Distributed Confidence-Weighted Classification on MapReduce

Nemanja Djuric
Temple University
Philadelphia, PA, USA
nemanja.djuric@temple.edu

Mihajlo Grbovic
Yahoo! Labs
Sunnyvale, CA, USA
mihajlo@yahoo-inc.com

Slobodan Vucetic
Temple University
Philadelphia, PA, USA
vucetic@temple.edu

***Abstract*—Explosive growth in data size, data complexity, and data rates, triggered by emergence of high-throughput technologies such as remote sensing, crowd-sourcing, social networks, or computational advertising, in recent years has led to an increasing availability of data sets of unprecedented scales, with billions of high-dimensional data examples stored on hundreds of terabytes of memory. In order to make use of this large-scale data and extract useful knowledge, researchers in machine learning and data mining communities are faced with numerous challenges, since the classification algorithms designed for standard desktop computers are not capable of addressing these problems due to memory and time constraints. As a result, there exists an evident need for development of novel, more scalable algorithms that can handle large data sets. In this paper we propose such method, named AROW-MR, a linear SVM solver for efficient training of recently proposed confidence-weighted (CW) classifiers. Linear CW models maintain a Gaussian distribution over parameter vectors, thus allowing a user to estimate, in addition to separating hyperplane between two classes, parameter confidence as well. The proposed method employs MapReduce framework to train CW classifier in a distributed way, obtaining significant improvements in both training time and accuracy. This is achieved through training of local CW classifiers on each mapper, followed by optimally combining local classifiers on the reducer to obtain aggregated, more accurate CW linear model. We validated the proposed algorithm on synthetic data, and further showed that AROW-MR algorithm outperforms the baseline classifiers on an industrial, large-scale task of Ad Latency prediction, with nearly one billion examples.**

***Keywords*-confidence-weighted classification, MapReduce**

I. Introduction

Recent advent of high-throughput applications which generate data sets that can easily reach terabytes in size, such as remote sensing, crowd-sourcing, high-energy physics, social networks, or computational advertising, has brought forward a clear need for computational approaches that can efficiently learn from Big Data problems [1], [2], [3]. Emerging conferences that specifically address the Big Data issues, as well as the number of recent publications related to large-scale tasks, underline the significance of the Big Data field. Moreover, recently introduced ”Big Data Research and Development Initiative” by the United States government, aimed at providing support for these efforts, clearly indicates globally-recognized, strategic importance, as well as future potential and impact of Big Data-related research [4].

With the emergence of extremely large-scale data sets, researchers in machine learning and data mining communities are faced with numerous challenges related to the sheer size of the problems at hand, as many well-established classification and regression approaches were not designed and are not suitable for such memory- and time-intensive tasks. The inadequacy of standard machine learning tools in this new setting has led to investment of significant research efforts into the development of novel methods that can address such challenges. Classification tasks are of particular interest, as the problem of classifying input data examples into one of finite number of classes can be found in many areas of machine learning. However, state-of-the-art non-linear classification methods, such as Support Vector Machines (SVMs) [5], are not applicable to truly big data due to very high time and memory overhead, which are in general super-linear and linear in the data size N, respectively, significantly limiting their use when solving large-scale problems. Several methods have been proposed to make SVMs more scalable, ranging from algorithmic speed-ups [6], [7], [8], [9], [10], [11], to parallelization approaches [12], [13], [14]. However, scalability of SVM training is inherently limited as non-linear SVMs are characterized by linear growth of model size with training data size N [15]. This led to an increased interest in linear SVM models [16], [17], [18], [19], [20], which have constant memory and $\mathcal{O}(N)$ training time. These linear models provide a scalable alternative to non-linear SVMs, albeit with a certain drop in prediction accuracy.

Unfortunately, even linear time complexity may not be sufficiently efficient for modern data sets stored across petabytes of memory space, requiring researchers to develop and adopt new machine learning approaches in order to address extremely large-scale classification tasks. Significant research efforts culminated in several highly-influential frameworks for solving parallelizable problems that involve data sets which can not be loaded on a single machine. These frameworks for parallel computations include MapReduce [21], [22], AllReduce [23], GraphLab [24], [25], Pregel [26], and others. MapReduce in particular has become very popular framework in industry, with major companies such as Yahoo!, Google, and Facebook spearheading its use in large-scale commercial systems [27], [28].

Unlike other distributed frameworks that assume frequent communication and shared memory between the computation nodes (e.g., [23], [24]), MapReduce framework, and its open-source implementation called Hadoop, allows limited communication overhead between the nodes, which results in very strong fault-tolerance and guaranteed consistency. These favorable properties of MapReduce led to development of parallelizable variants of popular machine learning algorithms, such as k-means, perceptron, logistic regression, PCA, and others [29], [30], [31], [32]. However, the proposed classification methods mostly rely on iterative training and two-way communication between the computation nodes [31], [32]. This may impose significant costs during training as it does not closely follow the computational paradigm of MapReduce, which derives its reliability from the high level of autonomy of computation nodes.

In this paper we describe an efficient linear SVM learner with sub-linear training time, capable of fully employing the MapReduce framework to significantly speed up the training. The algorithm uses recently proposed Confidence-Weighted (CW) linear classifiers [33], [34] to train a number of SVM models on each of the mappers. Following completion of the map step, the local CW models are sent to the reducer that optimally combines local classifiers to obtain a single model, more accurate than any of the individual ones. Compared to the CW algorithms, the proposed method, named AROW-MapReduce (AROW-MR), allows significantly more efficient training of accurate SVMs on extremely large data sets due to the distributed training. We validate our approach on real-world, large-scale problem of Ad Latency prediction with nearly one billion data examples, where AROW-MR achieved higher accuracy and faster training than the baseline approaches.

