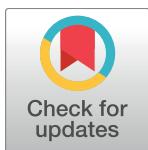


RESEARCH ARTICLE

Distributed optimization of multi-class SVMs

Maximilian Alber¹*, Julian Zimmert²*, Urun Dogan³, Marius Kloft²***1** Berlin Big Data Center, Berlin Institute of Technology, Berlin, Germany, **2** Department of Computer Science, Humboldt University of Berlin, Berlin, Germany, **3** Microsoft Research, Cambridge, United Kingdom

* These authors contributed equally to this work.

* kloft@hu-berlin.de

OPEN ACCESS

Citation: Alber M, Zimmert J, Dogan U, Kloft M (2017) Distributed optimization of multi-class SVMs. PLoS ONE 12(6): e0178161. <https://doi.org/10.1371/journal.pone.0178161>**Editor:** Quan Zou, Tianjin University, CHINA**Received:** January 19, 2017**Accepted:** May 8, 2017**Published:** June 1, 2017**Copyright:** © 2017 Alber et al. This is an open access article distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.**Data Availability Statement:** All data is available in the LSHTC corpus: <http://lshtc.iit.demokritos.gr/>.**Funding:** Some authors were supported by academic funding: MK acknowledges support by the German Research Foundation (DFG) under KL 2698/2-1 and by the Federal Ministry of Education and Research (BMBF) under 031L0023A and 031B0187B. MA acknowledges support by the Federal Ministry of Education and Research (BMBF) under 01IS14013A. Besides support in form of salaries or scholarships, the funders did not have any additional role in the study design, data collection and analysis, decision to publish, or preparation of the manuscript. The funder

Abstract

Training of one-vs.-rest SVMs can be parallelized over the number of classes in a straight forward way. Given enough computational resources, one-vs.-rest SVMs can thus be trained on data involving a large number of classes. The same cannot be stated, however, for the so-called all-in-one SVMs, which require solving a quadratic program of size quadratically in the number of classes. We develop distributed algorithms for two all-in-one SVM formulations (Lee et al. and Weston and Watkins) that parallelize the computation evenly over the number of classes. This allows us to compare these models to one-vs.-rest SVMs on unprecedented scale. The results indicate superior accuracy on text classification data.

Introduction

Modern data analysis requires computation with a large number of classes. As examples, consider the following. (1) We are continuously monitoring the internet for new webpages, which we would like to categorize. (2) We have data from an online biomedical bibliographic database that we want to index for quick access to clinicians. (3) We are collecting data from an online feed of photographs that we would like to classify into image categories. (4) We add new articles to an online encyclopedia and intend to predict the categories of the articles. (5) Given a huge collection of ads, we want to built a classifier from this data.

The problems above—taken from varying application domains ranging from the sciences to technology—involve a large number of classes, typically at least in the thousands. This motivates research on scaling up multi-class classification methods. In the present work, we address scaling up multi-class support vector machines (MC-SVMs) [1]. There are two major types of MC-SVMs:

1. One-vs.-one (OVO) and one-vs.-rest (OVR) MC-SVMs decompose the problem into multiple binary subproblems that are subsequently aggregated [1, 2]. Training can be parallelized in a straight forward way.
2. *All-in-one* MC-SVMs extend the concept of the margin to multiple classes. Because there is no unique extension of the margin concept, multiple all-in-one MC-SVMs have been proposed, including the ones by Crammer and Singer (CS) [3], Lee, Lin, and Wahba (LLW) [4], and Weston and Watkins (WW) [1, 5]. See [2, 6–11] for conceptual and empirical comparisons.

Microsoft Research provided support in the form of salaries for author UD, but did not have any additional role in the study design, data collection and analysis, decision to publish, or preparation of the manuscript. The specific roles of this author are articulated in the author contributions section.

Competing interests: UD is supported in the form of salary by Microsoft Research. There are no patents, products in development, or marketed products to declare. This does not alter our adherence to PLOS ONE policies on sharing data and materials.

Recently, Dogan et al. [11] have compared the various all-in-one MC-SVM variants on rather moderately sized datasets and showed advantages of all-in-one MC-SVMs over OVR MC-SVM, but—so far—slow training time has prohibited comparisons on data involving a large number of classes.

The reason is that (linear) state of the art solvers require time complexity of $\mathcal{O}(\bar{d}\bar{n} \cdot \mathcal{C}^2)$ and memory complexity at least of $\mathcal{O}(\bar{n}\mathcal{C}^2)$, where d is the feature dimensionality, \bar{d} the average number of non-zeros ($\bar{d} = d$ for dense data), and \bar{n} the average number of samples per class. This quadratic dependence on the number of classes \mathcal{C} can be prohibitive for large \mathcal{C} , often leaving OVO and OVR as the only MC-SVM options in the big data setting.

In this paper, we focus on the comparison between OVR SVMs and all-in-one SVMs. We do this by developing distributed algorithms where up to $\mathcal{O}(\mathcal{C})$ nodes solve WW and LLW in parallel, dividing model and computation evenly. The resulting solvers are compared to a state-of-the-art OVR solution.

The algorithm proposed for WW draws inspiration from a major result in graph theory: the solution to the 1-factorization problem of a graph [12]. The idea is that the optimization of a single coordinate $\alpha_{i,c}$ of the dual objective involves only the two hypotheses w_y and w_c . As in the 1-factorization problem, we can thus form pairs of classes where the corresponding blocks of coordinates can be optimized in parallel.

On the other hand, we parallelize LLW training by introducing an auxiliary variable \bar{w} into the dual problem that decouples the objective into a sum over \mathcal{C} many independent subproblems.

We provide both multi-core and distributed implementations of the proposed algorithms. We report on empirical runtime comparisons of the proposed solvers with the one-vs.-rest implementation by LIBLINEAR [13] on text classification data taken from the LSHTC corpus [14].

The main contributions of this paper are the following:

- We propose the first distributed, exact solver for WW and LLW.
- We provide both multi-core and truly distributed implementations of the solver.
- We give the first comparison of WW, LLW, and OVR on the DMOZ data from the LSHTC '10-'12 corpora using the full feature resolution.

