DCMSVM: Distributed Parallel Training For Single-Machine Multiclass Classifiers

Xufeng Han Alexander C. Berg
Computer Science Department
Stony Brook University

Abstract

We present an algorithm and implementation for distributed parallel training of single-machine multiclass SVMs. While there is ongoing and healthy debate about the best strategy for multiclass classification, there are some features of the single-machine approach that are not available when training alternatives such as one-vs-all, and that are quite complex for tree based methods. One obstacle to exploring single-machine approaches on large datasets is that they are usually limited to running on a single machine! We build on a framework borrowed from parallel convex optimization – the alternating direction method of multipliers (ADMM) – to develop a new consensus based algorithm for distributed training of single-machine approaches. This is demonstrated with an implementation of our novel sequential dual algorithm (DCMSVM) which allows distributed parallel training with small communication requirements. Benchmark results show significant reduction in wall clock time compared to current state of the art multiclass SVM implementation (Liblinear) on a single node. Experiments are performed on large scale image classification including results with modern high-dimensional features.

1. Introduction

As recognition in computer vision improves, researchers are pushing to recognize larger spaces of *labels*, from the 20 classes in Pascal VOC [15] to 1000 classes in the Pascal LSVRC [30] to 10,000+ classes in ImageNet [12] to over 100,000+ classes in work on learning similarity functions for web scale retrieval [6]. Even these numbers may seem small in the future when fine grained recognition using attributes effectively increases the label space by orders of magnitude [22, 26]. At the same time, high quality classification seems to require ever higher dimensional features whether for retrieval [23] or detection [16]. Together these factors present a formidable computational challenge.

This paper presents a new parallel algorithm for learning such large scale multiclass classifiers. In particular we present the first algorithm for efficiently distributing training of single machine classifiers, sometimes referred to as

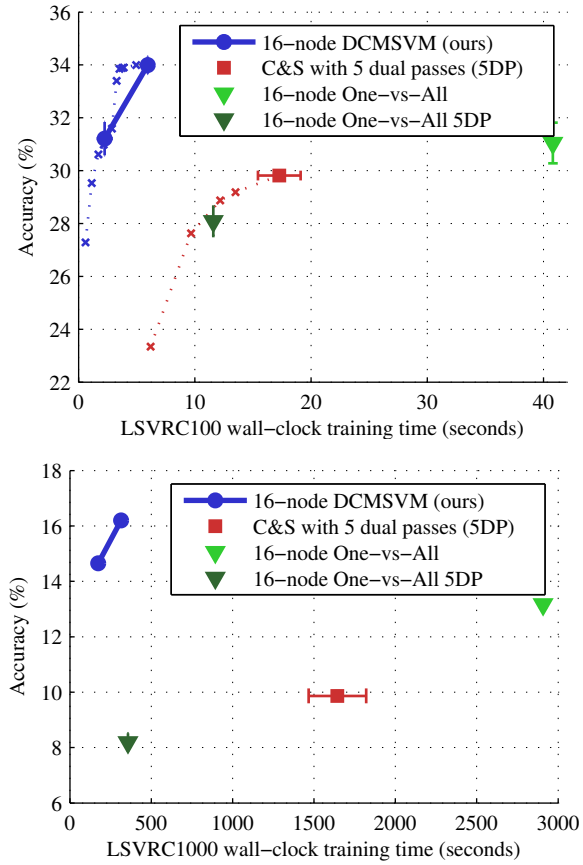

Figure 1. **Upper:** Image classification on LSVRC100 (100 classes). **Lower:** Image classification on LSVRC1000 (1000 classes). Comparison between our DCMSVM multiclass method and One-vs-all on 16 nodes, and Liblinear's Crammer & Singer implementation on a single node. Dashed lines in the uppper plot indicate progress during optimization (see Sec. 4). Accuracy is measured on held out test data. Error bars indicate variation in time and accuracy across 4 different train/test splits (they are too small to see for DCMSVM in the lower plot).

direct multiclass approaches. Such approaches offer potential advantages over the more commonly used *one versus all* multiclass techniques, but until now have been somewhat less scalable as no effective techniques were available for distributed parallelization.

This paper makes both a theoretical contribution – deriving a surprisingly simple sequential dual algorithm for parallel optimization of a direct multiclass SVM decision rule – and an experimental contribution – demonstrating that an implementation of this approach can significantly improve accuracy for a given amount of wall clock training time compared to the state of the art for single-machine methods using current computer vision data and features. As a result, single-machine methods can be trained on larger datasets than had been practical in the past, and can more used more broadly!