The paper is organized as follows. In Section II we describe the Confidence-Weighted classifiers, shown to achieve state-of-the-art performance on a number of real-world applications. In Section III we overview popular frameworks for distributed learning, MapReduce and AllReduce. Then, in Section IV we describe the proposed algorithm, a distributed variant of CW classifiers which can be used to efficiently train highly accurate models on large-scale problems. In Section V we validate our approach on synthetic data set, and further show that the method outperforms the baseline approaches on real-world, industrial data set from the computational advertising domain. Lastly, we give the concluding remarks in Section VI.

II. Confidence-Weighted Classification

In this section we review the recently proposed confidence-weighted classifiers. We first detail the CW algorithm, proposed in [33], followed by the description of Adaptive Regularization of Weights (AROW) algorithm from [34], an improved CW method shown to significantly outperform the original CW algorithm.

First described in [33], the CW algorithm is a linear classifier that, in addition to the prediction margin for the new data example, also outputs probability of the correct classification. This is achieved by maintaining a multivariate Gaussian distribution over the separating hyperplanes, and during the training procedure both mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of the distribution are learned. In this way a more expressive and informative model is found, giving us information about noise in each of the individual features, as well as about the relationship between features.

Let us assume that a trained CW model, with known mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, is given. Then, for an example $(\mathbf{x}, y)$ from data set $\mathcal{D}$, described by feature vector $\mathbf{x}$ and binary label $y \in \{-1, 1\}$, this induces a Gaussian distribution over the prediction margin $\hat{y}$ as follows,

$$\hat{y} \sim \mathcal{N}\big(y(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x}), \mathbf{x}^{\mathrm{T}}\Sigma\mathbf{x}\big). \tag{1}$$

Following (1), we can compute the probability of correct classification by using the equation for the normal cumulative distribution function, to obtain the expression

$$\mathbb{P}(\mathrm{sign}(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x}) = y) = \frac{1}{2}\big(1 + \mathrm{erf}(\frac{y(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x})}{\sqrt{2\mathbf{x}^{\mathrm{T}}\Sigma\mathbf{x}}})\big). \tag{2}$$

The CW classifier is learned online, and the current model is updated each round after observing a training example. During training, our belief about the classifier before the t^{th} training iteration, expressed through the current mean $\boldsymbol{\mu}_t$ and the current covariance matrix $\boldsymbol{\Sigma}_t$, is updated so that the new example ($\mathbf{x}_t$, y_t) is correctly classified with probability larger than some user-set parameter η. In addition, we impose an additional constraint that our new belief before iteration $t+1$ is not too far from our belief at the previous iteration t. More formally, the stated requirements yield the following optimization problem,

$$\begin{aligned} &(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) = \underset{\boldsymbol{\mu}, \boldsymbol{\Sigma}}{\arg\min}\, D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)\big) \\ &\text{subject to } \ \mathbb{P}\big(y_t(\boldsymbol{\mu}^{\mathrm{T}}\mathbf{x}_t \geq 0)\big) \geq \eta, \end{aligned} \tag{3}$$

where D_{KL} is the Kullback-Leibler (KL) divergence. Since the problem (3) is non-convex, the authors of [33] solve an approximate convex problem, and derive closed-form updates for the parameters of the Gaussian distribution.

As formulated in (3), we seek such an update of the classification model so that the new training example is correctly classified with certain probability η. In [34], the authors point out that this may be suboptimal for noisy data sets. More specifically, once the learning algorithm observes a noisy example, the update would modify the current model so that the noise is correctly classified, which could have an adverse effect on the generalization performance of the classifier. To address this issue, a new CW formulation is proposed in [34], called Adaptive Regularization of Weights

(AROW). In this approach, the following problem is solved,

$$(\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}) = \underset{\boldsymbol{\mu}, \boldsymbol{\Sigma}}{\arg\min} D_{KL}\big(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \| \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)\big) + \lambda_1 \big(\max(0, 1 - y_t \boldsymbol{\mu}^{\mathrm{T}} \mathbf{x}_t)\big)^2 + \lambda_2 (\mathbf{x}_t^{\mathrm{T}} \Sigma \mathbf{x}_t). \tag{4}$$

We can see that at each training step the old and the new belief are still constrained to be close, as measured by the KL-divergence. However, unlike in the CW algorithm which aggresively updates the model in order to accomodate new examples, in AROW formulation the aggresiveness of maximization of margin and minimization of uncertainty for the new example are controlled by the regularization parameters λ_1 and λ_2, respectively. As shown in [34], after finding the derivative of the objective function with respect to the parameters, update equations for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in the case of misclassification of the t^{th} example (i.e., when $\text{sign}(\boldsymbol{\mu}_t^{\mathrm{T}} \mathbf{x}_t) \neq y_t$), can be written in closed-form,

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t + \alpha_t y_t \boldsymbol{\Sigma}_t \mathbf{x}_t, \\ \boldsymbol{\Sigma}_{t+1} &= \boldsymbol{\Sigma}_t - \beta_t \boldsymbol{\Sigma}_t \mathbf{x}_t \mathbf{x}_t^{\mathrm{T}} \boldsymbol{\Sigma}_t, \end{aligned} \tag{5}$$

where α_t and β_t are computed as

$$\begin{aligned} \alpha_t &= \beta_t \max(0, 1 - y_t \boldsymbol{\mu}^{\mathrm{T}} \mathbf{x}_t), \\ \beta_t &= (\mathbf{x}_t^{\mathrm{T}} \Sigma \mathbf{x}_t + r)^{-1}, \end{aligned} \tag{6}$$

and $r = 1/(2\lambda_1)$, for $\lambda_1 = \lambda_2$. Online AROW training is initiated with a zero-vector $\boldsymbol{\mu}_0$ and an identity matrix $\boldsymbol{\Sigma}_0$, and it further proceeds to observe training examples and to update the parameters using equations (5) and (6).

