes to zero, it allows the use of constant step size. To ensure that the next iterate is close to the current iterate, SVRG-ADMM updates with two more terms $\frac{\rho}{2}\|Ax + By_t - c + u_{t-1}\|^2$ and $\frac{1}{2\eta}\|x - x_{t-1}\|_G^2$ in each iteration. In this case, although the calculation cost increases a little for each iteration, it does not require extra space to store variables. The space requirement of SVRG-ADMM has proved to be $O(1)$.

Due to the concept of variance reduction of SVRG, which uses approximated gradient as Eq. (7) as update term, the variance has a decreasing up limit, i.e. the variance reduces and goes to zero, which results in linear convergence rate $O(\frac{\log(\frac{1}{K})}{\text{mlog}(\frac{R(\tilde{x}_0,\tilde{y}_0)}{\varepsilon})})$ under strong convexity [4]. The authors also prove the convergence rate of SVRG-ADMM is sublinear $O(\frac{1}{\varepsilon})$ under general convexity.

Algorithm 3 SVRG-ADMM for strongly convex problems.

Input: $m, \eta, \rho > 0$.
initialize $\tilde{x}_0, \tilde{y}_0$ and $\tilde{u}_0 = -\frac{1}{\rho}(A^T)^\dagger \nabla f(\tilde{x}_0)$;
for $s = 1, 2, \ldots$ **do**
  $\tilde{x} = \tilde{x}_{s-1}$;
  $x_0 = \tilde{x}_{s-1}$; $y_0 = \tilde{y}_{s-1}$; $u_0 = \tilde{u}_{s-1}$;
  $\tilde{z} = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\tilde{x})$;
  for $t = 1, 2, \ldots, m$ **do**
    $y_t \leftarrow \arg\min_y g(y) + \frac{\rho}{2}\|Ax_{t-1} + By - c + u_{t-1}\|^2$;
    $x_t \leftarrow \arg\min_x \hat{\nabla} f(x_{t-1})^T x + \frac{\rho}{2}\|Ax + By_t - c + u_{t-1}\|^2 + \frac{\|x - x_{t-1}\|_G^2}{2\eta}$;
    $u_t \leftarrow u_{t-1} + Ax_t + By_t - c$;
  end for
  $\tilde{x}_s = \frac{1}{m}\sum_{t=1}^{m} x_t$; $\tilde{y}_s = \frac{1}{m}\sum_{t=1}^{m} y_t$; $\tilde{u}_s = -\frac{1}{\rho}(A^T)^\dagger \nabla f(\tilde{x}_s)$;
end for
Output: $\tilde{x}_s, \tilde{y}_s$;

C. *Accelerated Alternating Direction Method of Multipliers (A2DM2)*

Algorithm 4: Accelerated ADMM (A2DM2)

Require: $y_0 = \hat{y}_0 \in \mathbb{R}^{n_2}$, $\lambda_0 = \hat{\lambda}_0 \in \mathbb{R}^m$, $\tau > 0$, $a_0 = 1$

1. **for** $k = 0, 1, 2, 3, \cdots$ **do**
2. $x_k = \text{argmin}_x L(x, \hat{y}_k, \hat{\lambda}_k)$
3. $y_k = \text{argmin}_y L(x_k, y, \hat{\lambda}_k)$
4. $\lambda_k = \hat{\lambda}_k - \tau(Ax_k + By_k - c)$
5. $a_{k+1} = \frac{1+\sqrt{1+4a_k^2}}{2}$
6. $\hat{\lambda}_{k+1} = \lambda_k + \frac{a_k - 1}{a_{k+1}}(\lambda_k - \lambda_{k-1})$
7. $\hat{y}_{k+1} = \text{argmin}_y f_2(y) + \langle \hat{\lambda}_{k+1}, -By \rangle$
8. **end for**

Different from the two algorithms above, the Accelerated Alternating Direction Method of Multipliers (A2DM2) uses Nesterov's Acceleration instead. Nesterov's method maintains that it can alternate between gradient updates and proper extrapolation. So, it is not a descent method anymore and requires extra attention on its convergence.