To summarize our contributions we present:

- A new algorithm called Distributed Consensus Multiclass SVM (DCMSVM) for efficient distributed parallel training of "single machine" or direct multiclass SVMs. Making them much more widely usable. To our knowledge, this is the first such algorithm.
- An implementation of our algorithm to be released upon publication.
- Benchmarks on multiclass image classification including some using current high dimensional descriptors from state of the art systems. Results show significant improvements in wall-clock time and accuracy versus the very efficient implementation of Crammer & Singer's algorithm [10] in Liblinear [21].

2. Background

Single-machine methods train a multiclass classifier by setting up a single large optimization problem tying together all parameters and as a result are computationally expensive. There have been a number of recent papers exploring parallelization of training for models in related contexts – for conditional maxent models [24], structured prediction[25], stochastic gradient descent [31], and others. Albeit to varying degrees, many of these approaches share a common idea of training models in parallel and combining the trained parameters. In fact the first stage of processing in the method proposed here follows the same procedure. One perhaps surprising result here is that, at least in some settings, it is possible to improve upon this initial step – in terms of the time / accuracy trade-off – by using the combined model as regularization for further optimization. Furthermore this can produce better classifiers than using more data (cf [31]) or more time in the initial step.

We begin by considering a simple formulation for multiclass classification where a function f_c is learned for each class c and a data item x is classified into class $\text{argmax}_c f_c(x)$. There are a number of ways to learn the f_c, but we first focus on the distinction between one-vs-all based methods and single machine methods. In one-vs-all, each f_c is trained *independently* to give a large response to data items from class c (the "one") and a small response to all other classes (the "all" sometimes called the "rest"). The constraints for one-vs-all training generally have the form $f_c(x_p) > f_c(x_n)$ where x_p is any training item with label c and x_n is any training item with label $c' \neq c$. Single-machine approaches link the training of the functions f_c together so that for a training item x of class c, $f_c(x) > f_{c'}(x)$ for $c' \neq c$. There are many variations on those ideas – especially some considering the effects of different ways to regularize learning using margins – but the basic form of the constraints are often similar.

The single-machine approach has some potential advantages in weighing different types of errors during *training*. For instance it is possible to put more weight on avoiding confusing class a with class b than avoiding confusing class a with class c. The one-vs-all approach can only adjust weights on avoiding confusing class a with "any other class". This ability becomes more important when there is a natural hierarchical structure to classes (*e.g.* "animals", "dogs", and "black lab") where some are more similar than others and there is a notion of some labels being less different than others (*e.g.* "brown lab" and "black lab" vs "dog" and "cat"). In addition when there are many classes or classes are sampled sparsely or unevenly single-machine approaches may have an advantage. See Figure 2 for an example of the single-machine approach being more robust to some classes having a smaller number of training examples. Performance for the under-sampled classes is significantly higher for the single-machine approach as compared to one-vs-all (middle vs right of Figure 2).

Despite these potential advantages, it is much less straightforward to parallelize single-machine learning. This is in stark contrast to the one-vs-all approach, which for n classes easily distributes across as many as n machines or processors – simply training a single one-vs-all classifier on each node. Of course intermediate solutions where k one-vs-all classifiers are trained on each of $\frac{n}{k}$ nodes are also possible.[1]

In this work we explore an alternative approach that looks at training a multiclass classifier as a convex optimization problem in a general framework for parallelizing convex optimization, the alternating direction method of multipliers (ADMM). The result is a new algorithm distributing training of a single-machine classifier across multiple

[1] We note that this easy parallelization of one-vs-all approaches still requires *all* the data to be available to train each classifier. See [14] for a clever solution requiring only part of the data on each node at the cost of some communication overhead.

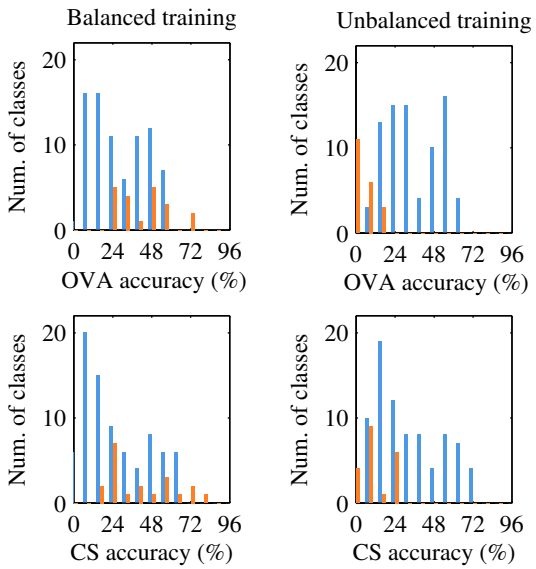

Figure 2. C&S is more stable for unbalanced training than 1-vs-all. See Sec. 4.5.

