Optimizing the Optimal Weighted Average: Efficient Distributed Sparse Classification

Fred Lu[1,2,*]✉, Ryan R. Curtin[1,*], Edward Raff[1,2], Francis Ferraro[2], and James Holt[3]

[1] Booz Allen Hamilton, Fred_Lu@bah.com, Curtin_Ryan@bah.com
[2] University of Maryland, Baltimore County
[3] Laboratory for Physical Sciences

Abstract. While distributed training is often viewed as a solution to optimizing linear models on increasingly large datasets, inter-machine communication costs of popular distributed approaches can dominate as data dimensionality increases. Recent work on non-interactive algorithms shows that approximate solutions for linear models can be obtained efficiently with only a single round of communication among machines. However, this approximation often degenerates as the number of machines increases. In this paper, building on the recent optimal weighted average method, we introduce a new technique, *ACOWA*, that allows an extra round of communication to achieve noticeably better approximation quality with minor runtime increases. Results show that for sparse distributed logistic regression, ACOWA obtains solutions that are more faithful to the empirical risk minimizer and attain substantially higher accuracy than other distributed algorithms. We also introduce isoefficiency analysis to distributed logistic regression and show that ACOWA maintains favorable scaling with respect to data size and processor count relative to prior distributed algorithms.

1 Introduction

Statistical and machine learning research trends have had one important underlying trend for the past few decades: practitioners want to train models on larger and larger datasets [22, 15]. Massive-scale datasets present significant computational issues, regardless of the complexity of the models being used. Even training linear models is a challenge when the datasets get large and high-dimensional. As an example, consider a logistic regression model, which may be penalized with either the L_1-regularizer for sparsity, the L_2-regularizer to prevent overfitting, or both (the 'elastic net' [47]). Given a dataset $\mathcal{X}$ with n points in d dimensions, and labels $\mathcal{Y}$ with value -1 or 1, we want to find

$$\hat{w} := \arg\min_{w} \mathcal{L}_w(\mathcal{X}) + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2 \tag{1}$$

[*] These authors contributed equally to this work

where $\mathcal{L}_w(\mathcal{X})$ is the logistic regression objective:

$$\mathcal{L}_w(\mathcal{X}) := \sum_{(x_i, y_i) \in (\mathcal{X}, \mathcal{Y})} \ln\left(1 + e^{-y_i w^\top x_i}\right) \tag{2}$$

Solving this problem on moderately-sized datasets is fast and easy [7], but on datasets with millions or more of samples or features (or both!), this is computationally challenging. Having observed this, we wish to accelerate the training of linear models on large-scale data. Here we will consider the logistic regression objective, but our approach can be easily adapted for more general models.

Importance of sparse linear modeling. Despite flourishing attention towards complex machine learning models, generalized linear models have real-world impact in both applied and scientific settings to this day. They have attractive properties for downstream inference and are very effective in high-dimensional problems with limited samples [42, 25]. Furthermore, research in linear models continues to improve our understanding of machine learning [20, 40]. As we will show, our work offers practical and theoretical scalability contributions toward distributed linear modeling.

Sparse linear models have particular value because they encompass methods which simultaneously perform feature selection and model fitting [16], including the Lasso and other L_1-regularized linear models [10]. There are relatively few methods which can do both in a statistically principled manner. Even in more complex models such as neural networks, recent work on sparsity leverages fundamental principles from sparse linear models [23]. Sparse models have significant advantages for explainability and computational efficiency.

Early singlethreaded attempts. We are far from the first to consider accelerating the training of logistic regression models. For smaller datasets, the problem has been intensively studied [16]. There is extra difficulty when considering the L_1 penalty ($\lambda_1 > 0$), as this causes $L_w(\mathcal{X})$ to be non-differentiable and thus simple gradient descent techniques cannot be applied. Instead, algorithms such as FISTA and FASTA [12], based on proximal gradient descent, are often used. Proximal Newton techniques, using coordinate descent to solve the inner step, are highly popular, with the *GLMNET* [10] and *newGLMNET* [41] algorithms offering fast convergence. *newGLMNET* is specifically tuned for expensive objective functions such as logistic regression, and through the LIBLINEAR library [9] has become likely the most widely-used solver in practice.

Multithreaded single-system approaches. As the number of cores on processors has increased, interest in single-system parallelism has also. In this vein, LIBLINEAR-MP [45] is a modified multi-core newGLMNET implementation. Hogwild [31] and the more recent SAUS [29] use lock-free parallelism to prevent conflicts during gradient updates.

Iterative distributed approaches. However, even with a multithreaded approach, very large datasets may be larger than the memory of a single system, and thus a distributed approach is required. The use of distributed algorithms to train logistic regression models has been studied extensively [13, 26, 44]. In typical approaches, the dataset $\mathcal{X}$ is partitioned across p nodes, and then the model

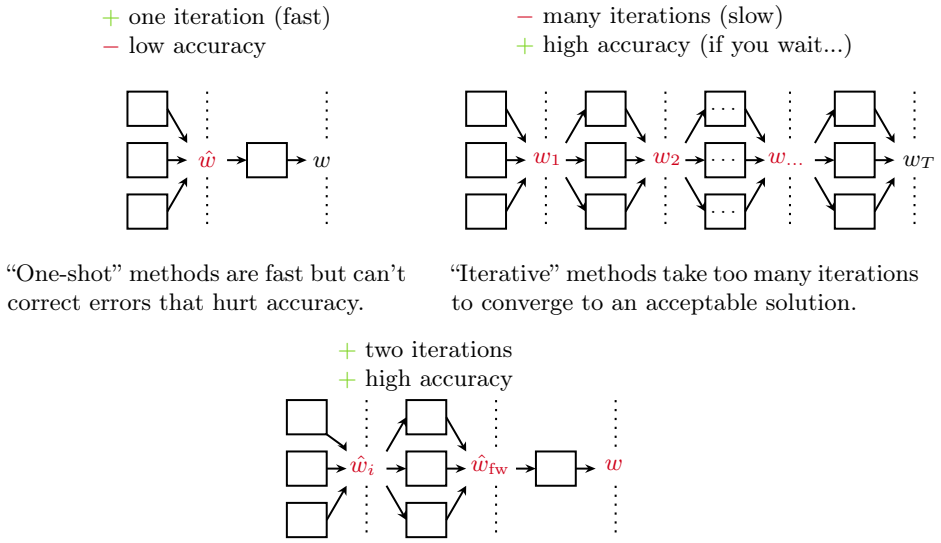


Fig. 1. Vertical dashed lines show synchronization points between threads and boxes indicate different compute nodes. Approaches for many-core and distributed training of models with an L_1 penalty are either one-shot (left), or iterative (right), neither of which produce satisfying solutions of high accuracy in a limited time frame. Our ACOWA strikes a careful balance of sharing information, such that a more accurate solution can be obtained with just two rounds of communication.