III. MapReduce framework

With the recent explosive growth of data set sizes, analysis and knowledge extraction from modern data sets using a single machine is becoming increasingly intractable. In particular, training time of popular classification and regression methods (e.g., SVM, classification trees) is at best linear in training set size, which may be too expensive for problems with billions of examples. To address this pressing issue, a number of frameworks for distributed learning on clusters of computation nodes has been introduced, offering different levels of parallelization, node independence, and reliability [21], [22], [23], [24], [25], [26]. In this section, we describe one such framework which has become very popular in industry, called MapReduce. We also discuss AllReduce distributed framework, which is utilized in Vowpall Wabbit, a state-of-the-art distributed machine learning package.

MapReduce framework [21], [22], implemented as an open-source platform Hadoop[1], consists of two distinct phases called *map* and *reduce*, which constitute one MapReduce *job*. In the map phase, mappers read parts of the dataset (possibly stored on multiple computers) and perform some action (e.g., filtering, grouping, counting, sorting) with final results being sent to reducer in a form of ordered (*key*, *value*) pairs. In the reduce phase, reducer performs a summary operation on the data received from the mappers, where the received data is sorted by their key values. There may be multiple mappers and reducers, and the framework guarantees that all values associated with a specific key will appear in one and only one reducer. Note that the limited communication between computation nodes in MapReduce framework, which is allowed only from mappers to reducers, ensures high independence of mappers and significant fault-tolerance of the framework. Even in the case of mapper failure, the entire job is not significantly affected as remaining mappers are not aware of the failure, which can be fixed with a simple restart of the failed node.

We illustrate MapReduce abstraction on a simple example. Given a dataset $\mathcal{D}$ with D features, we may want to find how many times each feature has a non-zero value, which can be achieved using several mappers and a single reducer. Each mapper reads a part of the dataset, and for each example outputs $(k, 1)$ if the k^{th} feature was non-zero. When the mappers finish outputing (*key*, *value*) pairs, the reducer starts reading these pairs sorted by their key. Then, on the reducer side, we initialize the count variable to 0, and add all values associated with the same key as the ordered pairs are received. Once a key that is different from the one associated with the current count is read, reducer outputs the total count and resets the variable to compute the non-zero count for the following feature. In this way there is no need to store the individual counts, which lowers memory cost of the reducer.

There are several ways of utilizing MapReduce for distributed learning: 1) read the data using multiple mappers, and learn the model on a reducer in an online learning manner; 2) maintain a global model that is used by all mappers to compute partial updates, which are aggregated on reducer to update the global model (requires running multiple MapReduce jobs for convergence); and 3) learn several local models on mappers, and combine them into a global one on a reducer. For the first option, distributed learning takes the same amount of time as learning on a single machine, with the benefit that there is no need to store the data on a single disk. The second option is typically used for batch learning [30], [31], [32], where each mapper computes partial gradient using the current model, while the reducer sums the partial gradients and updates the model. A new MapReduce job is then instantiated, with the updated model used by all mappers for the next round of gradient calculation. Thus, one job is analogous to one gradient descent step. Since learning may require several iterations to converge, multiple MapReduce jobs need to be ran one after another, which may be ineffective and time costly. In contrast, the third approach ensures more robust learning and small communication overhead as only a single job is run, and we utilize this approach to propose an efficient and accurate classifier in Section IV.

[1]http://hadoop.apache.org/, accessed June 2013

A. AllReduce framework

MapReduce abstraction allows very limited interaction between the computation nodes to ensure high fault-tolerance. In the following we introduce significantly less-constrained framework called AllReduce [23], which is utilized by the popular Vowpal Wabbit (VW) software package[2] [35]. Unlike MapReduce, AllReduce framework assumes communication between mappers as well, while the reducers are not used. In particular, when computing the update step for the current global model, partial update step computed on one mapper is communicated to all other mappers. Then, once every mapper receives a message from all other mappers, the aggregated update step is performed on all computation nodes simultaneously. A typical implementation of AllReduce is done by imposing a tree structure on the computation nodes, such that the partial messages are aggregated up the tree and then broadcasted down to all mappers.

Disadvantage of AllReduce framework is that the mappers need to run truly concurrently. However, it is common for large clusters to run many independent jobs requiring different amount of resources on the computation nodes, and for higher number of mappers there may be no guarantee that all of them will be available for concurrent execution. Furthermore, as we have observed in practice, due to the fact that all nodes are required to send their updates before the next learning iteration starts, AllReduce learning will stall if any individual mapper fails once the job has started.