We expect that the present work starts a line of research on parallelization of exact training of various all-in-one MC-SVMs, including Crammer and Singer, multi-class maximum margin regression [15], and the reinforced multiclass SVM [16], enabling comparison of all these methods on large datasets.

The paper is structured as follows. In the next section we discuss the problem setting and preliminaries. In Section Distributed Algorithms, we present the proposed distributed algorithms for LLW and WW, respectively. We analyze their convergence empirically in Section Experiments. Followed by sections Discussion of related work and Conclusion.

Preliminaries

We consider the following problem. We are given data $(x_1, y_1), \dots, (x_n, y_n)$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{1, \dots, \mathcal{C}\}$. Each class has in average \bar{n} samples. The largest number of samples for a single class is n_{max} . We are predicting using the model

$$\hat{y}(x) := \operatorname{argmax}_c w_c^T x, \quad (1)$$

where $W = (w_1, \dots, w_c) \in \mathbb{R}^{d \times c}$ are unknown parameters. The aim is to efficiently find good parameters in order to predict well on new data using Eq (1).

To address this problem setting, a number of generalizations of the binary SVM [17] have been proposed. We are specifically studying the following two formulations, dropping the bias terms in both. Throughout this paper, $l(x) = \max\{0, 1 - x\}$ will denote the hinge-loss.

Lee, Lin, and Wahba (LLW) [4]

$$\begin{aligned} \min_W \sum_{c=1}^C & \left[\frac{1}{2} \|w_c\|^2 + C \sum_{i:y_i \neq c} l(-w_c^T x_i) \right] \\ \text{s.t. } & \sum_c w_c = 0 \end{aligned} \quad (2)$$

Weston and Watkins (WW) [5]

$$\min_W \sum_{c=1}^C \left[\frac{1}{2} \|w_c\|^2 + C \sum_{i:y_i \neq c} l(w_{y_i}^T x_i - w_c^T x_i) \right] \quad (3)$$

Both formulations lead to very similar dual problems, as shown below. For the dualization of WW, we refer to [18]. The LLW dual is given below, where we introduce an auxiliary variable \bar{w} that is exploited by our distributed algorithm, as explained in the next section.

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^{n \times C}, \bar{w} \in \mathbb{R}^d} & \sum_{c=1}^C \left[-\frac{1}{2} \| -X\alpha_c + \bar{w} \|^2 + \sum_{i:y_i \neq c} \alpha_{i,c} \right] \\ \text{s.t. } & \forall i : \alpha_{i,y_i} = 0, \forall c \neq y_i : 0 \leq \alpha_{i,c} \leq C \end{aligned} \quad (4)$$

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^{n \times C}} & \sum_{c=1}^C \left[-\frac{1}{2} \| -X\alpha_c \|^2 + \sum_{i:y_i \neq c} \alpha_{i,c} \right] \\ \text{s.t. } & \forall i : \alpha_{i,y_i} = -\sum_{c \neq y_i} \alpha_{i,c}, \\ & \forall c \neq y_i : 0 \leq \alpha_{i,c} \leq C \end{aligned} \quad (5)$$

Derivation of lagrangian dual problems for Lin, Lee, and Wahba. Using slack variables, the primal LLW problem reads

$$\begin{aligned} \min_W \sum_{c=1}^C & \left[\frac{1}{2} \|w_c\|^2 + C \sum_{i:y_i \neq c} \xi_{i,c} \right] \\ \text{s.t. } & \sum_c w_c = 0 \\ \forall i : & \xi_{i,c} \geq 1 + w_c^T x_i \\ & \forall c \neq y_i : \xi_{i,c} \geq 0. \end{aligned} \quad (6)$$

We introduce Lagrangian multipliers $\alpha \in \mathbb{R}^{n \times C}$, $\beta \in \mathbb{R}^n$, and $\bar{w} \in \mathbb{R}^d$ with $\alpha_{i,c}, \beta_i \geq 0$.

$$L(W, \xi, \alpha, \beta, \bar{w}) = \sum_{c=1}^C \left[\frac{1}{2} \|\mathbf{w}_c\|^2 + \sum_{i:y_i \neq c} (C\xi_{i,c} + \alpha_{i,c}(1 + \mathbf{w}_c^T \mathbf{x}_i - \xi_{i,c}) - \beta_{i,c}\xi_{i,c}) \right] - \bar{w}^T \left(\sum_c \mathbf{w}_c \right) \quad (7)$$

Slater's condition holds; therefore, we have strong duality and can use the dual

$$\begin{aligned} \max_{\alpha, \beta, \bar{w}} \quad & \min_{W, \xi} L(W, \xi, \alpha, \beta, \bar{w}) \\ \text{s.t.} \quad & \forall i \forall c : \alpha_{i,c}, \beta_{i,c} \geq 0. \end{aligned}$$

The partial derivatives are given by

$$\begin{aligned} \frac{\partial}{\partial \xi_{i,c}} L(W, \xi, \alpha, \beta, \bar{w}) &= C - \alpha_{i,c} - \beta_{i,c} \\ \frac{\partial}{\partial \mathbf{w}_c} L(W, \xi, \alpha, \beta, \bar{w}) &= \mathbf{w}_c + \sum_{i:y_i \neq c} \alpha_{i,c} \mathbf{x}_i + \bar{w}. \end{aligned}$$

Setting those to zero leads to

$$\begin{aligned} \forall i \forall c : 0 &\leq \alpha_{i,c} \leq C \\ \mathbf{w}_c &= -\sum_{i:y_i \neq c} \alpha_{i,c} \mathbf{x}_i + \bar{w} \\ &= -X\alpha_c + \bar{w}. \end{aligned}$$

And plugging in into the lagrangian, finally gives the dual

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^{n \times C}, \bar{w} \in \mathbb{R}^d} \quad & \sum_{c=1}^C \left[-\frac{1}{2} \| -X\alpha_c + \bar{w} \|^2 + \sum_{i:y_i \neq c} \alpha_{i,c} \right] \\ \forall i : \quad & \alpha_{i,y_i} = 0 \\ \forall c \neq y_i : \quad & 0 \leq \alpha_{i,c} \leq C. \end{aligned} \quad (8)$$

Distributed algorithms

In this section, we derive algorithms that solve (LLW) and (WW) in a distributed manner. With start by addressing LLW.