nodes. Training is distributed across nodes by partitioning the data, but terms are added to encourage the individual nodes toward a common (consensus) solution. This approach (exactly) linearly decreases the amount of data processed per node and (nearly) linearly decreases the computation time per node. Generally training proceeds in stages, where parameters from the parallel computations are averaged after each stage. The key difference from other work is that the averages from the previous round are used to regularize the following round. There are several attractive properties of the specific realization of this approach – it is easy to adjust the total amount of computation, there is low communication overhead (infrequent communication of model parameters, never data), and it is straightforward to minimize or even eliminate synchronization delays. The *key* mathematical insight making this efficient is that there is a simple formulation for a dual version of the training objective augmented with the consensus terms. This formulation is not only simple, but is also amenable to a very similar treatment as that utilized in efficient sequential dual methods for SVM training on a single machine.

Generally there is wide interest in scaling discriminative classifiers like SVMs to deal with increasingly large datasets. Computation becomes too great to fit on a single machine and there have been many pieces of work designed scale SVM training by carefully distributing kernel computations and values *e.g.*, [5] or splitting classification into a network of decisions [18, 9] and others. We note that these approaches and the methods in this paper address distributing computation across a standard cluster of computers as opposed to specialized compute infrastructures where tighter parallelism is effective e.g. multi-core [7] or GPU style approaches. Nevertheless such techniques may indeed be used to speed up computation on individual nodes for our method.

In addition to the methods for efficient parallel learning of models mentioned in the introduction, Bengio *et al.* [1] (*c.f.* [14]) present an approach for learning a tree structured classifier to improve efficiency for evaluation with a large number of image classes – a simpler approach but with similar motivation to Filter-Trees from [3]. This is entirely compatible with our approach which could be seen as an effective way to scale the learning required for individual nodes in the decision tree which may still consider several hundred classes and can benefit from training on large amounts of data. In addition [1] describe the landscape for efficient large scale classification.

The alternating direction method of multipliers approach is well researched, see Bertsekas & Tsitsiklis [2] using ADMM for consensus optimization and Boyd *et al.* [4] for a wonderfully clear overview including summary and citations for convergence results. We know of only one previous piece of work using an ADMM approach to develop a consensus algorithm for distributed SVM training from Forero *et al.* [17]. Although the motivation and high level approach have similarities in spirit to our own, there are several differences. The main ones being that [17] do not derive or consider algorithmic solutions to the subproblem to be run on each node, and that they do not consider multi-class, either weighted or un-weighted. Finally they do not report results on the accuracy/time trade-off focusing more on high level capabilities for robust distributed optimization. In particular they look carefully at convergence to the non-distributed result (after a large number of iterations), which we find is far less relevant than accuracy after a small number of iterations – at least in the pursuit of reaping a wall clock time advantage from distributed computation.

In some recent work [27] there is ongoing debate about whether single-machines are obviously better than one-vs-all approaches to multiclass classification. We note that they explicitly exempt large scale problems with very many non-uniformly sampled classes or classes with varying levels of distinction from consideration (hierarchies of classes) – exactly the cases we are interested in pursuing for many computer vision problems!

Finally as part of this work we have implemented our approach inside the Liblinear framework [21] which not only provides a solid baseline, but also a very flexible and useful environment for developing new algorithms that can be evaluated at large scale. It includes shrinking which we used and out-of-core processing which we did not use, but is compatible with out approach and would allow additional scaling with lower in-core memory requirements.

3. Consensus Optimization for Training Multiclass SVMs

We start from Crammer & Singer's K-class multiclass SVM formulation[10]:

$$\begin{aligned} \underset{w_1,\cdots,w_K}{\text{minimize}} \quad & \frac{\lambda}{2}\sum_{c=1}^{K} w_c^T w_c + \sum_{i=1}^{N}\xi_i, \\ \text{subject to} \quad & (w_{y_i} - w_c)^T x_i \geq 1 - \xi_i - \delta_{y_i,c} \\ & \forall i = 1,\cdots,N, c = 1,\cdots,K. \end{aligned} \tag{1}$$

Here each class c has a weight vector w_c, and each of the N training items x_i has label y_i. The indicator function $\delta_{y_i,c} = 1$ if $y_i = c$ and 0 otherwise. The variables ξ_i are slack variables for each data item, so that only the margin between the correct class and the most confusing class is penalized – this is the main novelty of the Crammer & Singer formulation vs some others. Note that the constraint also compactly enforces that $\xi_i \geq 0$ when $c = y_i$.