In A2DM2, the authors assume that there is strong convexity between $f(x)$ and $g(z)$ in Eq. (1) [5]. By using Nesterov's method, A2DM2 extrapolates the update of λ in each iteration and further use the updated λ to update the variable y to guarantee the convergence as the step 5, 6, 7 of Algorithm 4. Although each iteration A2DM2 has to calculate the updates of three more terms, due to a faster convergence rate, the overall running time is reduced. The authors prove that A2DM2 has a better sublinear convergence rate $O(\frac{1}{\sqrt{\varepsilon}})$.

$$m_k \stackrel{\text{def}}{=} \frac{1}{\tau}\left\|\lambda_k - \hat{\lambda}_k\right\|^2 + \tau\|B(y_k - \hat{y}_k)\|^2 \tag{8}$$

The authors also introduce a variant of A2DM2 to address the issue of possible spiral movements around the optimal solution. The authors use the term Eq. (8) to check whether it is a good time to restart. The experiments show A2DM2+Restart sometimes has better performance than A2DM2.

D. *Distributed Accelerated Alternating Direction Method of Multipliers (D-A2DM2)*

Inspired by the fact that distributed ADMM algorithms decentralize a classification problem as a global consensus optimization problem with a series of sub-problems, the authors propose Distributed Accelerated Alternating Direction Method of Multipliers (D-A2DM2), which uses A2DM2 to optimize the global consensus and SDCA-ADMM to optimize the sub-problems [6].

As Algorithm 5 shows, Step 5,6 is the regular step for updating y and u in ADMM. A2DM2 uses Nesterov's method to extrapolate u and update global and dual variables twice in Steps 7, 8, and 9. Note that Step 8,9 is applied with a predictor-corrector step by Nesterov's method which is helpful for the local update. A further acceleration strategy to accelerate local update which uses the current result and the last local result to further correct local update is shown in Steps 2, 3, and 4.

It is proved that D-A2DM2 also has a better sublinear convergence rate $O(\frac{1}{\sqrt{\varepsilon}})$ due to the use of A2DM2 in global updates. However, since it uses SDCA in the sub-problems optimization, D-A2DM2 will have a better convergence rate and calculation cost than A2DM2.

Algorithm 5 D-A2DM2 with Accelerated Local Update

Input: datasets: $\{D_1, D_2, \ldots, D_N\}$, $x^0 = \hat{x}^1$, $a_1 = 1$, parameters (ρ, β, r^0, d^0), tolerances (ϵ^p, ϵ^d)
1: **while** $\|r^k\|_2 > \epsilon^p$ or $\|d^k\|_2 > \epsilon^d$ **do**
2: $x_i^{k+\frac{1}{2}} = \arg\min_{x_i} f_i(x_i) + \frac{\rho}{2}\|x_i - y^k + \hat{u}_i^k\|^2$
3: $x_i^{k+1} = x_i^{k+\frac{1}{2}} + \beta(x_i^{k+\frac{1}{2}} - \hat{x}_i^k)$
4: $u_i^{k+\frac{1}{2}} = \hat{u}_i^k + x_i^{k+1} - y^k$
5: $y^{k+1} = \arg\min_y g(y) + \frac{\rho}{2}\sum_{i=1}^{N}\|x_i^{k+1} - y + u_i^{k+\frac{1}{2}}\|^2$
6: $u_i^{k+1} = u_i^{k+\frac{1}{2}} + x_i^{k+1} - y^{k+1}$
7: $a_{k+1} = \frac{1+\sqrt{1+4a_k^2}}{2}$
8: $\hat{u}_i^{k+1} = u_i^{k+1} + \frac{a_k - 1}{a_{k+1}}(u_i^{k+1} - u_i^k)$
9: $\hat{x}_i^{k+1} = x_i^{k+1} + \frac{a_k - 1}{a_{k+1}}(x_i^{k+1} - x_i^k)$
10: $r^{k+1} = \sum_{i=1}^{N}(x_i^{k+1} - y^{k+1})$
11: $d^{k+1} = \rho(y^{k+1} - y^k)$
12: $k = k + 1$
13: **end while**
Output: The global variable y^{k+1}

IV. REVIEW

A. Conclusions

1. ADMM

[1] tests ADMM with logistic regression classifiers daily on rolling historical large-scale data. The primal objective function loss shows a sublinear convergence rate (logarithmic scale). The accuracy consistently improves as more iterations pass.

[1] successfully implements ADMM for MapReduce which allows a simpler workflow when moving modeling ideas from prototype to production-scale on Hadoop. It shows a workable example on ADMM with logistic regression classifier and gets a relatively good convergence result. However, [1] does not solve the problem that ADMM has a slow convergence rate and does not include a fault tolerance mechanism. The experiments show that 25 iterations run in 1 hour and 25 minutes.