IV. Confidence-Weighted Classification using MapReduce Framework

In this section we present a distributed AROW algorithm, which can be used to efficiently train very accurate linear classifiers on large-scale data. Let us assume that we have M mappers, and on each mapper an AROW model is trained using only a subset of the whole data set $\mathcal{D}$, by running the algorithm described in Section II. More specifically, on the m^{th} mapper an AROW model is trained using a data set $\mathcal{D}_m \subset \mathcal{D}$, such that $\bigcup_{m=1,\ldots,M} \mathcal{D}_m = \mathcal{D}$ and $\mathcal{D}_i \cap \mathcal{D}_j = \varnothing, i \neq j$. We denote the trained AROW parameters for the m^{th} mapper as $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$, which are sent to the reducer after the completion of the map stage. During the reduce stage, we learn the final, aggregated parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$ such that the multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ is an optimal combination of M multivariate Gaussian distributions learned on mappers.

More formally, let us define objective function $\mathcal{L}$ to be minimized on the reducer,

$$\mathcal{L} = \mathbb{E}_{\mathcal{N}(\boldsymbol{\mu},\boldsymbol{\Sigma})}[D_{KL}^S\big(\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)\|\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})\big)], \qquad (7)$$

where the expectation is taken over the distributions over hyperplanes that separate the data set $\mathcal{D}$, and D_{KL}^S is the symmetric KL-divergence, defined as

$$D_{KL}^S(A\|B) = \frac{1}{2}\big(D_{KL}(A\|B) + D_{KL}(B\|A)\big). \qquad (8)$$

As can be seen from (7), the reducer computes aggregated parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$ such that the expected symmetric KL-divergence between the aggregated Gaussian distribution and hyperplane distributions, drawn from the probability distribution over separating hyperplane distributions for the data set $\mathcal{D}$, is minimized. We note that the proposed method can be viewed as a generalization of the averaging CW model used for large-scale data sets, briefly discussed in [33]. Given the mapper-specific parameters $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m, m = 1, \ldots, M$, empirical estimate of the objective function $\mathcal{L}$ can be expressed as follows,

$$\mathcal{L} = \sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)\ D_{KL}^S\big(\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)\|\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big), \qquad (9)$$

where we define $\mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)$, or probability of the m^{th} distribution over the separating hyperplanes, as the fraction of the training set used to train the m^{th} AROW classifier. We refer to the final CW classification model as AROW-MR.

A. Reducer optimization of AROW-MR

The optimization function (9) is convex, thus there exists a unique set of $(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$ parameters that minimize $\mathcal{L}$. In this section we derive update equations for AROW-MR parameters, the mean and the covariance matrix of the aggregated Gaussian distribution over the separating hyperplanes.

In order to solve (9), we compute the first derivatives of the objecive function $\mathcal{L}$ with respect to the parameters of the aggregated Gaussian distribution. After finding the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\mu}_*$ and equating the resulting expression with 0, we obtain the following update rule for mean $\boldsymbol{\mu}_*$,

$$\begin{aligned}\boldsymbol{\mu}_* = &\Big(\sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)\ (\boldsymbol{\Sigma}_*^{-1} + \boldsymbol{\Sigma}_m^{-1})\Big)^{-1} \\ &\Big(\sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)\ (\boldsymbol{\Sigma}_*^{-1} + \boldsymbol{\Sigma}_m^{-1})\boldsymbol{\mu}_m\Big).\end{aligned} \qquad (10)$$

In order to compute the update rule for covariance matrix, we find the derivative of $\mathcal{L}$ with respect to $\boldsymbol{\Sigma}_*$ and equate the resulting equation with 0. After derivation, we obtain the following expression,

$$\begin{aligned}&\boldsymbol{\Sigma}_*\Big(\sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)\ \boldsymbol{\Sigma}_m^{-1}\Big)\boldsymbol{\Sigma}_* = \\ &\sum_{m=1}^{M} \mathbb{P}\big(\mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)\big)\ \big(\boldsymbol{\Sigma}_m + (\boldsymbol{\mu}_* - \boldsymbol{\mu}_m)(\boldsymbol{\mu}_* - \boldsymbol{\mu}_m)^{\mathrm{T}}\big).\end{aligned} \qquad (11)$$

Equation (11) is a Riccati equation of the form $\mathbf{XAX} = \mathbf{B}$, solved with respect to matrix $\mathbf{X}$ with matrices $\mathbf{A}$ and $\mathbf{B}$ given. After finding the decomposition of matrix $\mathbf{A}$ as

[2]https://github.com/JohnLangford/vowpal_wabbit, accessed June 2013

```
Algorithm 1 AROW-MapReduce (AROW-MR)
Inputs: Data set $\mathcal{D}$; number of mappers $M$
Output: Parameters of AROW-MR $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$

1. Map: Train the $m^{\text{th}}$ AROW classifier on subset $\mathcal{D}_m$ of the data set $\mathcal{D}$ using equations (5) and (6), one AROW classifier
        for each mapper, to obtain $\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_m$, where $m = 1, \ldots, M$
2. Reduce: Combine the local AROW classifiers into an aggregated AROW classifier using equations (10), (11) and (12)
3. Output aggregated AROW parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$
```