Algorithm for Lee, Lin, and Wahba

First note the following optimality condition in (LLW):

$$\bar{w} = \frac{1}{C} \sum_{c=1}^C X\alpha_c.$$

Which was exploited by prevalent solvers to remove the variable \bar{w} from the optimization. In contrast, the core idea of our LLW solver is to actually keep this auxiliary variable, as it

decouples the objective function into the following sum:

$$\text{obj}(\alpha) = \sum_{c=1}^C \text{subobj}(\alpha_c, \bar{w}).$$

Then we perform dual block coordinate ascent (DBCA) [18, Algorithm 3.1] with a specially tailored block structure, considering as blocks \bar{w} as well as each single coordinate $\alpha_{i,c}$. As we observe from [Eq \(9\)](#), the optimization of the columns $\alpha_{:,c}$ is mutually independent of each other, given fixed \bar{w} . Hence, it can be distributed evenly over C many nodes. On the c th node, we run coordinate ascend on the subobjective $\text{subobj}(\alpha_c, \bar{w})$ over $\alpha_{i,c}$, $i = 1, \dots, n$, as described in the next paragraph. After one epoch of α computation, the variable \bar{w} is updated via [Eq \(9\)](#). The final algorithm is shown in Algorithm 1.

Algorithm 1 Lee, Lin, and Wahba

```

1: function SOLVE-LLW ( $C, X, Y$ )
2:   for  $c = 1..C$  do in parallel
3:      $w_c \leftarrow 0$ 
4:      $\alpha_c \leftarrow 0$ 
5:     for  $i \in \mathcal{I}$  do
6:        $k_i \leftarrow x_i^T x_i$ 
7:       while not optimal do
8:         optimal  $\leftarrow$  True
9:         shuffleData()
10:        for  $i \in \mathcal{I} \setminus \mathcal{I}_c$  do
11:          solve1DimLLW ( $i, c$ )
12:         $\bar{w} \leftarrow \text{Reduce}(\sum_c w_c / C)$ 
13:         $w_c \leftarrow w_c - \bar{w}$ 

```

As necessary step within Algorithm 1, we need to update every single $\alpha_{i,c}$. Optimizing $\alpha_{i,c}$ is solving the problem

$$\begin{aligned} & \underset{\delta}{\operatorname{argmax}} D(\alpha + \delta e_{i,c}, \bar{w}) \\ & \text{s.t. } 0 \leq \alpha_{i,c} + \delta \leq C, \end{aligned} \tag{9}$$

where $e_{i,c} \in \mathbb{R}^{n \times C}$ is one at the (i, c) th coordinate and zero elsewhere. Set $w_c := -X\alpha_c + \bar{w}$; then the gradient for δ is $\frac{\partial}{\partial \delta} [D(\alpha + \delta e_{i,c})] = x_i^T w_c - x_i^T x_i \delta + 1$. Hence, the optimal solution of [Eq \(9\)](#) is given by $\delta = \min \{C - \alpha_{i,c}, \max \{-\alpha_{i,c}, -\frac{x_i^T w_c - 1}{x_i^T x_i}\}\}$. The corresponding pseudo-code can be found in Algorithm 2.

Algorithm 2 Solving 1-dim sub-problem of LLW

```

1: function SOLVE1DIMLLW ( $i, c$ )
2:   global  $C, X, k, \alpha_c, w_c, \text{optimal}$ 
3:    $g \leftarrow w_c^T x_i - 1$ 
4:   if  $g < -\epsilon$  and  $\alpha_{i,c} < C$  then
5:      $\delta \leftarrow \min \{C - \alpha_{i,c}, -g / k_i\}$ 
6:      $\text{optimal} \leftarrow \text{False}$ 
7:   if  $g > \epsilon$  and  $\alpha_{i,c} > 0$  then
8:      $\delta \leftarrow \max \{-\alpha_{i,c}, -g / k_i\}$ 
9:      $\text{optimal} \leftarrow \text{False}$ 
10:     $w_c \leftarrow w_c + \delta x_i$ 
11:     $\alpha_{i,c} \leftarrow \alpha_{i,c} + \delta$ 

```

Convergence. It is well known that the block coordinate ascent method converges under suitable regularity conditions e.g. [19, 20]. Our objective is continuously differentiable and

strictly convex. The constraints are solely box constraints, hence the feasible set decomposes as a Cartesian product over the blocks. Algorithm 1 traverses the two blocks in cyclic order. Under these conditions, the DBCA method provably converges (e.g. Prop. 2.7.1 in [20]).

Note that in practice, we observed speedups by updating \bar{w} in Algorithm 1 after each tenth of an epoch, breaking the cyclic order. The blocks of coordinates are then traversed in so-called *essentially cyclic order* e.g. Section 2 in [19], meaning that there exists $T \in \mathbb{N}$ such that each block is traversed at least once after T iterations. Closer inspection of the proof in Prop. 2.7.1 in [20] reveals that the result holds also under this slightly more general assumption.

Further, we drop variables $\alpha_{i,c}$ in the optimization (shrinking) if they are not updated in three subsequent epochs. Once the stopping condition holds, we run another epoch of optimization over all variables (including the ones that were dropped). If the stopping criterion is then met, we terminate the algorithm and continue optimization elsewhere.

Implementation details. Our implementation uses OpenMPI for inter-machine (MPI) [21] and OpenMP (MC) [22] for intra-machine communication. Note that Algorithm 1 has very mild communication requirements: the only communication needed is the sum of all weight vectors $\bar{w} = \sum_c w_c$. Hence, MPI suffers very little from communication overhead between the various machines. In practice, we may not be able to fully parallelize to the maximum of \mathcal{C} cores; therefore our algorithm will divide the set of classes into number-of-cores many chunks and optimize each class sequentially.