One attractive aspect of the Crammer & Singer work is an efficient sequential dual algorithm for solving the problem. However the number of dual variables grows linearly with the number of training samples with a factor of K, the number of classes. In the context of image classification, both K and the number of samples per class can be large, so we would like to split the data into smaller sets that are each tractable on a single computing node. We can break up the objective function over splits of the data. Let

$$f(\lambda, w_1, \ldots, w_K, x_1, \ldots, x_N) = \frac{\lambda}{2}\sum_{c=1}^{K} w_c^T w_c + \sum_{i=1}^{N}\xi_i \tag{2}$$

so that we can write the objective function in terms of S splits of the data:

$$f(\lambda, w_1, \ldots, w_K, x_1, \ldots, x_N) \tag{3}$$

$$= \sum_{s=1}^{S} f(\frac{\lambda}{S}, w_1, \ldots, w_K, x_{(s-1)\frac{N}{S}+1}, \ldots, x_{s\frac{N}{S}}) \tag{4}$$

We can see that each split will solve for a multiclass classifier on a subset of the data with a smaller regularization parameter.

At the same time we want the solutions to all be the same, a consensus optimization problem.

3.1. Consensus optimization using ADMM

The idea of consensus optimization is to decompose the original problem into subproblems and solve each of the subproblems while at the same time constraining the solution to the subproblem to be equal. For instance if we can split our objective function f into S functions f_s so that $f(w) = \sum_{s=1}^{S} f_s(w)$, then we want to solve

$$\text{minimize} \sum_{s=1}^{S} f_s(w_s) \tag{5}$$

$$\text{subject to } w_s - z = 0, \quad s = 1,\ldots,S. \tag{6}$$

Although a simple dual decomposition followed by a dual descent method can be used to solve the problem, it converges slowly. To help convergence, a linear and quadratic term are introduced, forming the *augmented Lagrangian*:

$$\begin{aligned} L_\rho(w_1,\ldots,w_S,z,y) = &\sum_{s=1}^{S}(f_s(w_s) + y_s^T(w_s - z) \\ &+ (\rho/2)\|w_s - z\|_2^2), \end{aligned} \tag{7}$$

where y are the dual variables. Note that the augmentation, effectively a smoothing term that aids convergence scaled by ρ, has no effect on the solution, only on the convergence of the algorithm to follow. It is shown in [4] that at the k-th iteration $z^k = \overline{w}^k$, so that we can optimize by alternately solving for:

$$\begin{aligned} w_s^{k+1} :=& \underset{w_s}{\text{argmin}}(f_s(w_s) + (y_s^k)^T(w_s - \overline{w}^k) \\ &+ (\rho/2)\|w_s - \overline{w}^k\|_2^2) \end{aligned} \tag{8}$$

$$y_s^{k+1} := y_s^k + \rho(w_s^{k+1} - \overline{w}^{k+1}). \tag{9}$$

These have a relatively simple interpretation: At the first part of each iteration, S separate optimizations are performed on the original objective functions f_s along with two terms that encourage consensus — one weighted by the dual variables y — and the augmenting term weighted by ρ. The second part of each iteration consists of calculating new dual variables y_s^{k+1}, and calculating $\overline{w}^{k+1}$ the mean of the w_s^{k+1}. We can consider $w^k = \overline{w}^k$ the solution after iteration k. Note that only the w^k, $\overline{w}^{k+1}$ need to be communicated between the independent optimizations — hence the low communication overhead. So far we have introduced the standard consensus optimization framework using ADMM algorithm. We refer the reader to Boyd *et al.* [4] for a very clear review.

Substituting one of the objective functions in Equation 4 into the augmented consensus optimization in Equation 7, and renumbering the data items for each split as $1 \ldots N/S$ for notational simplicity we have

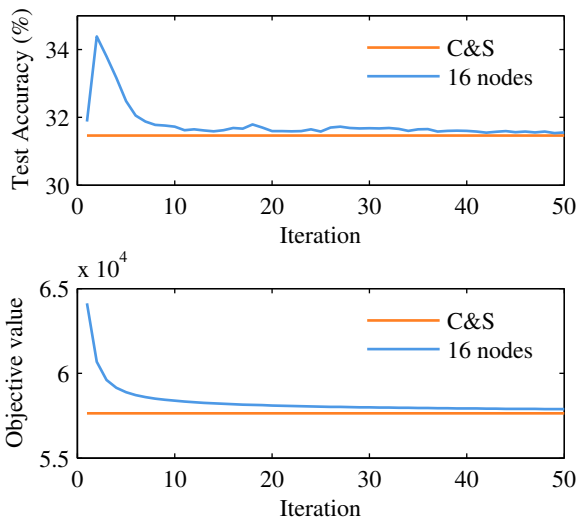

Figure 3. Convergence of DCMSVM (on 16 nodes) in terms of test accuracy (top) and objective value (bottom).