$\mathbf{A} = \mathbf{U}^{\mathrm{T}}\mathbf{U}$ (e.g., using the Cholesky decomposition), we can compute $\mathbf{X}$ in a closed-form using the following steps,

$$\begin{aligned}\mathbf{XAX} &= \mathbf{B}\\ \mathbf{XU}^{\mathrm{T}}\mathbf{UX} &= \mathbf{B}\\ \mathbf{UXU}^{\mathrm{T}}\mathbf{UXU}^{\mathrm{T}} &= \mathbf{UBU}^{\mathrm{T}}\\ (\mathbf{UXU}^{\mathrm{T}})^2 &= \mathbf{UBU}^{\mathrm{T}}\\ \mathbf{UXU}^{\mathrm{T}} &= \mathbf{U}^{0.5}\mathbf{B}^{0.5}(\mathbf{U}^{\mathrm{T}})^{0.5}\\ \mathbf{X} &= \mathbf{U}^{-0.5}\mathbf{B}^{0.5}(\mathbf{U}^{\mathrm{T}})^{-0.5}.\end{aligned} \tag{12}$$

By matching the elements of equation (12) with the elements of equation (11), we can find the closed-form solution for the covariance matrix $\boldsymbol{\Sigma}_*$. Then, in order to find the optimal parameters $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$, equations (10) and (11) are solved iteratively until convergence (we empirically found that only a few iterations are sufficient for the optimization procedure to converge). The pseudocode given in Algorithm 1 summarizes the steps of the AROW-MR algorithm.

Let us discuss the time complexity of AROW and its distributed version AROW-MR. Given a D-dimensional data set $\mathcal{D}$ of size N, complexity of AROW training amounts to $\mathcal{O}(ND^2)$. On the other hand, complexity of AROW-MR is $\mathcal{O}(ND^2/M + MD^2 + D^3)$, where the first term is due to local AROW training on mappers, and the second and the third term are due to reducer optimization, which involves summation over M matrices of size $D \times D$ and Cholesky decomposition of the result, respectively. For large-scale data sets, for which it holds $N \gg M$, we can conclude that AROW-MR offers efficient training with significantly lower time overhead when compared to AROW algorithm.

V. Experiments

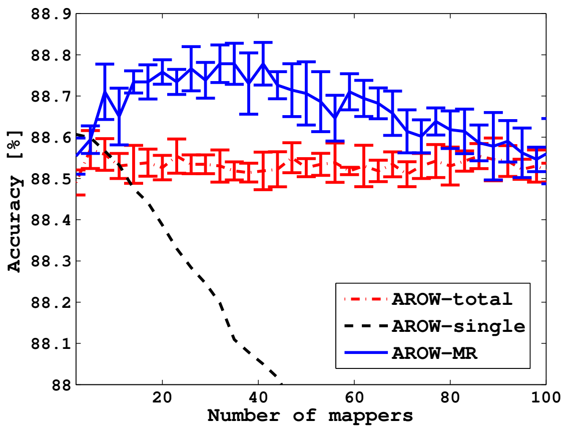
In this section we present the results of empirical evaluation of the AROW-MR algorithm. We first validated our method on a synthetic data set, then explored its performance on a real-world, large-scale task of Ad Latency prediction.

A. Validation on synthetic data

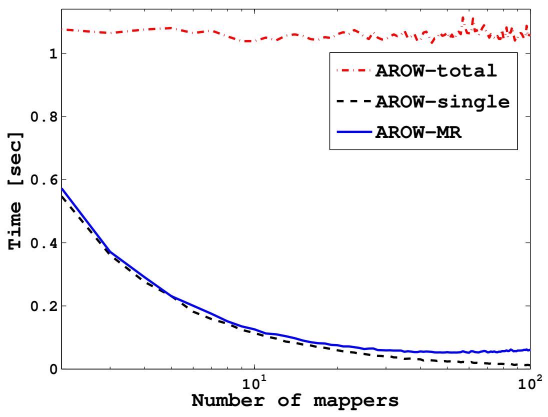
In order to better characterize the proposed distributed algorithm, in the first set of experiments we compared AROW and AROW-MR algorithms on synthetic data. We used the *waveform* data set generator, available from the UCI Repository, where we labeled the first and the second class as being positive and the third class as being negative. We generated $50{,}000$ training examples and $5{,}000$ test examples, and set $\lambda_1 = \lambda_2 = 0.1$ through cross-validation. We split the training set into M disjoint subsets of equal sizes and used one subset to train a local AROW on one mapper, where we increased the number of mappers M from 2 to 100 to evaluate the effect of higher levels of parallelization. We report the results of the original AROW which used the entire training data set (denoted by AROW-total, which was not affected by the number of mappers), the results of AROW-MR, as well as the results of a local AROW model trained on a single mapper, denoted by AROW-single, for which the number of training examples decreased as the number of mappers was increased (i.e., number of training examples for each local model was $50{,}000/M$). We included AROW-single results to illustrate the performance of local AROW models that are eventually combined on the reducer to obtain AROW-MR model. Experiments on synthetic data were run in Matlab, on MacBook Pro with 2GHz Intel Core i7 with 8GB of DDR3 memory.

Mean accuracy and training time after 10 repetitions are shown in Figures 1a and 1b, respectively, where the error bars in Figure 1a represent intervals of two standard deviations (we omitted error bars for AROW-single as the standard deviation was around 0.5 and error bars would clutter the figure). We can see in Figure 1a that the accuracy of AROW-MR initially increased as the number of mappers increased, statistically significantly outperforming AROW-total. The accuracy of local AROW models trained on each mapper (shown as AROW-single) dropped steadily with the increase of the number of mappers, which was expected as less training examples were used. Nevertheless, even though the accuracy of local models decreased, AROW-MR consistently outperformed AROW-total. Interestingly, as the number of mappers further increased, we can see that the accuracy of AROW-MR started decreasing when M surpassed 40, until it decreased to reach the accuracy of AROW for $M = 100$. This decrease is due to the fact that there are too few training examples on mappers for the local models to be close to convergence, which in turn affected accuracy of the aggregated model.