is learned iteratively. The simplest approach is to partition by data points; this partitioning strategy has been paired with distributed Newton methods [35, 44] and also ADMM [3]. Block coordinate descent methods which split over features have also been shown to work [38, 32]. A large disadvantage of these techniques is that they involve significant communication overhead: at every iteration, the gradients from each machine must be sent back to the main node. These communication costs become very painful with increased problem dimensionality.

A number of approaches have been developed to reduce communication costs. The CoCoA [19, 36] and ProxCoCoA+ [37] frameworks are two notable examples. ProxCoCoA+ uses the dual of the logistic regression objective function, partitioning the data by dimension instead of points. In each iteration, each worker solves a local quadratic approximation of the objective, communicating its solution back to the main node for aggregation. The DANE and CSL frameworks [35, 39] first find single-partition solutions independently, which are averaged as an initial estimator. Global gradients are then collected and combined with local higher-order derivatives on each partition to solve a surrogate likelihood function that has bounded loss with respect to the true likelihood.

Non-interactive algorithms. Still, even with efforts to reduce the amount of communication, the iterative nature of all the previous algorithms presents a problematic overhead when the number of iterations is large. Hence, there has recently been increased interest in *non-interactive* or *one-shot* algorithms [28, 43], which use only a single round of communication. These methods tend to be extremely fast, but produce models with a larger amount of approximation. The recent *optimal weighted average* (OWA) approach is a compelling example of a non-interactive algorithm [18]. In the OWA approach, data is partitioned along samples; each worker trains a model independently on its data partition and returns its trained model weights to the main node; then, the final model is a learned linear combination of each partition's model.

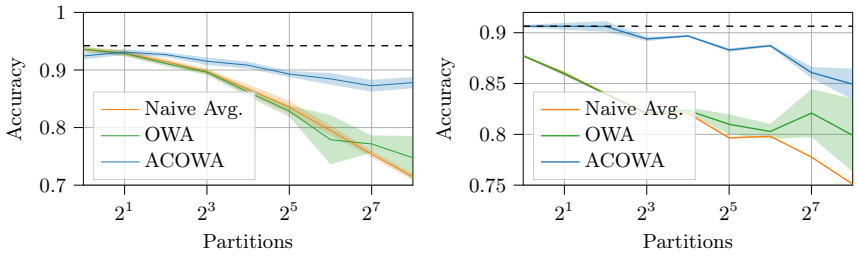
Our contribution: ACOWA. We have observed the compelling speedups of non-interactive algorithms, but found ourselves disappointed by the approximation quality—especially as the number of partitions p grows large, and as the dataset becomes sparse. Aiming to trade a small amount of speed for a much better approximation, we relax the one-shot requirement, and use two rounds of communication. Starting from OWA [18], we introduce ACOWA with a number of novel improvements:

1. We *augment each partition's data with summary information from other partitions*, reducing the variance of each partition's model.
2. We allow a *second round of weighted distributed learning*. This improves approximation quality by ensuring that ACOWA selects only features that have support across many partitions' models.
3. We introduce the study of *isoefficiency* (a measure of communication efficiency) to distributed logistic regression. We then show that ACOWA has isoefficiency comparable to the original OWA, and thus *retains its favorable scaling properties*.
4. Our experimental results demonstrate the *significant quality increases* that ACOWA yields, on a variety of datasets, with only a modest additional runtime cost. Our ablation studies show that all of our proposed improvements are mutually beneficial.

The differences between ACOWA and other approaches is shown in Figure 1. A publicly-available implementation of our algorithm can be found online at `https://github.com/FutureComputing4AI/Acowa`.

2 Problems with OWA

Before describing OWA in detail, we first consider the simplest one-shot approach: *naive averaging*. In both approaches, the dataset $(\mathcal{X}, \mathcal{Y})$ is split into p equal-sized partitions $(\mathcal{X}_i, \mathcal{Y}_i)$. For naive averaging, each worker i learns a model $\hat{w}_i$ independently on its partition, and then all models are collected on the main worker and averaged: $\hat{w}_{\text{na}} := (1/p) \sum_{i \in [p]} \hat{w}_i$. Naive averaging is trivial to implement, only involves one round of communication, and gives reasonable approximate solutions to the true $\hat{w}$. While naive averaging has been shown to



(a) *newsgroups* dataset (11k × 54k). (b) *ember-100k* dataset (600k × 100k).

Fig. 2. Accuracy on held-out test sets for different numbers of partitions p, when sparsity is fixed. The quality of the naive averaging and OWA models degrade significantly as p increases. ACOWA improves accuracy across all levels of $p > 1$.

be asymptotically optimal for nearly unbiased linear models, in high-dimensional models higher-order loss terms cause increasing error [33]. Specifically, naive aggregates show greater error as the number of partitions p grows, likely due to increasing bias of the subsampled estimators.