$$\begin{aligned}
\underset{w_1,\cdots,w_K,\xi_1,\cdots,\xi_{N/S}}{\text{minimize}} \quad & \frac{\lambda}{2S}\sum_{c=1}^{K} w_c^T w_c + \sum_{i=1}^{N/S}\xi_i \\
& + \sum_{c=1}^{K}\alpha_c^T(w_c-\overline{w}_c) + \frac{\rho}{2}\sum_{c=1}^{K}\|w_c-\overline{w}_c\|^2, \\
\text{subject to} \quad & (w_{y_i}-w_c)^T x_i \geq 1-\xi_i-\delta_{y_i,c} \\
& \forall i=1,\cdots,N/S, c=1,\cdots,K.
\end{aligned} \tag{10}$$

Each subproblem above differs from the original Crammer & Singer problem due to the extra linear and a quadratic terms measuring how close the solution is to the consensus. The problem is still quadratic and convex, and ρ determines how much to weigh consensus.

3.2. Dual Formulation for Distributed Consensus Multiclass SVM (DCMSVM)

A number of recent methods devised especially for Crammer & Singer multiclass SVM have been proposed [28, 8, 19, 20, 21]. Keerthi et al's sequential dual method follows Crammer & Singer's original approach closely and demonstrated good performance in experiments [21]. In search of a similar method to solve our consensus formulation, we derive the Langrangian dual (see the supplementary material for details) of the subproblem Equation 10. The dual turns out to have the following simple form:

Algorithm 1 DCMSVM

- Divide the training data into S splits, $T_1 \ldots T_S$;
- $\overline{w} \leftarrow 0$;
- FOR $k := 1$ TO max_iter DO
  - FOR $s := 1$ TO S DO IN PARALLEL
    - $\beta \leftarrow 0; \quad w^s \leftarrow -\frac{\rho}{\lambda/S+\rho}\overline{w}$
    - UNTIL $v_i < \epsilon$ (tolerance parameter), $\forall i$ DO
      - FOR $i := 1$ TO $|T_s|$ DO
        - Compute g_i^c using Equation 12;
        - $v_i \leftarrow \max_c g_i^c - \min_{c:\beta_i^c<C_i^c} g_i^c$ (See Equation 6 in [21]);
        - If $v_i < \epsilon$, find β_i' using FixedPointAlgorithm[10]
          - $\Delta\beta_i \leftarrow \beta_i' - \beta_i; \quad \beta_i \leftarrow \beta_i';$
          - $w_c^s \leftarrow w_c^s + \Delta\beta_{i,c}x_i;$
      - END
    - END
    - $\overline{w} \leftarrow \frac{1}{S}\sum_s w^s;$
  - END
- RETURN $\overline{w}$;

$$\begin{aligned}
\underset{\beta}{\text{minimize}} \quad & h(\beta) = \frac{1}{2}\sum_{c=1}^{K}\|w_c(\beta)\|^2 + \sum_{i,c}\beta_{i,c}e_{i,c}, \\
\text{subject to} \quad & \beta_{i,c} \leq C\delta_{y_i,c}, \quad \sum_{c=1}^{K}\beta_{i,c} = 0, \\
& \forall i=1,\cdots,N/S, \quad c=1,\cdots,K,
\end{aligned} \tag{11}$$

where $C = 1/(\lambda+\rho)$, $e_{i,c} = 1-\delta_{y_i,c}$, and $w_c(\beta) = \sum_{i=1}^{N}\beta_{i,c}x_i - Ct_c$. Here, $t_c = \alpha_c - \rho\overline{w}_c$ is a constant vector for each class. Although the relation between the primal and dual variables, respectively w_c and β, are different from those in Crammer and Singer's dual formulation [10], the resemblance in form suggests the sequential dual method (SDM) introduced in [21].

3.3. Sequential Dual Algorithm for Subproblems in DCMSVM

We will develop a sequential dual method (SDM) to solve the subproblem defined by Equation 11, following the general procedure from Keerthi *et al*, with the inner-most optimization following Crammer and Singer's algorithm as described in Section 6 of [11]. The idea is to iteratively consider a data item x_j and solve for the corresponding dual variables $\beta_j = (\beta_{j,1},\ldots,\beta_{j,K})$ (K is the number of classes) that minimize $h(\beta)$ with respect to β_j. The primal solution, $w = (w_1,\ldots,w_K)$, is then incrementally updated

using the difference between the new β_j and the old. These along with other efficient implementation techniques such as shrinking are covered in [21].