As illustrated in Figure 1b, distributed training also resulted in a significant speed-up in training time. We can see that AROW-MR training is order of magnitude faster than



(a) Classification accuracy



(b) Training time

Figure 1: Results on the synthetic *waveform* data set (with 50,000 training examples)

training of AROW-total, while at the same time achieving higher accuracy. Similarly to the accuracy results, the training time did not decrease further as we increased the number of mappers beyond a certain point. Although the mapper time continued to drop (shown by the dashed line), this was countered by longer time spent to solve the optimization problem (9) in the reduce phase due to larger M. This validates the known result that for certain problems "too much parallelization or distribution can have negative consequences" [36], and that the level of parallelization should be determined after deeper analysis of the problem being solved. Having said that, we can conclude that distributed training of AROW model resulted in significant accuracy and training time gains over the original AROW.

B. *Ad Latency problem description*

In the following section we compare the performance of AROW-MR and the baseline methods on large-scale, industrial task of Ad Latency prediction. However, before moving on to the discussion of empirical results, we first introduce this important problem in online advertising, as well as the large-scale data set used in the experiments.

Over the previous decade, income generated by internet companies through online advertising has been growing steadily at amazing rates, with the total revenue reaching a record $36.6 billion in the US in 2012 alone[3]. This burgeoning, highly competitive market consists of several key players: 1) advertisers, companies that want to advertise their products; 2) publishers, websites that host advertisements; and 3) intermediate players that connect advertisers to publishers. In some cases such clear segmentation is not possible, and certain companies can be found in more than one role (e.g., Yahoo! or Google may provide both the products and the advertising space). Typically, advertiser designs an image of the advertisement, called a creative, specifying size and dimension requirements of the image to be shown on websites. This is then sent to the intermediate companies which have contracts with publishers, and which decide when and to whom the ads will be shown in order to maximize profits.

In order to retain existing and attract new users, publishers aim at improving user experience by minimizing page load times. In addition, equally important task for publishers is to ensure that the ads are delivered on time. Considering that ad latency time accounts for a significant percentage of the overall load time, improving ad load times would directly benefit both the user experience and the website revenue. Thus, correctly predicting ad latency time, and using this information to decide which ad should be shown to a user, is an extremely important problem in online advertising where an additional latency of several milliseconds could result in a significant loss of revenue. In this paper, we considered Ad Latency dataset consisting of nearly 1.3 billon ad impressions, for which 21 features were measured at serve time, along with ad latency given in milliseconds. Features can be divided into several groups:

- User-specific features include user's device type, operating system, and browser. We also used user's geographic location (i.e., state and city), user's physical distance from the colocation center serving the ad, user's connection speed (e.g., broadband, dial-up), as well as internet service provider used by the user.
- Advertiser-specific features include the advertiser's account ID, size of the advertisement (2-D dimensions of the creative, and its size in kilobytes), as well as the creative ID of a specific image used by the advertiser.
- Publisher's website can be partitioned into several re-

[3]http://news.yahoo.com/us-internet-ad-revenue-grows-15-percent-2012-153752947.html, accessed June 2013

Table I: Features from the Ad Latency data set

Feature name	Cardinality
Device	15
Operating system	22
Browser	100
Connection speed	10
State	50
City	574
ISP	×
Distance to colocation center	continuous
Account ID	×
Creative ID	×
Ad size (dimensions)	33
Ad size (size in KB)	continuous
Region ID	×
Space ID (location on the page)	×
Ad position	28
Hostname	×
Ad network	×
Serve type	×
Colocation center	×
Hour of the day	24
Day of the week	7

gions, where each region has several spaces on which the ad can be shown. Further, ad can be placed at several different positions in the space (e.g., top, bottom). Thus, publisher-specific features include region ID, space ID, position, ad network used to serve the ad, serve type, hostname, as well as colocation center used to serve the ad.

- Lastly, we use time-stamp of ad impression, using time of day and day of the week.

In Table I we give the data set features, as well as the cardinalities for discrete features (we omitted sensitive information, which is marked with the '×' symbol).

We can represent the problem as a binary classification by thresholding the value of ad latency. An ad is considered late (i.e., $y = 1$) if the time period from the moment when the webpage loads to the moment when the ad renders is longer than k milliseconds, and not late otherwise (i.e., $y = -1$). Value of k can be selected depending on a product or ad campaign requirements, and we omit the specific value used in the experiments as it represents a sensitive information.

C. Validation on Ad Latency data

In order to evaluate performance of the classification algorithms, we randomly split the Ad Latency data set into training set, consisting of 997,055,154 examples, and non-overlapping testing set with 279,026,428 labeled examples. For the Ad Latency prediction task, the publishers prefer low False Positive Rate (FPR), ensuring that very few ads are wrongly classified as late, thus minimally hurting revenue. At the same time, the publisher prefer high True Positive Rate (TPR), which improves user experience and increases profit. As these two goals are often conflicting, the optimal strategy is to maximize the area under the Receiver

Table II: Performance comparison of AROW and AROW-MapReduce on Ad Latency task in terms of the AUC

# mappers	# reducers	Avg. map time	Reduce time	AUC
1	0	408h	n/a	0.8442
100	1	30.5h	1 min	0.8552
500	1	34 min	4 min	0.8577
1,000	1	17.5 min	7 min	0.8662
10,000	1	2 min	1h	0.8621

Operating Characteristic (ROC) curve, referred to as the AUC [37]. Thus, unlike in the experiments with the synthetic data, we report AUC as a measure of performance. Given a predicted margin for the n^{th} example, $\hat{y}_n \in \mathbb{R}$, a binary classification prediction is found as $\text{sign}(\hat{y}_n - \theta)$, where different values of threshold θ result in different predictions and different TPR and FPR values. We can obtain an ROC curve by sliding the threshold θ from $-\infty$ to ∞, and plotting the resulting TPR and FPR in a 2-D plot.