Recall, \bar{d} is the average number of non-zero entries per sample and \bar{n} is the average number of samples per class. Given c cores and I iteration steps, the optimization has an asymptotic runtime estimate of $\mathcal{O}(I \cdot (\frac{\mathcal{C}}{c} \cdot \bar{n} \cdot \bar{d} + d \log_2 c))$, where the first part of the sum amounts for the gradient updates and the second for the model communication and update. Given an average dual sparsity as $1 - \bar{a}$ the algorithm needs at each node an asymptotic space complexity of $\mathcal{O}(\frac{\mathcal{C}}{c} \cdot (\bar{d} + \bar{n} \cdot \bar{a}) + d)$ to store the weight matrix, the dual coefficients and the weight vector used for averaging.

Algorithm for Weston and Watkins

In this section, we propose a distributed algorithm for WW, which draws inspiration from the 1-factorization problem of a graph. The approach is presented below.

Preliminaries. Our approach is based on running dual coordinate ascend, e.g. algorithm 3.1 in [18], over $\alpha_{i,c}$ on the (WW) objective function as follows. Denote the objective in Eq (5) by $D(\alpha)$ and recall from [18] that optimizing $\alpha_{i,c}$ is solving the problem

$$\begin{aligned} & \underset{\delta}{\operatorname{argmax}} D(\alpha + \delta \mathbf{e}_{i,c}) \\ & \text{s.t. } 0 \leq \alpha_{i,c} + \delta \leq C. \end{aligned} \quad (10)$$

Setting $w_c = \sum_{i:y_i \neq c} (-x_i \alpha_{i,c} + \sum_{c:c \neq y_i} x_i \alpha_{i,c})$, the gradient for δ is given by $\frac{\partial}{\partial \delta} [D(\alpha + \delta \mathbf{e}_{i,c})] = -x_i^T (w_{y_i} - w_c) - x_i^T x_i \delta + 1$. Which is optimal at:

$$\delta = \min \left(C - \alpha_{i,c}, \max \left(-\alpha_{i,c}, \frac{x_i^T (w_{y_i} - w_c) - 1}{2x_i^T x_i} \right) \right) \quad (11)$$

This computation is summarized in Algorithm 3.

Algorithm 3 Solving 1-dim sub-problem of WW

```

1: function SOLVE1DIMWW ( $i, c$ )
2:   global  $C, X, w_{y_i}, w_c, \alpha_c, \text{optimal}$ 
3:    $g \leftarrow (w_{y_i}^T - w_c^T) x_i - 1$ 

```

```

4:  if  $g < -\epsilon$  and  $\alpha_{i,c} < C$  then
5:     $\delta \leftarrow \min\{C - \alpha_{i,c}, -g/2k_i\}$ 
6:    optimal  $\leftarrow$  False
7:  if  $g > \epsilon$  and  $\alpha_{i,c} > 0$  then
8:     $\delta \leftarrow \max\{-\alpha_{i,c}, -g/2k_i\}$ 
9:    optimal  $\leftarrow$  False
10:    $w_{y_i} \leftarrow w_{y_i} + \delta x_i$ 
11:    $w_c \leftarrow w_c - \delta x_i$ 
12:    $\alpha_{i,c} \leftarrow \alpha_{i,c} + \delta$ 

```

Core observation. We observe from above that optimizing with regard to $\alpha_{i,c}$ will require only the weight vectors w_{y_i} and w_c . In other words, given four different classes c_1, c_2, c_3, c_4 , the optimization of the block of variables $(\alpha_i, c_1)_{i:y_i=c_2}$ —according to Eq (11)—is independent of the optimization of the block $(\alpha_i, c_3)_{i:y_i=c_4}$. Hence it can be parallelized. In the next section we describe how we exploit this structure to derive a distributed optimization algorithm.

Excursus: 1-factorization of a graph. Assume that \mathcal{C} is even. The core idea now is to form $\frac{\mathcal{C}}{2}$ many disjoint blocks $(\alpha_i, c_1)_{i:y_i=c_2}, \dots, (\alpha_i, c_{\mathcal{C}-1})_{i:y_i=c_{\mathcal{C}}}$ of variables. Each of these blocks can be optimized in parallel. The challenge is to derive a maximally distributed optimization schedule where each block $(\alpha_i, c_j)_{i:y_i=c_k}$ for any $j \neq k$ is optimized.

To better understand the problem, we consider the following analogy. We are given a football league with \mathcal{C} teams. Before the season, we have to decide on a schedule such that each team plays any other team exactly once. Furthermore, all teams shall play on every matchday so that in total we need only $\mathcal{C} - 1$ matchdays. This problem is the *1-factorization problem in graph theory*, e.g. [12]. The solution to this problem, illustrated in Fig 1, is as follows.

We arrange one node centrally and all other nodes in a regular polygon around the center node. At round r , we connect the centered node with node r and connect all other nodes orthogonal to this line. The pseudocode to compute the partner of a given node c at a certain round r is given in Algorithm 5. Note that in case of an uneven number of classes, we introduce a dummy class $\mathcal{C} + 1$, making the number of classes even. We run the algorithm, but skip all computations involving the dummy class.

Algorithm. The algorithm, shown in Algorithm 4, performs DBCA over the variables $\alpha_{i,c}$ using the schedule derived in Section and the coordinate updates derived in Section.