The gradient of $h(\beta)$ is the key to whether a SDM is possible. In our problem, the gradient, expressed in terms of w as follows,

$$g_i^c = \frac{\partial h(\beta)}{\partial \beta_{i,c}} = w_c(\beta)^T x_i + e_{i,c}, \quad \forall i = 1, \cdots, N/S, \quad c = 1, \cdots, K. \tag{12}$$

has the same general form as that in [21]. As a result the sequential update can be applied naturally and the only change in implementation necessary is to initialize $\beta = 0$ and $w_c = -Ct_c$. Details are covered in the supplemental material. This has a simple, rough, interpretation as adding the consensus and augmenting terms by changing the "center" of the dual variables – effectively pushing the solution toward the average.

We write the the overall algorithm in Algorithm 3.3.

4. Experiments

We perform a series of experiments to verify that our DCMSVM algorithm is effective for distributed parellelization of training a "single-machine" multiclass classifier, and that it compares favorably with respect to both Liblinear's Cramer & Singer implementation on a single node, as well as the standard one-vs-all, winner take all, approach to multiclass classification distributed across nodes.

For these experiments we use three datasets. All are subsets of the PASCAL Large Scale Visual Recognition Challenge dataset (LSVRC) [29], which in turn is part of ImageNet [13]:

- **LSVRC100** consists of 100 classes randomly sampled from LSVRC with 800 images per class. Features are 1000 dimensional SIFT BOW released by LSVRC.
- **LSVRC1000** consists of all 1000 classes in LSVRC with 600 imagees per class. Features are 1000 dimensional SIFT BOW released by LSVRC.
- **LSVRC100-HD** consists of 100 classes randomly sampled from LSVRC with 600 images per class. Features are **210,000-dimensional** local coordinate code (LCC) features [23].

Each dataset is evenly divided into four folds for cross validation, and each training split is further divided into four "train"/"validation" subsets for parameter selection. For all methods, except where noted, the regularization parameter λ for each experiment was chosen to be the best on the train/validation sets for each experiment cross validation fold, and the tolerance parameter $\epsilon = 0.1$.

In the first ADMM iteration of our DCMSVM algorithm, each node solves the multiclass problem on it's own split of the data in parallel. These parameters are averaged and used in the second iteration following to initialize the subsequent iteration following Equations 8&9.

Result Plots: We present most of the results in time/accuracy plots. In those plots, larger markers connected by thick solid lines show results after ADMM iterations in DCMSVM. Smaller markers connected by dotted lines show progress of the optimization inside each iteration, and are plotted after a certain number of passes of the sequential dual method through the data (after shrinking) in each problem . Since sequential dual methods like our DCMSVM and Liblinear's Cramer & Singer solver can be stopped at any given point and output the current estimate on w, those intermediate results are not difficult to obtain.

The first order result is that for all datasets, our DCMSVM produces classifiers that are at least as accurate (and usually more than as accurate) as the baseline 1-vs-all winner take all approach, and Liblinear's Cramer & Singer implementation. Furthermore in terms of wall-clock-time our approach trains these models more quickly than either Liblinear on a single node or 1-vs-all distributed on the same number of nodes as DCMSVM. See Figs. 1 and the LSVRC100-HD results in Sec. 4.4.

4.1. Convergence and Regularization

ADMM methods are known to converge [4] in the limit, although sometimes many iterations are required to achieve high accuracy in approximating the non-distributed objective. In practice only a few iterations of DCMSVM are necessary to get good solutions – in all our experiments, solutions after a few iterations were consistently superior to the non-distributed solution. A natural question would be whether the boost in performance is a result of some (positive) artifact of effective regularization from averaging solutions based on subsets of the data, and whether the same effect is achievable by adjusting λ in the original single machine formulation. Our answer to the second question is "No". Figure 5 (top) shows that no choice of λ allows the single-split version (Crammer & Singer) to match the 16 split version of DCMSVM[2]. We can choose the number of iterations to run as part of parameter search during cross-validation on training data – in our experiments cross validation almost always chooses 2 iterations. In some plots we show more than two iterations to provide an idea of what happens as optimization progresses.

4.2. Time/accuracy trade-off and number of splits

As can be seen in all the figures, there is a significant reduction in wall clock time from distributing training. We

[2]Fig. 5 also indicates some numerical instability for Liblinear's C&S implementation when using small λ.