OWA [18] works similarly to naive averaging, but has an improved merge step that results in an optimal rate of decay for approximation and generalization error of $\mathcal{O}(\sqrt{d/n})$. To reduce the bias of $\hat{w}_{\text{na}}$, OWA instead learns a weighted linear combination of each $\hat{w}_i$: after each $\hat{w}_i$ is computed and returned to the main node, a small subsample $\mathcal{X}_C \subseteq \mathcal{X}$ and $\mathcal{Y}_C \subseteq \mathcal{Y}$ is computed. This sample can be as small as pn/d points, which in most settings is substantially smaller than n. Then, the optimal weighted average is defined as $\hat{w}_{\text{owa}} := \hat{W}\hat{v}$, where $\hat{W} := [\hat{w}_1, \ldots, \hat{w}_p]$. For logistic regression, $\hat{v} \in \mathbb{R}^p$ is the linear combination of models found by solving the optimization

$$\min_{v} \sum_{(x_i, y_i) \in (\mathcal{X}_c, \mathcal{Y}_c)} \ln\left(1 + e^{-y_i (\hat{W}v)^\top x_i}\right) + \lambda_{\text{cv}} \|v\|_2^2 \tag{3}$$

Here, the original penalty is replaced with an L_2 penalty for v. This term can be taken as a surrogate for the 'true' penalty term $\lambda_1 \|\hat{W}v\|_1 + \lambda_2 \|\hat{W}v\|_2^2$, and it is suggested that λ_{cv} be set by cross-validation. Because $\mathcal{X}_C$ and $\mathcal{Y}_C$ are small subsets, the cost of cross-validation is generally small as compared to the cost of training each $\hat{w}_i$. Adapting the OWA strategy to other statistical problems just involves reworking the original objective function to learn v instead of $\hat{W}v$.

While the OWA estimator $\hat{w}_{\text{owa}}$ tends to improve over the naive average $\hat{w}_{\text{na}}$, in practice the accuracy of the resulting model also degrades significantly when the number of partitions p becomes large. This may be due to the need for a subsample for the second optimization, which remains biased. In addition, the variance of the OWA estimator increases. Fig. 2 shows a representative example.

3 First Improvement: Centroid Augmentation

The increasing variance of OWA as p increases is a result of degradation in each $\hat{w}_i$: the smaller $\mathcal{X}_i$ is, the more likely $\hat{w}_i$ is to be further from the true $\hat{w}$. We now leverage theory on *coresets*: subsamples of data with bounded loss approximation error to the original dataset [27].

As a first line of reasoning, view $\mathcal{X}_i$ as a uniform subsample of $\mathcal{X}$: as $|\mathcal{X}_i|$ shrinks, $\mathcal{X}_i$ behaves as an ϵ-coreset whose bound on the relative error increases quadratically [7]. That is, let $L_w(\mathcal{X})$ be the full objective minimized in Eq. 1, evaluated over $\mathcal{X}$. Then for any w, we have that $|L_w(\mathcal{X}_i) - L_w(\mathcal{X})| \leq \epsilon \cdot L_w(\mathcal{X})$.

Secondly, consider that a coreset of a high-dimensional sparse dataset may contain features that are entirely zero-valued. These 'dead' features are then effectively ignored by any model on that coreset. In either viewpoint, $\mathcal{X}_i$ may not contain enough information to produce an accurate approximation of $\hat{w}$. Thus, consider a scheme where we augment $\mathcal{X}_i$ with centroids of other partitions:

$$\mathcal{X}_i^{(\text{aug})} := \mathcal{X}_i \cup \left(\bigcup_{j \in [p] \setminus \{i\}} \mu_j^+ \cup \mu_j^- \right) \tag{4}$$

with positive and negative centroids μ^+ and μ^- defined as

$$\mu_j^+ := \frac{1}{|\mathcal{X}_j^+|} \sum_{x_k \in \mathcal{X}_j^+} x_k, \quad \mu_j^- := \frac{1}{|\mathcal{X}_j^-|} \sum_{x_k \in \mathcal{X}_j^-} x_k \tag{5}$$

where $\mathcal{X}_j^+$ is the subset of $\mathcal{X}_j$ with positive labels in $\mathcal{Y}_j$, and correspondingly for $\mathcal{X}_j^-$. The weight of any point in $\mathcal{X}_i$ is taken as 1, and the weight of any μ_j^+ or μ_j^- is $|\mathcal{X}_j^+|$ or $|\mathcal{X}_j^-|$, respectively. (This is trivially adaptable to the multiclass case.)

The idea of centroid augmentation is theoretically justifiable; it can be shown that augmenting $\mathcal{X}_i$ with $2p$ centroids is guaranteed to produce a better approximation to $\hat{w}$ than increasing the size of $\mathcal{X}_i$ by sampling $2p$ additional points

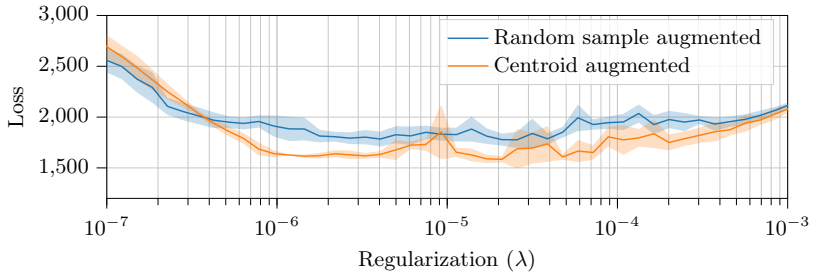


Fig. 3. Model loss values for *newsgroups* dataset with 128 partitions, augmented with centroids of other partitions vs. random samples. Lower loss is better. Centroid augmentation produces models with lower loss as regularization increases. (Too much regularization causes underfitting.)

from $\mathcal{X}$. See the supplementary material for details. To experimentally observe and confirm the theoretical result, we swept λ across a range of values for the newsgroups dataset, comparing the loss of the best models obtained when using partitions of the newsgroups dataset augmented with centroids, and augmented with random samples. The result is shown in Figure 3, confirming the result holds as regularization increases.

4 Second Improvement: Feature Weighting

When $\mathcal{X}$ is high-dimensional and sparse, not only do we have the problem of 'dead features' as previously discussed, but we also may have the situation where individual features are significantly over- or under-represented in any $\mathcal{X}_i$. This can also cause greater variance in the performance of OWA, as seen in Fig. 2. This phenomenon is amplified because L_1-regularized logistic regression (and elastic net) is not guaranteed to be consistent or possess the oracle property [46]. Even in low dimensions, L_1-regularized LASSO-type procedures are known to be inconsistent in variable selection [11, 24]. Thus, if an $\mathcal{X}_i$ over- or under-represents a feature of $\mathcal{X}$, the effects on variable selection can be even worse.