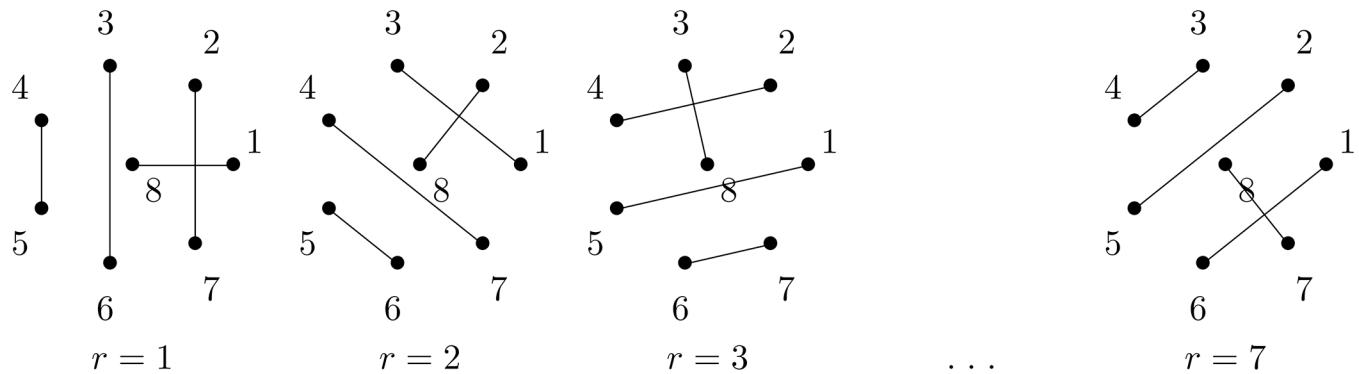


Fig 1. 1-factorization. Illustration of the solution of the 1-factorization problem of a graph with $\mathcal{C} = 8$ many nodes. Node 8 gets arranged centrally and at each step the pattern is rotated by one.

<https://doi.org/10.1371/journal.pone.0178161.g001>

Algorithm 4 Watkins-Weston

```

1: function SOLVE-WW( $c, X, Y$ )
2:   for  $c = 1..C$  do in parallel
3:      $w_c \leftarrow 0$ 
4:      $\alpha_c \leftarrow 0$ 
5:   for  $i \in I$  do
6:      $k_i \leftarrow x_i^T x_i$ 
7:   while not optimal do
8:     optimal  $\leftarrow$  True
9:     shuffleData()
10:    for  $r = 1..C - 1$  do
11:      for  $c = 1..C$  do in parallel
12:         $\tilde{c} \leftarrow \text{matchClass}(C, c, r)$ 
13:        if  $\tilde{c} > c$  then
14:          for  $i \in I_c$  do
15:            solve1DimWW( $i, \tilde{c}$ )
16:          for  $i \in I_{\tilde{c}}$  do
17:            solve1DimWW( $i, c$ )

```

Algorithm 5 Solving the graph 1-factorization problem. Indices start with one.

```

1: function MATCHCLASS( $C, c, r$ )
2:   if  $C$  is even and  $c = C$  then
3:     return  $r$ 
4:   if  $c = r$  then
5:     if  $C$  is even then
6:       return  $C$ 
7:     else
8:       return  $c$ 
9:   return mod( $2r - c, C - 1$ )

```

Convergence and implementation details. Furthermore, note that our algorithm performs the same coordinate updates as Algorithm 3.1 in [18]. Hence, they share the same favorable convergence behavior. Formally, convergence is guaranteed for exactly the same reasons discussed in Section. We also employ the same speedup tricks, i.e. shrinking and updating every tenth of an epoch.

In practice, because of limitations of computational resources, we might not be able to fully parallelize to the maximum of $C/2$ cores. In that case, our algorithm divides the set of classes into number-of-cores many chunks and solves each bundle sequentially. For optimal speedup, it is advisable to arrange the classes into chunks of equal number of classes and data points.

Given the average number of non-zero entries per sample \bar{d} and the average number of samples per class \bar{n} , c cores and I iteration steps, the optimization has an asymptotic runtime estimate of $\mathcal{O}(I \cdot C \cdot (\frac{C}{c} \cdot \bar{n} \cdot \bar{d} + d))$, where the first part of the sum amounts for the gradient updates and the second for the model communication. Given an average dual sparsity as $1 - \bar{a}$ the algorithm needs at each node an asymptotic space complexity of $\mathcal{O}(\frac{C}{c} \cdot (\bar{d} + \bar{n} \cdot \bar{a}))$ to store the weight matrix and the dual coefficients.

As with LLW, we implemented a mixed MPI-OpenMP solver for WW. However, note that, while LLW has mild communication needs, WW needs to pair the weight vectors of the matched classes c and \tilde{c} in each epoch, for which $C/2$ weight vectors needs to be communicated among computers. Therefore it is crucial to communicate efficiently.

We tackled the problem as follows. First of all, we use OpenMP for computations on a single machine (efficiently parallelizing among cores). Here, due to the shared memory, no

weight vectors need to be moved. The more challenging task is to handle inter-machine communication efficiently. Our approach is based on two key observations.

If the data is high-dimensional data, yet sparse, we keep the full weight matrix in memory for fast access, yet communicating only the non-zero entries between computers. Regardless of the increased computational effort, this takes only a fraction of time compared to sending the dense data.

Furthermore, we relax the WW matching scheme. Coming back to a football, consider each country hosts a league, and inside the league, we match the teams as known. Now we would like to match teams across leagues. In order to do so, we first match the countries with the scheme from Section. For each pair of countries, call them A and B, every team from country A plays any other team from country B. Coming back to classes and machines, this means we transfer bundles of classes (countries) between computers. This drastically reduces the network communication.

Experiments

This section is structured as follows. First we empirically verify the soundness of the proposed algorithms. Then we introduce the employed datasets, on which we investigate the convergence and runtime behavior of the proposed algorithms as well as the induced classification performance.

Each training algorithm was run three times, using randomly shuffled data, and the results were averaged. Note that the training set is the same in each run, but the different order of data points can impact the runtime of the algorithms.

Setup

For our experiments we use two different types of machines. Type A has 20 physical cpu cores, 128 GB of memory and a 10 GigaBit Ethernet network. Type B has 24 physical cpu cores and 386 GB of memory. On type B we ran the experiments involving CS due to the memory requirements.

Training repetitions were run on training sets with a random order of the data (note that the training set is the same in each run; only the order of points is shuffled, which can impact the DBCA algorithm). For *LIBLINEAR* solvers we use the newest available version as of April 2016 with the default settings.

We implemented our solvers using OpenMP, OpenMPI, and the Python-ecosystem. In more detail we used [23, 24], and [25].