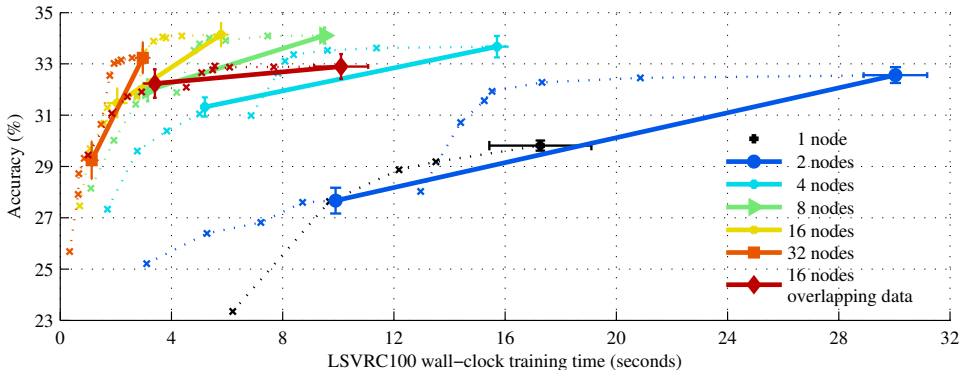

Figure 4. Comparison between our DCMSVM split across varying numbers of nodes and Liblinear's Crammer & Singer implementation on ILSVRC100. Dashed lines indicate progress during optimization. See Sec. 4.2 for details.

see in Figure 1 (left) that DCMSVM with 16 splits after two iterations has higher accuracy than Cramer & Singer on a single node and is about 2.5 times faster in wall clock time, while doing about 6.5 times as much total computation. The progress of optimization is shown in many of the plots and indicates that more aggressive early stopping could produce even larger speedups. Note that for more than 16 splits we begin to see a decrease in accuracy after 2 iterations – although solutions are still significantly better than a single split. In Figure 4 we include a plot using 16 splits where each split has 1/8 of the data (instead of 1/16th of the data). This does not improve the accuracy vs wall-clock time trade-off, and careful examination of the progress during optimization shows that earlier stopping would not help. Even more impressive speedups are achieved for 1000 classes as shown in Figure 1 (right).

To make a fair comparison, we split the one-vs-all training tasks across the same number of nodes as used for DCMSVM. Note that one-vs-all training requires all the data to train each classifier, unlike DCMSVM which requires only a subset of the data in each split. In this paper all algorithms run single threaded on a node for the sake of comparison. All three can be accelerated by taking advantage of parallelism on each node.

4.3. Early Stopping

In almost all experiments early stopping after 5 passes through the data (with shrinking) was used in the sequential dual optimization for DCMSVM. In Figure 5 we show results without early stopping and with early stopping. We also show the detailed progress of the Cramer & Singer optimization, showing that, while early stopping might help a little, the effects are not as dramatic as for DCMSVM. We also experimented with early stopping after five passes for one-vs-all training and found improved speed at the cost of lower classifier accuracy for both LSVRC100 and LSVRC1000 datasets. See Fig. 1.

4.4. High Dimensional Image Features

High dimensional features such as local coordinate coding used by [23] in their winning Pascal Large Scale Visual Recognition Challenge results have shown good discriminative power in large scale image classification tasks. We ran DCMSVM on our ILSVRC100-HD dataset to test its behavior in handling high dimensional features. For this experiment we use DCMSVM split over 8 nodes, achieving accuracy of 61.2% in 1509 seconds vs 61.7% in 2959 seconds for Liblinear's Cramer & Singer implementation.

4.5. Unbalanced Training Set

The LSVRC data provides an almost uniform number of examples per class – and we use an exactly uniform number of examples per class in most of our experiments for the purpose of benchmarking – but this is unrealistic in many scenarios. In fact unbalanced training data is one of the settings where a single machine, direct training, method for multiclass classifiers can have an advantage. To demonstrate this we train a classifier for the 100 class LSVRC100 dataset using Cramer & Singer and one-vs-all. We separate the the classes into a set of 80 and a set of 20. Figure 2 top-left and bottom-left, show histograms of accuracy for the two sets of classes with equal training per class. When only 1/5th of the training data is used for the 20 classes, then accuracy on those classes reduces dramatically, with *larger decrease* in performance for the one-vs-all approach (top-right) than for the single machine approach (bottom-right).

5. Conclusion

We derived an efficient consensus based distributed parallel training algorithm for Crammer & Singer's multiclass single-machine SVM formulation. The key mathematical insight was an efficient dual method for the consensus augmented optimization problem. The approach is validated with an implementation and extensive evaluation on image classification, showing advantages in both accuracy and

wall-clock training time. This method allows single machine methods to scale to larger problems than had previously been possible.

Acknowledgment Funding from Stony Brook University Office of the Vice President of Research.