Zou [46] proposed a solution for the consistency of the simple linear regression Lasso estimator with the adaptive Lasso, which makes variable selection consistent by applying weights to each feature. This was then extended to the case of L_1-regularized logistic regression with the 'iterated Lasso' [17], where a first model is trained on the data, and then its weights are used to weight each feature for a second round of learning. The iterated Lasso is selection-consistent and possesses the oracle property under a few general assumptions on the data.

The strategy of the iterated Lasso is straightforward to adapt to the distributed case: we relax the one-round communication constraint and allow an additional round of feature-weighted learning, using weights from the first round of learning. Given first-round models $\hat{w}_i$, we can compute the percentage of models $\hat{w}_i$ that used a particular feature j: $P_j := (1/p) \sum_{i \in [p]} \mathbb{1}(\hat{w}_{ij} \neq 0)$.Then, we can define the weight for feature j as $\alpha_j := 1 + \beta P_j$ where β is a tunable parameter that controls the severity of the feature weighting. Then, separately in each partition, we solve an adaptive feature-weighted second round optimization:

$$\hat{w}^{\text{fw}} := \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \ell(y_i, x_i^\top w) + \lambda_1 \sum_{j \in [d]} \alpha_j^{-1} |w_j| + \lambda_2 \sum_{j \in [d]} (\alpha_j^{-1} w_j)^2. \quad (6)$$

Note that this problem is equivalent to scaling each dimension j of $\mathcal{X}$ by α_j with a rescaled penalty parameter $\lambda_{fw} := \lambda d / (\sum_{i \in [d]} \alpha_i^{-1})$. This use of a weighted second round is another improvement over standard OWA, and in our experiments, it improves stability by acting as a soft feature-selection step. This matches our expectation: the iterated Lasso can be understood as doing the same thing.

Algorithm 1 ACOWA.

Input: $\mathcal{X} \in \mathbb{R}^{n \times d}$, $\mathcal{Y} \in \{-1, 1\}^n$, regularization penalty λ, p processors, β
Output: learned model $\hat{w}_{\text{acowa}}$

split $(\mathcal{X}, \mathcal{Y})$ into p partitions and distribute to p workers
for $i \in [p]$ **in parallel do**
  compute μ_i^+ and μ_i^- using Eq. 5
end for
distribute all μ_i^+ and μ_i^- to all p workers
for $i \in [p]$ **in parallel do**
  learn $\hat{w}_i$ using local optimizer on $\mathcal{X}_i^{\text{aug}}$ (Eq. 4)
end for
compute α_j for all $j \in [d]$
for $i \in [p]$ **in parallel do**
  learn $\hat{w}_i^{\text{fw}}$ (Eq. 6) using local optimizer on $\mathcal{X}_i^{\text{aug}}$
end for
form $(\mathcal{X}_C, \mathcal{Y}_C)$ as a sample of size $\max(n/p, pn/d)$ from $(\mathcal{X}, \mathcal{Y})$
compute $\hat{w}_{\text{acowa}}$ as the solution to Eq. 3 with $\hat{W} = \{\hat{w}_1^{\text{fw}}, \ldots, \hat{w}_p^{\text{fw}}\}$

5 ACOWA

With these two major improvements over OWA, we can now introduce ACOWA, shown in Algorithm 1. ACOWA uses centroid augmentation, described earlier, for the first distributed round of learning, and then computes feature weights as described in the previous section for a second distributed round of learning. Then, the standard OWA merge step (Eq. 3) is applied. We highlight two additional improvements:

Larger merge set. OWA's merge step uses a dataset $\mathcal{X}_C$, of size pn/d. For high-dimensional problems, this set can be extremely small, causing high variance in $\hat{w}_{\text{owa}}$. Thus, it makes sense to increase the subsample size. Given that data is already distributed across partitions, we can simply use the main node's partition (with size n/p) as the second round. This significantly reduces the variance of $\hat{w}_{\text{owa}}$ with negligible runtime cost (see supplementary material).

Better optimizer. Our implementation (described further in the Experiments section) uses *newGLMNET* as provided by LIBLINEAR [9]. We found that relaxing the optimizer by reducing the number of inner coordinate descent iterations to 50 and the number of outer Newton iterations to 20 gives good speedup without loss in accuracy. Since the first round of learning can intuitively be understood as a 'soft' feature selection step, it is not necessary to run the first round of optimization to full convergence.

Extensions. In our exposition, for the sake of simplicity, we have specifically considered the regularized logistic regression problem, and our theoretical results have been restricted to that problem. However, ACOWA is a general algorithm, and as such it is straightforward to substitute any other type of linear model

(such as, e.g., the linear SVM). Theoretical results for centroid augmentation still apply, by adapting Lemma 2 in the supplementary material to the objective function of interest. Theoretical scaling results (in the next section) apply so long as the individual solvers for each partition scale similarly to *newGLMNET* (if not, the results can be adapted).

6 Scalability Analysis: Isoefficiency

Background. We next aim to characterize the *computational scalability* of ACOWA. Here, we introduce and discuss a classic parallel performance metric known as *isoefficiency* [14]. Isoefficiency measures how a distributed algorithm scales as the number of processors and the dataset size is increased, while taking into account communication costs, synchronization, and other overheads. To our knowledge, this is the first isoefficiency analysis of distributed logistic regression.

Although isoefficiency is not often studied in a machine learning context, we highlight that it is in fact well suited to analysis of distributed learning algorithms, as it is a principled way to quantify the marginal shrinkage of improvement as the number of processors p is increased. Intuitively, increasing p from 10 to 20 on a task should greatly improve runtime. But while further increasing p to 40 may still reduce runtime, it would likely be to a smaller extent than would be expected by simple mathematical analysis. This is because communication overhead rises with the number of partitions, which outweighs the speedup due to parallelism. A parallel algorithm which is more isoefficient than another will better utilize parallelism with fewer communication costs, and thus will scale better with the number of processors. This will be formalized in the following exposition. Due to space constraints, all proofs are in the supplementary materials.