Validation of solver

In our first experiment, we validate the correctness of the proposed solvers. We downloaded data from the *LIBLINEAR* (<https://www.csie.ntu.edu.tw/~cjlin/liblinear/>) [13] and UCI (<https://archive.ics.uci.edu/ml/datasets.html>) [26] dataset repositories. Where training and test splits are unavailable, we split the data once into 90% train and 10% test sets. For each dataset, the optimal feature scaling was selected, in order to maximize the average accuracy on the test sets. Datapoints in *iris* and *news* were thus normalized to unit norm, *letter* and *satimage* were normalized to unit variance. All other data was considered unnormalized.

Then we compare our LLW and WW solvers with the state-of-the-art implementation contained in the ML library Shark [27]. We implemented the same stopping criteria as [27]. The results (averaged over 10 runs) are shown in Table 1. We observe good accordance of the results and model sparsity of the proposed solvers and the reference implementation from the

Table 1. Comparison to existing solver.

Dataset:	D-LLW		S-LLW		D-WW		S-WW	
	Err.	Den.	Err.	Den.	Err.	Den.	Err.	Den.
SensIT (com.)								
log(C): -1	21.34	100.0	21.34	100.0	19.88	100.0	19.88	100.0
0	20.95	100.0	20.95	100.0	19.51	100.0	19.51	100.0
1	20.78	100.0	20.78	100.0	19.38	100.0	19.38	100.0
glass								
log(C): -1	66.67	100.0	66.67	100.0	38.10	100.0	38.10	100.0
0	61.90	100.0	61.90	100.0	19.05	100.0	19.05	100.0
1	33.33	100.0	33.33	100.0	19.05	100.0	19.05	100.0
iris								
log(C): -1	13.33	100.0	13.33	100.0	6.67	100.0	6.67	100.0
0	26.67	100.0	26.67	100.0	13.33	100.0	13.33	100.0
1	26.67	100.0	26.67	100.0	13.33	100.0	13.33	100.0
letter								
log(C): -1	87.04	100.0	87.04	100.0	28.25	100.0	28.26	100.0
0	87.24	100.0	87.24	100.0	29.04	100.0	29.03	100.0
1	61.91	100.0	87.24	100.0	28.92	100.0	28.93	100.0
news20								
log(C): -1	29.23	97.24	29.23	97.24	15.32	51.16	15.30	49.72
0	22.97	97.24	22.97	97.24	14.80	44.74	14.80	42.70
1	16.15	97.17	16.15	97.04	15.98	45.97	15.98	43.47
rcv1								
log(C): -1	47.96	78.00	47.96	78.00	11.31	26.42	11.31	23.45
0	33.27	78.00	33.41	77.98	11.52	22.93	11.52	20.12
1	12.03	78.00	12.03	77.98	12.03	23.05	12.03	20.06
satimage								
log(C): -1	26.75	100.0	26.73	100.0	15.80	100.0	15.80	100.0
0	26.80	100.0	26.80	100.0	15.47	100.0	15.53	100.0
1	26.90	100.0	26.90	100.0	15.96	100.0	16.00	100.0
splice								
log(C): -1	16.29	100.0	16.37	100.0	16.16	100.0	16.16	100.0
0	16.09	100.0	16.15	100.0	16.37	100.0	16.28	100.0
1	16.34	100.0	16.28	100.0	16.32	100.0	16.24	100.0
usps								
log(C): -1	31.84	100.0	31.84	100.0	8.17	100.0	8.17	100.0
0	30.09	100.0	30.04	100.0	9.37	100.0	9.37	100.0
1	28.00	100.0	28.00	100.0	10.51	100.0	10.51	100.0

Error on the test set and density in % of the Shark solver (denoted S) and the proposed solver (denoted D). The results across solver implementations show good accordance.

<https://doi.org/10.1371/journal.pone.0178161.t001>

Shark toolbox, thus confirming that our respective solvers are indeed exact solvers of LLW and WW.

At random we tested whether the duality-gap closes or not. We did this for both solvers with different C values and datasets. In any case the duality gap closed, i.e. decreased by an order of two magnitudes. Based on this we chose our stopping criteria ϵ equal to 0.1 for the LSHTC datasets.

Datasets

We experiment on large classification datasets, where the number of classes ranges between 451 and 27,875. The relevant statistics of the datasets are shown in [Table 2](#). The LSHTC-* datasets are high-dimensional text datasets taken from the well-known LSHTC corpus [14]. The datasets belong to the released competition rounds 1 to 3, i.e. '10-'12. LSHTC-2011 and LSHTC-2012 originate from the DMOZ corpus. The features were extracted using TF/IDF representation and we use the full feature resolution for training.

Speedup

In order to measure the speedup provided by increasing the number of machines/cores, we run a fix amount of iterations over the whole LSHTC-large dataset. We use 10 runs over 10 iterations with a fixed parameter C equal 1 without shrinking. While the MC execution works on one machine, the MPI executes on two or four machines, i.e. spreading the used cores evenly on each node.

The results are shown in [Fig 2](#). Both solvers exhibit linear speedup regardless if distributed or not, due to the small communication cost. Yet the speedup of WW is bounded by a larger constant compared to LLW.

Timing and classification results

Now we evaluate and compare the proposed algorithms on the LSHTC datasets for a range of C values, i.e. we perform no cross-validation. For comparison we use a solver from the well-known LIBLINEAR package, namely the multi-core implementation with L2L1-loss (OVR) [28]. For completeness we also include the single-core Crammer-Singer implementation (CS) [13]. Due to the lack of performance we do not compare to the LLW and WW implementation in the Shark ML library.

For the multi-core solvers, i.e. OVR and WW-MC, we use 16 cores. MPI spreads over 2 or 4 machines using 8 and 4 cores respectively at each node, *thus trains the model distributed*.

[Table 3](#) shows the error and the model sparsity for the compared solutions. We further provide the Micro-F1 and Macro-F1 score in [Table 4](#).

For all datasets the canonical multi-class formulations, i.e. CS and WW, perform significantly better than OVR. On one hand the error is smaller and the F1-scores better. On the other hand the learned models are much sparser, i.e. up to a magnitude. The results justify the increased solution complexity of canonical formulations.

Table 2. Dataset properties.