The experiments were conducted using MapReduce on Apache Hadoop, with AROW-MR mapper and reducer implemented in Perl. Performance of AROW-MR was compared to AROW algorithm which was run on a single machine, as well as to the logistic regression implemented in highly-scalable Vowpal Wabbit package, run both on a single machine and in a distributed manner using AllReduce on Hadoop. We ensured that Hadoop scheduled exactly M mappers by storing the data in M gzip-compressed part files.

We first compared the performance of AROW learned on a single machine with AROW-MR learned in a distributed manner. Similarly to experiments presented in Section V-A, we increased the number of mappers to evaluate the effects of different levels of parallelization. The results are given in Table II. We can see that the running time of AROW-MR drastically improved over the non-distributed AROW (trained using a single mapper). While it took 17 days for AROW to train, we were able to train accurate AROW-MR models in less than an hour. Expectedly, average mapper time decreased and reducer time increased as the number of mappers was increased, as each mapper was trained on smaller partition of the data and reducer was required to combine higher number of local models. Interestingly, we can also see that the results in Table II validate the results obtained on synthetic data regarding performance gains with increasing levels of parallelization. In particular, both the accuracy and training time improved until we reached $M = 1{,}000$, and dropped slightly for higher number of mappers. The detailed results are presented in Figure 2, where we plotted ROC curves of the confidence-weighted models trained using different number of mappers. We can see that the curve for non-distributed AROW, denoted by "1 mapper", results in the smallest AUC, while the level of parallelization achieved with 1,000 mappers represents the best choice for the Ad Latency prediction task.

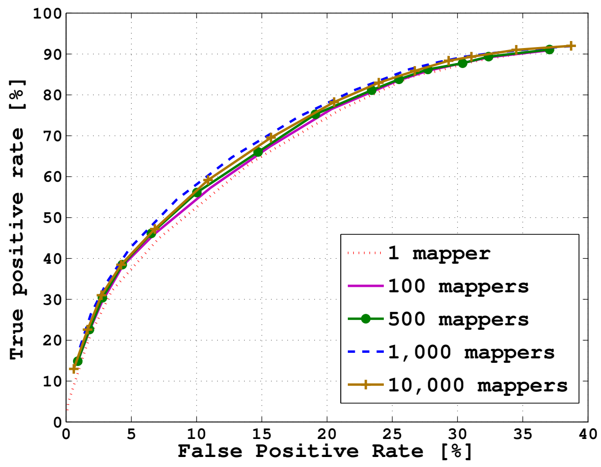


Figure 2: ROC curve for AROW-MR and AROW

Next, in Table III we show the performance of logistic regression (LR) model trained using VW package in both distributed mode (using AllReduce) and non-distributed manner (on a single machine). We can see that AROW-MR achieved higher accuracy than LR, which is a very popular approach using in large-scale classification. While AROW-MR obtained AUC of 0.8662, the best AUC achieved by LR was 0.8508; this increase in accuracy may result in significant increase of revenue in computational advertising domain. Interestingly, the results also indicate that more distributed training of LR actually hurt its generalization performance, which slightly dropped as the number of mappers was increased. Further, we can see that LR was trained in 8 minutes, while it took 25 minutes to train AROW-MR model. However, although LR training is seemingly faster than training of AROW-MR, it is important to emphasize that VW package implements logistic regression in C language while, for technical reasons, AROW-MR was implemented in Perl. Considering that Perl is not an optimal choice for mathematical computations, we expect AROW-MR training time to improve significantly once implemented in C.

It is also worth noting that we were not able to run the LR experiment for more than one thousand mappers. The LR implementation in VW package uses AllReduce framework, which requires that all mappers run concurrently without any node failures, otherwise the training might fail. However, this is usually hard to guarantee in practice even for larger clusters, and it further exemplifies the advantage of the proposed algorithm over the competition. We can conclude that AROW-MR offers robust, highly efficient training of accurate classifiers, outperfoming the existing state-of-the-art for extremely large-scale, industrial-size problems.

VI. Conclusion

In this paper we presented AROW-MR, a distributed linear SVM solver capable of learning accurate models in time sub-

Table III: Performance of distributed logistic regression

# mappers	# reducers	Avg. map time	Reduce time	AUC
1	0	7h	n/a	0.8506
100	0	1h	n/a	0.8508
500	0	8 min	n/a	0.8501
1,000	0	6 min	n/a	0.8498

linear in the size of a training set. The proposed method utilizes the MapReduce paradigm, which provides distributed, fault-tolerant environment for large-scale machine learning. Map phase of the method involves training a number of local AROW models on each mapper, followed by reduce phase which combines local classifiers to obtain a single, aggregated model more accurate than any of the individual ones. The experiments on synthetic data and real-world Ad Latency data set with nearly one billion examples indicate that AROW-MR allows for significant accuracy and training time improvements over the original AROW algorithm. Moreover, on Ad Latency task the proposed method outperformed distributed logistic regression available in a popular Vowpal Wabbit package, suggesting that the proposed distributed confidence-weighted model provides a scalable, accurate tool for large-scale classification problems.