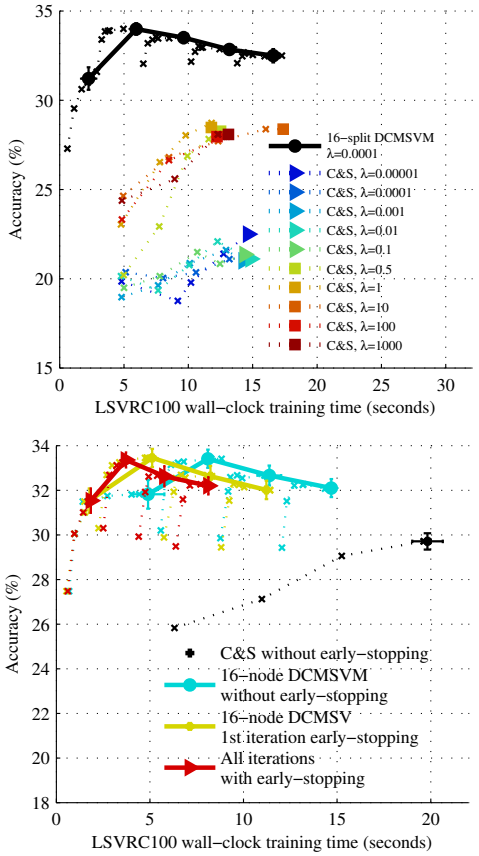

Figure 5. Comparison between our solution split across 16 nodes and Liblinear's Crammer & Singer implementation on the ILSVRC100 data. Dashed lines indicate progress during optimization. Accuracy is measured on held out data. Error bars indicate variation in time and accuracy across different train/test splits. **Top** Shows that no λ allows a single split to match 16 splits in generalization accuracy, see Sec. 4.1. **Bottom** Effects of early stopping over 6 validation splits ($\lambda = 10$) see Sec. 4.3.

References

[1] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *NIPS*, 2010. 3

[2] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation:Numerical Methods*. AthenaScientific, 1997. 3

[3] A. Beygelzimer, J. Langford, and P. Ravikumar. Error-correcting tournaments. *ALT*, pages 247–262, 2009. 3

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends(R) in Machine Learning*, 3(1), 2011. 3, 4, 6

[5] E. Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, , H, and Cui. Psvm: Parallelizing support vector machines on distributed computers. In *NIPS*, 2007. 3

[6] G. Chechik, V. Sharma, U. Shalit, and S. Bengio. Large Scale Online Learning of Image Similarity Through Ranking. *JMLR*, 11:1109–1135, Mar. 2010. 1

[7] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS*. 2007. 3

[8] M. Collins, A. Globerson, T. Koo, X. Carreras, and P. L. Bartlett. Exponentiated gradient algorithms for conditional random fields and max-margin markov networks. *JMLR*, 9:1775–1822, June 2008. 5

[9] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. In *NIPS*, 2002. 3

[10] K. Cramer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292, 2001. 2, 4, 5

[11] K. Cramer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47, 2002. 5

[12] J. Deng, A. Berg, K. Li, and L. Fei-Fei. What does classifying more than 10,000 image categories tell us? In *ECCV*, 2010. 1

[13] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 6

[14] J. Deng, S. Satheesh, A. Berg, and L. Fei-Fei. Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*, 2011. 2, 3

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 88(2):303–338, June 2010. 1

[16] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *TPAMI*, 32(9):1627–1645, 2010. 1

[17] P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-based distributed support vector machines. *JMLR*, 11:1663–1707, 2010. 3

[18] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade svm. In *NIPS*. 2005. 3

[19] T. Joachims. Training linear SVMs in linear time. KDD, pages 217–226. ACM, 2006. 5

[20] T. Joachims, T. Finley, and C.-N. J. Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009. 5

[21] S. S. Keerthi, S. Sundararajan, K. wei Chang, C. jui Hsieh, and C. jen Lin. A sequential dual method for large scale multi-class linear svms. In *KDD*, 2008. 2, 3, 5, 6

[22] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and Simile Classifiers for Face Verification. In *ICCV*, 2009. 1

[23] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, and K. Yu. Large-scale Image Classication:Fast Feature Extraction and SVM Training. In *CVPR*, 2011. 1, 6, 7

[24] G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *NIPS*. 2009. 2

[25] R. McDonald, K. Hall, and G. Mann. Distributed training strategies for the structured perceptron. In *NAACL*, 2010. 2

[26] D. Parikh and K. Grauman. Relative attributes. In *ICCV*, 2011. 1

[27] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *JMLR*, 5:101–141, December 2004. 3

[28] C. H. Teo, A. Smola, S. Vishwanathan, and Q. V. Le. A scalable modular convex solver for regularized risk minimization. In *ACM SIGKDD*, pages 727–736, 2007. 5

[29] http://www.image-net.org/challenges/LSVRC/2010/. 6

[30] http://www.image-net.org/challenges/LSVRC/2011. 1

[31] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. In *NIPS*. 2010. 2