2. CS-GADMM

Since CS-GADMM is designed specifically for imbalanced data, [2] tests on three binary imbalanced datasets. The experiments show CS-GADMM performs better performance: improve the classification performance of imbalanced data and save time in between-node imbalance issues.

[2] analyzes three kinds of imbalance issues in practice and provides corresponding solutions: CSSVM, GADMM, and MDCD. The experiments show that imbalanced issues are alleviated largely which improves accuracy and time efficiency. By using GADMM, CS-GADMM also guarantees better time complexity and a faster convergence rate. However, CS-GADMM still has a sublinear convergence rate and does not include a fault tolerance mechanism.

3. SDCA-ADMM

[3] tests SDCA-ADMM on both artificial and real data and also compare with other stochastic optimization methods such as RDA, OL-ADMM, etc. The experiments on artificial data show SDCA-ADMM converges linearly. The experiments on real data show SDCA-ADMM converges faster than other tested algorithms on two different datasets.

[3] integrates SDCA algorithms into ADMM, which updates primal and dual variables to reduce the required iterations to converge linearly. Since linear convergence rate is significantly better than sublinear convergence rate, SDCA-ADMM shows large advantages on running time complexity. However, SDCA-ADMM does not include a fault tolerance mechanism and requires $O(n)$ to store the past gradients and weights or dual variables.

4. SVRG-ADMM

[4] performs experiments on the generalized lasso model and compares it with other variance reduction methods such as SAG-ADMM and SDCA-ADMM. The results show although all three algorithms have comparable speeds, SVRG-ADMM requires much less storage. When there are a large number of outputs, the small requirement of space makes SVRG-ADMM becomes advantageous.

As [4] finds the larges space requirement of previous stochastic variance reduction optimization methods, SVRG-ADMM is proposed to address the problem. The results of experiments show the SVRG-ADMM successfully reduces the space requirement while keeping the linear convergence rate. However, SVRG-ADMM does not include a fault tolerance mechanism and does not improve the convergence rate on general convex problems.

5. A2DM2

[5] tests its performance on ranking algorithms on artificial data with standard Gaussian entries. The experiments show that A2DM2 converges faster than ADMM due to its better sublinear convergence rate. A2DM2+Restart has been observed to have the best performance. The experiments of comparison between ranking algorithms show that the standard is competitive with the state-of-the-art TopPush algorithm both in terms of ranking accuracy at the top and training efficiency.

[5] uses Nesterov's method to extrapolate variables to result in a faster convergence rate. The experiments with ranking algorithms show that A2DM2 has better accuracy and training efficiency in practice. However, A2DM2 does not include a fault tolerance mechanism and its convergence rate is still sublinear, which will be much slower than the linear convergence rate algorithm. Besides, A2DM2 assumes there is strong convexity first and thus does not apply to the general convex problem.

6. D-A2DM2

[6] tests on six large-scale binary classification datasets with L2-regularized squared hinge loss SVM model in C++. The results are compared with other combinations among ADMM, DCA, SDCA, Nesterov's method, and A2DM2. The results of comparisons show that D-A2DM2-COM, which combines A2DM2 and SDCA, has the best performance in accuracy and training efficiency. D-A2DM2 also is showed to simultaneously improve computational efficiency through faster convergence.

[6] integrates A2DM2 and SDCA to accelerate the computation of global and local updates correspondingly. A2DM2 part results in a faster sublinear convergence rate and SDCA reduces the calculation cost and further improves convergence rate. However, D-A2DM2 still does not include a fault tolerance mechanism and its convergence rate is still sublinear, which will be much slower than the linear convergence rate algorithm. Besides, similar to A2DM2, D-A2DM2 assumes there is strong convexity first and thus does not apply to the general convex problem.

B. Comparisons

I compare these six surveyed algorithms in their convergence rate (general convex and strong convex) and performance on imbalanced data in Table 1.

It is easy to find that the stochastic variance reduction methods SDCA-ADMM and SVRG-ADMM have linear convergence rate in strong convex and thus they have the fastest running speed since calculations costs in each iteration are negligible for a faster convergence rate, i.e. fewer iterations at all. The convergence rate of most of algorithms under general convex have not been well studied yet and these algorithms all focus on the strong convex situation.

Among these six algorithms, only CS-GADMM can have a good performance on imbalanced data due to its three parts of solutions. Although the other five algorithms could adapt to soft-margin cost-sensitive SVM to within-node solve class imbalance issue, other algorithms do not use the idea of GADMM and MDCD and thus cannot reduce the time delay on between-node imbalance issues.