Dataset	n train	n test	C	d
LSHTC-small	4,463	1,858	1,139	51,033
LSHTC-large	128,710	34,880	12,294	381,581
LSHTC-2012	383,408	103,435	11,947	575,555
LSHTC-2011	394,754	104,263	27,875	594,158

The used datasets from the LSHTC-corpus and their properties. **n** train and **n** test denote the number of samples in the training and test set respectively, C the number of classes and **d** the number of dimensions. The most challenging dataset is given by LSHTC-2011. It contains the most samples, classes and dimensions.

<https://doi.org/10.1371/journal.pone.0178161.t002>

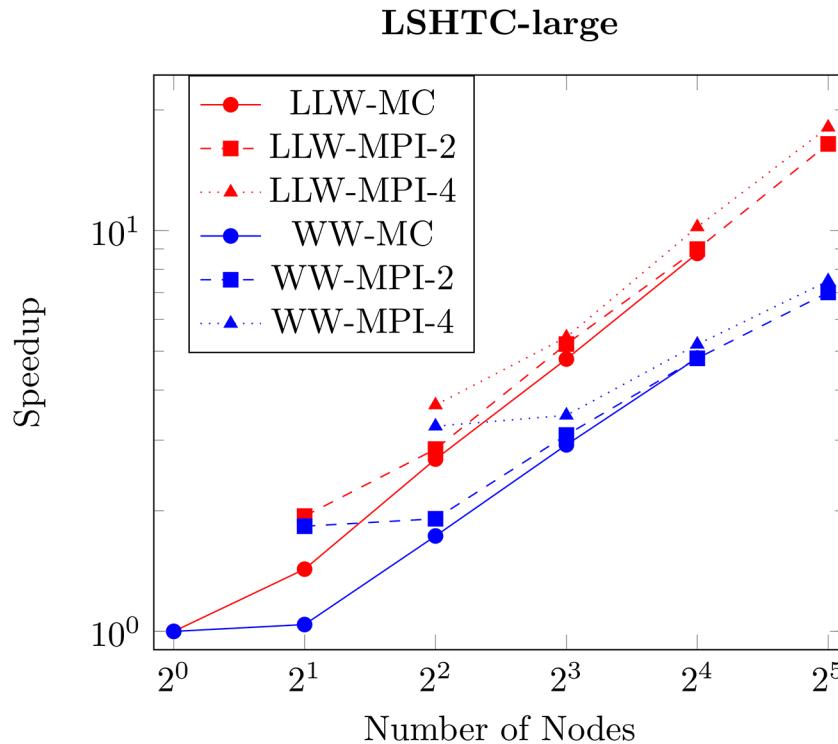


Fig 2. Speed-up. Speed-up of our solver averaged over 10 repetitions respectively in the number of cores. For * -MPI-2 and * -MPI-4 the number of cores is split evenly on 2 and 4 machines respectively. We observe a linear speedup in the number of cores for both solvers.

<https://doi.org/10.1371/journal.pone.0178161.g002>

Comparing CS and WW, CS performs as well or slightly better at classifying. Though WW leads to a sparser model. To the best of our knowledge this is the first comparison of these well-known multi-class SVMs on the studied LSHTC data.

From Fig 3, we observe that the runtime of our solver outperforms the one of OVR and CS by up to two orders of magnitude. Even when distributed our solver outperforms multi-core OVR in all except one case.

All WW experiments use the same amount of cores, but with a varying degree of distribution. We observe that the communication imposes a modest overhead. This overhead is influenced by the model density which is higher for smaller C values and effect of shrinking which is higher with larger C values. Yet in the regime with best classification results, e.g. C equal 0.1 and 1, the overhead is small.

Lin, Lee, & Wahba. Knowing that LLW converges to the correct solution, as the duality-gap closes, the results indicate that the chosen C range is not suitable. For LSHTC-small we conducted experiments with much larger C values. And indeed, as shown in Table 5, LLW performs best in a nearly unconstrained setting. In our experiments we observed that the model learned by LLW is never sparse, neither in the weight matrix W , nor in dual factors α . Resource limitations and slow convergence properties hindered us to conduct experiments with even larger C values. It is left to future work to explore this space or even develop a unconstrained version of LLW.

Table 3. Test error and model density.

Dataset:	Error				Model-Density			
	OVR	CS	WW	LLW	OVR	CS	WW	LLW
LSHTC-small								
log(C): -3	93.00	59.74	72.82	93.00	92.74	11.11	69.73	92.74
-2	85.36	59.74	65.34	93.00	81.54	11.13	16.44	92.74
-1	74.54	59.74	57.59	93.00	46.76	11.12	6.06	92.74
0	64.37	55.49	54.57	93.00	38.20	11.76	5.74	92.74
1	57.75	54.57	54.41	93.00	38.63	11.69	5.73	92.74
LSHTC-large								
log(C): -3	88.12	58.57	66.47	95.86	75.26	2.53	18.50	100.0
-2	85.21	58.57	60.58	95.86	45.14	2.53	4.45	100.0
-1	77.96	57.82	55.28	95.86	25.28	2.55	1.71	100.0
0	63.11	53.61	53.98	95.86	18.33	2.69	1.61	100.0
1	57.18	54.18	54.41	*	18.55	2.67	1.66	*
LSHTC-2012								
log(C): -3	83.66	49.81	58.02	92.63	72.60	1.73	16.97	99.52
-2	75.15	49.65	50.20	92.63	46.20	1.71	4.06	99.52
-1	60.38	46.14	44.94	92.63	25.87	1.76	1.52	99.52
0	47.33	42.67	44.01	*	18.20	2.06	1.42	*
1	46.83	45.60	46.15	*	18.46	2.09	1.47	*
LSHTC-2011								
log(C): -3	87.95	59.09	68.19	96.18	72.38	1.57	13.49	100.0
-2	85.85	59.09	62.14	96.18	45.97	1.57	3.16	100.0
-1	76.78	58.18	57.31	96.18	25.97	1.55	1.19	100.0
0	63.11	55.58	56.94	*	18.24	1.69	1.11	*
1	60.01	57.78	58.32	*	18.46	1.70	1.14	*

Test set error and model density (in %) as achieved by the OVR, CS, WW, and LLW solvers on the LSHTC datasets. For each solver the result with the best error is in bold font. For LLW entries with a '*' did not converge within a day of runtime.