Suppose we have a parallel computing system with p processors and dataset size z (to be defined later). We let $T_1(z)$ be the serial solve time of logistic regression and $T_p(z)$ be the solve time using p processors. Generally, increasing p causes a boost in relative speed $S_p(z) = T_1(z)/T_p(z)$ compared to serial, but also incurs increasing overhead $T_0(z)$, which is defined as $T_0 \coloneqq pT_p - T_1$. (We suppress the dependence on z when convenient.) We notice here that by definition, the overhead represents an excess cost invoked by the parallel algorithm compared to serial, which is not limited to work but can also include idle time. In an algorithm and system with no overhead, then $pT_p = T_1$, and $S_p = p$. In reality, for a given z, when p is increased linearly, S_p tends to grow sublinearly: if we define *efficiency* as $E_p(z) = S_p(z)/p$, then E_p decreases as p increases.

However, the more interesting and actionable quantity to a practitioner is the rate at which p must increase to retain the same efficiency E_p as the problem size z increases. This rate is formalized as the *isoefficiency function* and is the target of our analysis. The smaller this isoefficiency function, the more scalable the algorithm. For instance, an isoefficiency function of $z = \Theta(p)$ implies that when p is doubled, the overall runtime and efficiency can be maintained if z is also doubled. Such a fortunate situation cannot generally be expected, though;

a linear isoefficiency function implies an embarrassingly parallel problem with no communication overhead, which of course is not going to be the case for any distributed machine learning algorithm.

Computing the isoefficiency. The next step is to properly define the data size z. In our case, if we define z as the number of nonzero entries in $\mathcal{X}$, the serial newGLMNET solver runs linearly in z, as shown in Lemma 3 in the supplementary material. Conveniently, the linear scaling of the serial solver simplifies the next calculation.

Obtaining the isoefficiency function is equivalent to finding the function f describing $z \propto f(p)$ so that $E_p(z)$ is constant. Solving for E_p, we get

$$E = \frac{S}{p} = \frac{T_1}{pT_p} = \frac{T_1}{T_0 + T_1} = \frac{1}{1 + T_0/T_1} \propto \frac{1}{1 + T_0/z} \tag{7}$$

since $T_1 \propto z$. From this we see that for E_p to remain constant, we require that $z \propto T_0$. Further substituting in the definition of T_0, we see that the isoefficiency reduces to computing z as a function of p in the equation $z \propto pT_p - T_1$.

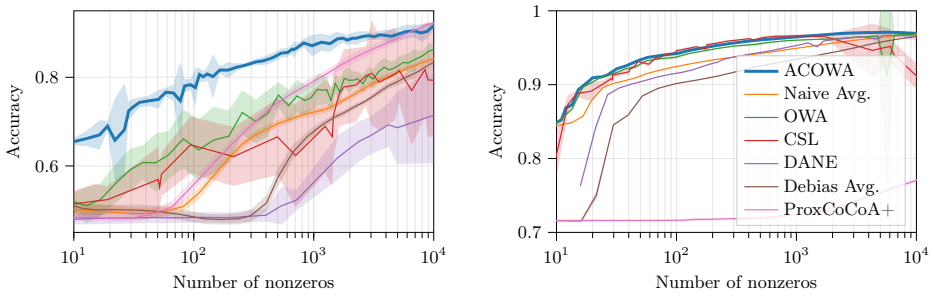
Results for distributed logistic regression. Because z can scale in various ways with the underlying dataset dimensions, the relative growth rates of samples n and features d can affect an algorithm's scaling. We keep in mind two regimes: (1) bounded d where $z \propto n$, and (2) $z \propto nd$. (1) implies we get more samples, while in (2) we get more samples and features, with the same underlying sparsity ratio. After distributed training, we define the support set S to be the union of all non-zero features across the partitions, and let $s = |S|$. Our first result concerns naive averaging.

Theorem 1. *Given distributed sparse logistic regression with sparsity level s, naive averaging has isoefficiency function $z = \Theta(sp \log p)$. If we suppose that (1) d is constant, or (2) $d(z) \to D$ for bounded D, then $z = \Theta(p \log p)$. Alternately, if n and d grow at the same rate such that $z \propto nd$, then the isoefficiency function is $z = \Theta(p^2 \log^2 p)$.*

Moving on to consider OWA, we find that the behavior not only depends on growth rates of the dataset, as in naive averaging, but also on certain parameters involved in the algorithm. In particular, the second round on the subsampled set of size n_c can pose a challenge to scalability if n_c grows on par with z, due to the overhead of solving the objective while the other processors are idle. In practice, n_c can be kept small as z grows without loss in accuracy.

Theorem 2. *Consider OWA with subsampled second round dataset $\mathcal{X}_C$ with n_c rows. If the growth rate of n_c is such that $n_c \propto z^\alpha$ for some $0 < \alpha < 1$, then OWA has isoefficiency function $z = \Theta(\max\{p^2, p^{2/(1-\alpha)}\})$. When $n \propto z$, we have $n_c \propto n^\alpha$. When $n \propto \sqrt{z}$, we have $n_c \propto n^{2\alpha}$.*

As a practical example, suppose a user observes a $5\times$ speedup running OWA with p processors on 10^4 data points compared to serial *newGLMNET*. Using $2p$ processors, they would need roughly 10^8 samples to maintain a $5\times$ speedup. We finally show that despite its additional round of computation, ACOWA has the same scalability as OWA.



(a) *newsgroups*, 256 partitions. (b) *amazon7*, 256 partitions.

Fig. 4. Number of nonzeros vs. test set accuracy in the single-node parallel setting. ACOWA has consistently better performance than other distributed methods, especially for sparser solutions on *newsgroups*. It generally also performs the best on *amazon7* across a range of sparsities, compared to the second best method (CSL).