Acknowledgment

The authors would like to thank Vladan Radosavljevic for helpful discussions. ND and SV would also like to acknowledge NSF support through grant IIS-0546155.

References

[1] C. Bizer, P. Boncz, M. L. Brodie, and O. Erling, “The meaningful use of big data: four perspectives–four challenges,” *ACM SIGMOD Record*, vol. 40, no. 4, pp. 56–60, 2012.

[2] A. Labrinidis and H. Jagadish, “Challenges and opportunities with big data,” *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2032–2033, 2012.

[3] S. Lohr, “The age of big data,” *New York Times*, vol. 11, 2012.

[4] J. Mervis, “Agencies Rally to Tackle Big Data,” *Science*, vol. 336, no. 6077, pp. 22–22, 2012.

[5] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[6] J. Platt, “Fast training of Support Vector Machines using Sequential Minimal Optimization,” *Advances in kernel methods - support vector learning, MIT Press*, 1998.

[7] J. Kivinen, A. J. Smola, and R. C. Williamson, “Online learning with kernels,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2002.

[8] S. V. N. Vishwanathan, A. J. Smola, and M. N. Murty, “SimpleSVM,” in *International Conference on Machine Learning*, 2003.

[9] A. Severyn and A. Moschitti, “Large-scale support vector learning with structural kernels,” in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 229–244.

[10] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, “Core vector machines: Fast SVM training on very large data sets,” *Journal of Machine Learning Research*, vol. 6, no. 1, p. 363, 2005.

[11] P. Rai, H. Daumé III, and S. Venkatasubramanian, “Streamed learning: one-pass SVMs,” in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2009, pp. 1211–1216.

[12] H. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik, “Parallel Support Vector Machines: The cascade SVM,” *Advances in Neural Information Processing Systems*, vol. 17, pp. 521–528, 2004.

[13] E. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui, “Psvm: Parallelizing support vector machines on distributed computers,” *Advances in Neural Information Processing Systems*, vol. 20, p. 16, 2007.

[14] Z. A. Zhu, W. Chen, G. Wang, C. Zhu, and Z. Chen, “P-packSVM: parallel primal gradient descent kernel SVM,” in *IEEE International Conference on Data Mining*, 2009, pp. 677–686.

[15] I. Steinwart, “Sparseness of support vector machines,” *Journal of Machine Learning Research*, vol. 4, pp. 1071–1105, 2003.

[16] C. Gentile, “A new approximate maximal margin classification algorithm,” *Journal of Machine Learning Research*, vol. 2, pp. 213–242, 2002.

[17] Y. Li, H. Zaragoza, R. Herbrich, J. Shawe-Taylor, and J. Kandola, “The perceptron algorithm with uneven margins,” in *International Conference on Machine Learning*, 2002, pp. 379–386.

[18] S. Shalev-Shwartz, Y. Singer, and N. Srebro, “Pegasos: Primal estimated sub-gradient solver for SVM,” in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 807–814.

[19] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, “LIBLINEAR: A library for large linear classification,” *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[20] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin, “Large linear classification when data cannot fit in memory,” *ACM Transactions on Knowledge Discovery from Data*, vol. 5, no. 4, p. 23, 2012.

[21] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] ——, “MapReduce: a flexible data processing tool,” *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[23] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford, “A reliable effective terascale linear learning system,” *arXiv preprint arXiv:1110.4198*, 2011.

[24] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, “GraphLab: A new parallel framework for machine learning,” in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2010, pp. 340–349.

[25] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, “Distributed GraphLab: A framework for machine learning and data mining in the cloud,” *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[26] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, “Pregel: a system for large-scale graph processing,” in *Proceedings of the ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[27] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash *et al.*, “Apache Hadoop goes realtime at Facebook,” in *International Conference on Management of Data*. ACM, 2011, pp. 1071–1080.

[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2010, pp. 1–10.

[29] J.-H. Böse, A. Andrzejak, and M. Högqvist, “Beyond online aggregation: Parallel and incremental data mining with online map-reduce,” in *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*. ACM, 2010, p. 3.

[30] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang, “Large-scale Image Classification: Fast Feature Extraction and SVM Training,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2011, pp. 1689–1696.

[31] A. Gesmundo and N. Tomeh, “HadoopPerceptron: a toolkit for distributed perceptron training and prediction with MapReduce,” in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2012, pp. 97–101.

[32] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun, “Map-reduce for machine learning on multicore,” *Advances in Neural Information Processing Systems*, vol. 19, p. 281, 2007.

[33] M. Dredze, K. Crammer, and F. Pereira, “Confidence-Weighted Linear Classification,” in *Proceedings of the International Conference on Machine Learning*. ACM, 2008, pp. 264–271.

[34] K. Crammer, A. Kulesza, and M. Dredze, “Adaptive Regularization of Weight Vectors,” *Advances in Neural Information Processing Systems*, vol. 22, pp. 414–422, 2009.

[35] J. Langford, L. Li, and T. Zhang, “Sparse online learning via truncated gradient,” *The Journal of Machine Learning Research*, vol. 10, pp. 777–801, 2009.

[36] C. Hughes and T. Hughes, *Parallel and distributed programming using C++*, pp. 31–32, 2004.

[37] T. Fawcett, “ROC graphs: Notes and practical considerations for researchers,” *Machine Learning*, vol. 31, pp. 1–38, 2004.