Algorithms	**Performance Measure**		
	Convergence Rate (Strong Convex)	***Convergence Rate (General Convex)***	***Performance on Imbalanced Data***
ADMM	$O(\frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon})$	Normal
CS-GADMM	$O(\frac{1}{\varepsilon})$	$O(\frac{1}{\varepsilon})$	**Good**
SDCA-ADMM	$\boldsymbol{O\left(\frac{1}{Klog\left(\frac{n}{\varepsilon}\right)}\right)}$	Unknown	Normal
SVRG-ADMM	$\boldsymbol{O(\frac{\log(\frac{1}{K})}{m\log(\frac{R(\bar{x}_0,\bar{y}_0)}{\varepsilon})})}$	$O(\frac{1}{\varepsilon})$	Normal
A2DM2	$O(\frac{1}{\sqrt{\varepsilon}})$	Unknown	Normal
D-A2DM2	$O(\frac{1}{\sqrt{\varepsilon}})$	Unknown	Normal

TABLE I. ALGORITHMS COMPARISONS

C. Potential Improvements

a) Fault Tolerance Mechanism

When operating on a large scale of data, it is very important to have a fault tolerance mechanism to prevent sudden failures. And it is also easy to develop a fault tolerance mechanism for distributed algorithms. However, in surveyed six ADMM-related algorithms, there is no fault tolerance at all. Thus, I suggest including a fault tolerance to prevent the situation that one of the workers is failed and the overall calculation has to be restarted.

Before starting the calculation, workers are divided into three groups: calculation workers, idle workers, and the coordinator. Calculation workers are the workers that are mainly responsible for sub-problems calculations. If some calculation workers are failed, idle workers will be activated to continue on the computation of failed workers. The coordinator has copies of all data partitions. When getting a global consensus, the coordinator can store local variables as a "checking point". If some worker fails in an iteration, the coordinator can use the variables in the last checking point to restore the progress.

b) Reduce Communication Cost

Due to the setting of ADMM that a global consensus has to be achieved to update one of the variables each iteration, the communication cost is huge. Compared to computation cost, the communication cost is more expensive. Thus, it will be better to focus on reducing communication costs, even at the expense of computation complexity. I propose two methods to reduce the communication costs of ADMM-related algorithms.

As SVRG suggests, in each stage the global gradient could be close to a constant. Thus, I propose the local update method. The algorithm can locally update all variables for a decreasing constant number of iterations. Then, all workers communicate and get a consensus about the new global variable value and gradient. When variables become closer to the optimal, the chance to converge increases due to the decreasing number of local iterations.

c) Hierarchically Group-Based ADMM

It is inspired by Group-based ADMM (GADMM), where group workers and update consensus intra-group. Similarly, the algorithm could groups workers and hierarchically get a consensus. For example, if there are 100 workers and 15 iterations, The 15 iterations will divide into 5 stages. The first iteration of a stage is to communicate and get a consensus in every 5-worker group. The second iteration of a stage is to communicate and get a consensus in every 25-worker group. The third iteration of a stage is to communicate and get a consensus in every 100-worker group. In this way, it guarantees that a global consensus could be achieved at the end of a stage but less computation on getting a consensus due to intra-group updates.

REFERENCES

[1] Lubell-Doughtie, Peter & Sondag, Jon. (2013). *Practical distributed classification using the Alternating Direction Method of Multipliers algorithm*. IEEE International Conference on Big Data.

[2] Wang, Huihui & Gao, Yang & Shi, Yinghuan & Wang, Hao. (2016). *A Fast Distributed Classification Algorithm for Large-Scale Imbalanced Data*. IEEE 16th International Conference on Data Mining.

[3] Suzuki, Taiji. (2013). *Stochastic Dual Coordinate Ascent with Alternating Direction Multiplier Method*. 31st International Conference on Machine Learning, ICML 2014. 2.

[4] Zheng, Shuai & Kwok, James. (2016). *Fast-and-Light Stochastic ADMM*. Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16).

[5] Kadkhodaie Elyaderani, Mojtaba & Christakopoulou, Konstantina & Sanjabi, Maziar & Banerjee, Arindam. (2015). *Accelerated Alternating Direction Method of Multipliers*. The 21st ACM SIGKDD International Conference.

[6] Wang, Huihui & Meng, Shunmei & Qiao, Yiming & Zhang, Jing. (2019). Fast Classification Algorithms via Distributed Accelerated Alternating Direction Method of Multipliers. IEEE International Conference on Data Mining (ICDM).