Theorem 3. *Consider the ACOWA algorithm with subsampled second round dataset $\mathcal{X}_C$ with n_c rows. If the growth rate of n_c is such that $n_c \propto z^\alpha$ for some $0 < \alpha < 1$, then ACOWA has isoefficiency function $z = \Theta(\max\{p^2, p^{2/(1-\alpha)}\})$.*

7 Experiments

We conducted thorough experiments to compare the performance and runtime of ACOWA with competitive baselines: standard OWA [18], naive averaging, ProxCoCoA+ [37], CSL [21], and DANE [35].

We implemented ACOWA, OWA, naive averaging, and debiased averaging in C++ with OpenMP and MPI using the Armadillo linear algebra library [34], the ensmallen optimization library [5], and adapted parts of the mlpack machine learning library [6]. Our OWA implementation is tuned to allow a higher optimizer tolerance and a larger merge set size (like ACOWA). CSL and DANE were implemented similarly, using $\hat{w}_{\text{owa}}$ as the initial solution and OWL-QN [2] from libLBFGS (see `https://github.com/chokkan/liblbfgs`) as the per-iteration solver. To keep communication costs similar to OWA and ACOWA, we only ran one iteration of CSL and DANE. (We found further iterations did not improve the model significantly, and caused CSL and DANE to take much longer.) For ProxCoCoA+ we used the Scala implementation from the authors.

For datasets, we used several large-scale real-world datasets with sizes from approximately 10MB to 250GB. Our aim was to replicate a variety of real-world usage scenarios. All except EMBER [1] are available on the LIBSVM website [4] or UCI repository [8]. For EMBER, we compute length-8 n-grams [30] and keep the most common 100k and 1M to produce *ember-100k* and *ember-1M*.

We are interested in two settings: *(1)* single-node multicore, and *(2)* fully distributed. The first setting is relevant in modern environments, as modern

systems can have very many cores available. In our case, we used a powerful server with 256 cores and 4TB of RAM for our single-node experiments. Simple parallelized solvers such as the OpenMP version of LIBLINEAR struggle in this setting, as they were designed for only a few threads and cannot distribute large enough work chunks to very large numbers of cores. Applying distributed algorithms in this context is an effective strategy; the algorithms operate the same as in the fully distributed setting, but communication costs are lower as no network latency is incurred. For our fully distributed setting, we use a cluster with 16 nodes, with 32 cores and 1TB of RAM each.

Approximation Error. In the first set of experiments, we sweep over a logarithmic grid of λ_1 and compute the model's accuracy on a held-out test set, with λ_2 set to 0. We perform 10 trials with random partitions and random seeds. Because practitioners often try to tune sparse logistic regression to optimize accuracy at a certain level of sparsity, we plot the number of nonzeros in the solution versus accuracy. This gives us a good picture of how each algorithm behaves at different sparsity levels.

Fig. 4 shows the results of the sweep on smaller datasets in the single-node multi-core environment. ACOWA (blue) consistently attains higher accuracy, especially as the solution becomes more sparse (our setting of interest). This is also true in the fully distributed setting (Fig. 5). We found that CSL and DANE sometimes struggled to produce sparse solutions; for instance, on the EMBER datasets, the models produced by CSL and DANE only become competitive when they are dense (not our setting of interest).

ACOWA, due to the centroid augmentation and feature reweighting, is able to identify relevant features for the full model. This is especially true for sparser models, where other methods struggle due to the variance of models produced

Table 1. Runtime results for different techniques. *Fail* indicates the method took over two hours or had an out-of-memory issue. Although ACOWA takes longer to converge than naive averaging and OWA, it provides significantly better performance (see Fig. 4). This also generally holds when comparing with CSL and DANE.

(a) single-node parallel

dataset	n	d	**nnz**	ProxCoCoA+	Naive Avg.	OWA	ACOWA
newsgroups	11k	54k	1.5M	40.129s	0.226s	0.242s	2.154s
amazon7	1.3M	262k	133M	531.778s	2.356s	14.933s	31.675s
criteo	45M	1M	1.78B	*Fail*	17.085s	218.092s	264.200s
ember-100k	600k	100k	8.48B	*Fail*	7.811s	13.245s	80.814s

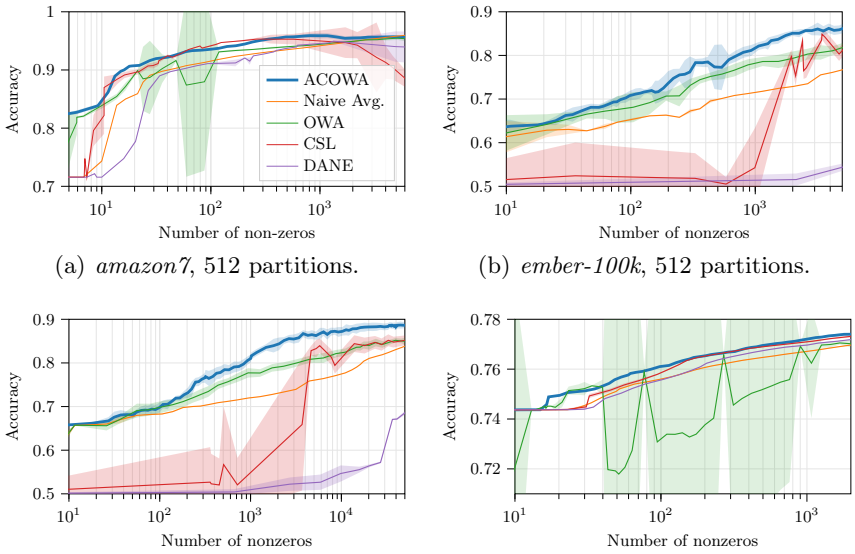
(b) fully distributed

dataset	n	d	**nnz**	CSL	DANE	Naive Avg.	OWA	ACOWA
ember-100k	600k	100k	8.48B	28.831s	40.318s	0.863s	1.129s	19.147s
ember-1M	600k	1M	38.0B	81.634s	68.975s	6.176s	5.836s	145.051s
criteo	45M	1M	1.78B	36.069s	65.618s	2.349s	25.885s	150.383s

by each partition. In the supplementary material, we perform several additional experiments: an ablation study shows that *both* centroid augmentation and feature reweighting are necessary for the improved approximation that ACOWA gives. Approximation results are similar for $\lambda_2 \neq 0$ (the elastic net); in addition, ACOWA is also robust to the choice of β.

Runtime. In the second set of experiments, we characterize ACOWA's runtime. We expect ACOWA to be slower than other one-shot algorithms, as we chose to increase the amount of communication modestly in exchange for significantly improving model performance. We tune λ_1 to produce approximately 1000 nonzeros in the final model, and take $\lambda_2 = 0$. We record the time taken to learn the model (excluding data loading and unrelated preprocessing).

Results are shown for each dataset in Table 1. These results match expectations: ACOWA is slower than the other one-shot algorithms, because it involves an additional round of communication, plus the initial communication of the centroids. ProxCoCoA+ is unable to complete within two hours for many datasets; we believe this to be a result of high communication overhead. In the distributed setting, the increased complexity of solving the surrogate loss function for CSL (and similar for DANE) causes slowdowns. As mentioned earlier, in our experiments we only use one iteration of CSL and DANE. Were we to run those to convergence, the runtime (and communication costs) would be much higher.



(a) *amazon7*, 512 partitions.

(b) *ember-100k*, 512 partitions.

(c) *ember-1M*, 512 partitions.

(d) *criteo*, 512 partitions.

Fig. 5. Number of nonzeros vs. test set accuracy in the multi-node distributed setting. ACOWA outperforms, again especially for sparser solutions. OWA on the criteo dataset exhibited significant variance. We were unable to run ProxCoCoA+ in this setting due to memory usage issues and extremely long runtimes.

Table 2. Runtime breakdown for ACOWA. Each step is associated with lines in Alg 1.

step	ember-100k	ember-1M	criteo
Centroids (4–5)	1.786s	6.724s	2.021s
All-to-all (6)	7.658s	58.268s	109.464s
Round 1 (7–8)	5.284s	45.376s	8.855s
Model gather	0.167s	1.540s	1.099s
Compute α_j (9)	0.134s	1.443s	1.319s
Round 2 (10–11)	4.281s	29.875s	7.068s
Model gather	0.173s	1.274s	0.940s
Round 3 (12–13)	0.310s	0.551s	19.617s
Total	19.793s	145.051s	150.383s

Runtime breakdown. Next we perform a detailed breakdown of the runtime cost of each step of ACOWA. Again tuning for 1000 nonzeros, we ran ACOWA in the fully distributed setting for 10 trials, collecting the average runtime of each step (specifically splitting out communication costs) in Table 2.

We can see that the communication rounds of ACOWA ('model gather') take a negligible fraction of the overall runtime, and because the communication being performed is only each (sparse) model $\hat{w}_i$, adding more data (but preserving the sparsity of the solution) does not affect the communication cost. Although the centroid computation and communication steps are computationally intensive, they do not scale with the dataset size (only with the number of partitions), and the computational and communication burdens of this step could be significantly alleviated by, e.g., the use of sparse centroids or other approximations. We plan to investigate this improvement in future work. As mentioned by Izbicki and Shelton [18], the last round of learning on the smaller set $(\mathcal{X}_C, \mathcal{Y}_C)$ takes a negligible amount of time compared to the rest of ACOWA.

Additional studies. Due to space constraints, we are unable to fit all of our experiments in the main paper; some are described in the appendix:

- *Ablation study.* Our results indicate that the combination of both centroid augmentation and a feature reweighted second round are mutually beneficial, and both improvements are necessary to provide the best accuracy.
- *General objective functions.* Our general approach can be readily adapted for other loss functions besides L_1 penalized logistic regression. We use ACOWA to solve Elastic Net logistic regression; ACOWA works successfully in this setting too, and can be further applied to any linear modeling problem.
- *Oracle solution.* We compare more thoroughly with the serial full-data solution provided by LIBLINEAR. We find that ACOWA can often come close to full-data performance, especially for highly sparse models.
- *Effect of β.* We investigate the effects of the parameter β, showing that ACOWA is robust to the choice of β.

8 Conclusion

We presented a minimally interactive method, ACOWA, for distributed logistic regression, which substantially improves on prior one- or few-shot distributed estimators. Our method scales to massive datasets, with better accuracy-to-sparsity ratio and similar runtimes than other methods, across multi-core and multi-node experiments, and has favorable theoretical justification.

References

1. Anderson, H.S., Roth, P.: EMBER: an open dataset for training static PE malware machine learning models. arXiv preprint arXiv:1804.04637 (2018)
2. Andrew, G., Gao, J.: Scalable training of L1-regularized log-linear models. In: 24th International Conference on Machine Learning. pp. 33–40 (2007)
3. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning **3**(1), 1–122 (2011)
4. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**(3), 1–27 (2011)
5. Curtin, R.R., Edel, M., Prabhu, R.G., Basak, S., Lou, Z., Sanderson, C.: The ensmallen library for flexible numerical optimization. JMLR **22**(1), 7552–7557 (2021)
6. Curtin, R.R., Edel, M., Shrit, O., Agrawal, S., Basak, S., Balamuta, J.J., Birmingham, R., Dutt, K., Eddelbuettel, D., Garg, R., et al.: mlpack 4: a fast, header-only C++ machine learning library. Journal of Open Source Software **8**(82) (2023)
7. Curtin, R.R., Im, S., Moseley, B., Pruhs, K., Samadian, A.: Unconditional coresets for regularized loss minimization. In: AISTATS 2020. pp. 482–492 (2020)
8. Dua, D., Graff, C.: UCI ML Repository (2017), `http://archive.ics.uci.edu/ml`
9. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. JMLR **9**, 1871–1874 (2008)
10. Friedman, J., Hastie, T., Tibshirani, R.: Regularization paths for generalized linear models via coordinate descent. Journal of statistical software **33**(1), 1 (2010)
11. Fu, W., Knight, K.: Asymptotics for lasso-type estimators. The Annals of Statistics **28**(5), 1356–1378 (2000)
12. Goldstein, T., Studer, C., Baraniuk, R.: A field guide to forward-backward splitting with a FASTA implementation. arXiv preprint arXiv:1411.3406 (2014)
13. Gopal, S., Yang, Y.: Distributed training of large-scale logistic models. In: ICML. pp. 289–297 (2013)
14. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE Parallel & Distributed Technology: Systems & Applications **1**(3), 12–21 (1993)
15. Halevy, A., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. IEEE intelligent systems **24**(2), 8–12 (2009)
16. Hastie, T., Tibshirani, R., Wainwright, M.: Statistical learning with sparsity: the lasso and generalizations (2015)
17. Huang, J., Ma, S., Zhang, C.H.: The iterated lasso for high-dimensional logistic regression. Univ. of Iowa, Dept. of Statistics and Actuarial Sciences **7** (2008)
18. Izbicki, M., Shelton, C.R.: Distributed learning of non-convex linear models with one round of communication. In: ECML PKDD 2019. pp. 197–212 (2020)

19. Jaggi, M., Smith, V., Takác, M., Terhorst, J., Krishnan, S., Hofmann, T., Jordan, M.I.: Communication-efficient distributed dual coordinate ascent. NIPS **27** (2014)
20. Jin, R., Li, D., Gao, J., Liu, Z., Chen, L., Zhou, Y.: Towards a better understanding of linear models for recommendation. In: KDD 2021. pp. 776–785 (2021)
21. Jordan, M.I., Lee, J.D., Yang, Y.: Communication-efficient distributed statistical inference. Journal of the American Statistical Association (2018)
22. Jordan, M.I., Mitchell, T.M.: Machine learning: Trends, perspectives, and prospects. Science **349**(6245), 255–260 (2015)
23. Kassani, P.H., Lu, F., Le Guen, Y., Belloy, M.E., He, Z.: Deep neural networks with controlled variable selection for the identification of putative causal genetic variants. Nature Machine Intelligence **4**(9), 761–771 (2022)
24. Leng, C., Lin, Y., Wahba, G.: A note on the lasso and related procedures in model selection. Statistica Sinica pp. 1273–1284 (2006)
25. Li, R., Chang, C., Justesen, J.M., Tanigawa, Y., Qian, J., Hastie, T., Rivas, M.A., Tibshirani, R.: Fast Lasso method for large-scale and ultrahigh-dimensional Cox model with applications to UK Biobank. Biostatistics **23**(2), 522–540 (2022)
26. Lin, C.Y., Tsai, C.H., Lee, C.P., Lin, C.J.: Large-scale logistic regression and linear support vector machines using Spark. In: BIGDATA 2014. pp. 519–528 (2014)
27. Lu, F., Raff, E., Holt, J.: A coreset learning reality check. In: AAAI. vol. 37, pp. 8940–8948 (2023)
28. Mcdonald, R., Mohri, M., Silberman, N., Walker, D., Mann, G.: Efficient large-scale distributed training of conditional maximum entropy models. NIPS **22** (2009)
29. Raff, E., Sylvester, J.: Linear models with many cores and CPUs: A stochastic atomic update scheme. In: BIGDATA. pp. 65–73 (2018)
30. Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., Tracy, A., McLean, M., Nicholas, C.: An investigation of byte n-gram features for malware classification. Journal of Computer Virology and Hacking Techniques **14**, 1–20 (2018)
31. Recht, B., Re, C., Wright, S., Niu, F.: Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. NIPS **24** (2011)
32. Richtárik, P., Takáč, M.: Distributed coordinate descent method for learning with big data. The Journal of Machine Learning Research **17**(1), 2657–2681 (2016)
33. Rosenblatt, J.D., Nadler, B.: On the optimality of averaging in distributed statistical learning. Information and Inference **5**(4), 379–404 (2016)
34. Sanderson, C., Curtin, R.: Armadillo: a template-based C++ library for linear algebra. Journal of Open Source Software **1**(2), 26 (2016)
35. Shamir, O., Srebro, N., Zhang, T.: Communication-efficient distributed optimization using an approximate newton-type method. In: ICML. pp. 1000–1008 (2014)
36. Smith, V., Forte, S., Chenxin, M., Takáč, M., Jordan, M.I., Jaggi, M.: CoCoA: A general framework for communication-efficient distributed optimization. JMLR **18**, 230 (2018)
37. Smith, V., Forte, S., Jordan, M.I., Jaggi, M.: L1-regularized distributed optimization: A communication-efficient primal-dual framework. arXiv preprint arXiv:1512.04011 (2015)
38. Trofimov, I., Genkin, A.: Distributed coordinate descent for l1-regularized logistic regression. In: AIST 2015. pp. 243–254 (2015)
39. Wang, J., Kolar, M., Srebro, N., Zhang, T.: Efficient distributed learning with sparsity. In: ICML. pp. 3636–3645 (2017)
40. Yang, S., Yuan, H., Zhang, X., Wang, M., Zhang, H., Wang, H.: Conversational dueling bandits in generalized linear models (2024)
41. Yuan, G.X., Ho, C.H., Lin, C.J.: An improved GLMnet for l1-regularized logistic regression. In: KDD 2011. pp. 33–41 (2011)

42. Zhang, X., Zhou, Y., Ma, Y., Chen, B.C., Zhang, L., Agarwal, D.: Glmix: Generalized linear mixed models for large-scale response prediction. In: KDD 2016. pp. 363–372 (2016)
43. Zhang, Y., Duchi, J., Wainwright, M.: Divide and conquer kernel ridge regression. In: COLT. pp. 592–617 (2013)
44. Zhuang, Y., Chin, W.S., Juan, Y.C., Lin, C.J.: Distributed newton methods for regularized logistic regression. In: PAKDD 2015. pp. 690–703 (2015)
45. Zhuang, Y., Juan, Y., Yuan, G.X., Lin, C.J.: Naive parallelization of coordinate descent methods and an application on multi-core l1-regularized classification. In: CIKM 2018. pp. 1103–1112 (2018)
46. Zou, H.: The adaptive lasso and its oracle properties. Journal of the American statistical association **101**(476), 1418–1429 (2006)
47. Zou, H., Hastie, T.: Regularization and variable selection via the elastic net. Journal of the Royal Stat. Soc. Series B: Statistical Methodology **67**(2), 301–320 (2005)