<https://doi.org/10.1371/journal.pone.0178161.t003>

Discussion of related work

Most approaches to parallelization of MCSVM training are based on OVO or OVR [29], including a number of approaches that attempt to learn a hierarchy of labels [30–36] or train ensembles of SVMs on individual subsets of the data [37–39].

There is a line of research on parallelizing stochastic gradient (SGD) based training of MC-SVMs over multiple computers [40, 41]. SGD builds on iteratively approximating the loss term by one that is based on a subset of the data (mini-batch). In contrast, batch solvers (such as the ones proposed in the present paper) are based on the full sample. In this sense, our approach is completely different to SGD. While there is a long ongoing discussion whether the batch or the SGD approach is superior, the common opinion is that SGD has its advantages in the early phase of the optimization, while classical batch solvers shine in the later phase. In this sense, the two approaches are complementary and could also be combined.

The related work that is the most closest to the present work is by [42]. They build on the alternating direction method of multipliers (ADMM) [43] to break the Crammer and Singer optimization problem into smaller parts, which can be solved individually on different computers. In contrast to our approach, the optimization problem is parallelized over the samples, not the optimization variables. In our problem setting, high-dimensional sparse data, the

Table 4. F1-Scores.

Dataset:	Micro-F1				Macro-F1			
	OVR	CS	WW	LLW	OVR	CS	WW	LLW
LSHTC-small								
$\log(C): -3$	7.00	40.26	27.18	7.00	0.61	22.08	10.73	0.61
-2	14.42	40.26	34.66	7.00	2.70	22.08	16.15	0.61
-1	25.46	40.26	42.41	7.00	8.72	22.08	24.71	0.61
0	35.47	44.46	45.43	7.00	16.42	26.70	28.75	0.61
1	42.41	45.48	45.59	7.00	25.09	28.73	29.15	0.61
LSHTC-large								
$\log(C): -3$	11.77	41.35	33.53	4.14	0.88	25.43	15.05	0.09
-2	14.80	41.52	39.42	4.14	1.51	25.41	20.83	0.09
-1	22.02	42.19	44.72	4.14	3.35	25.83	27.90	0.09
0	36.86	46.41	46.02	*	14.76	30.99	31.29	*
1	42.80	45.83	45.59	*	25.87	31.13	31.12	*
LSHTC-2012								
$\log(C): -3$	16.34	50.19	41.98	7.37	0.28	20.55	8.08	0.01
-2	24.85	50.35	49.80	7.37	0.69	20.72	16.17	0.01
-1	39.62	53.86	55.06	7.37	2.64	23.76	25.94	0.01
0	52.67	57.33	55.99	*	12.46	32.57	32.06	*
1	53.17	54.40	53.85	*	24.41	31.84	30.95	*
LSHTC-2011								
$\log(C): -3$	12.05	40.91	31.81	3.82	0.46	22.44	10.47	0.05
-2	14.15	40.91	37.86	3.82	0.62	22.46	16.48	0.05
-1	23.22	41.82	42.69	3.82	1.89	23.37	23.17	0.05
0	36.89	44.42	43.06	*	10.60	26.97	27.25	*
1	39.99	42.22	41.86	*	21.30	26.31	26.97	*

Micro-F1 and Macro-F1 scores (in %) as achieved by the OVR, CS, WW, and LLW solvers on the LSHTC datasets. For each solver and each metric the best result across C values is in bold font. For LLW entries with a '*' did not converge within a day of runtime.

<https://doi.org/10.1371/journal.pone.0178161.t004>

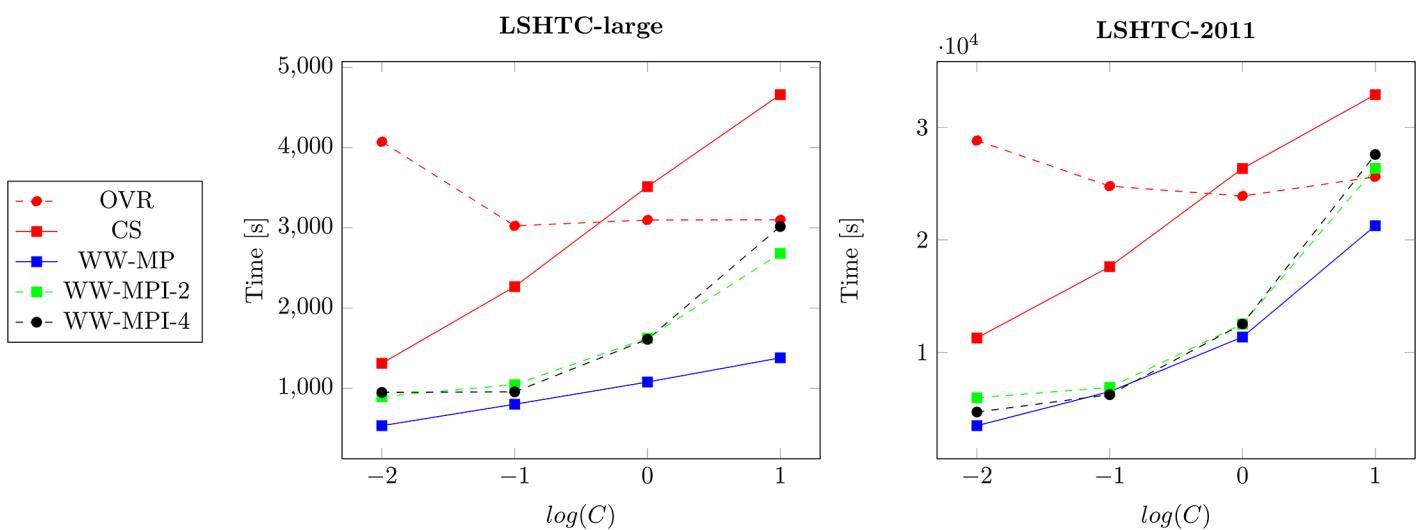


Fig 3. Training times. Training time averaged over 10 repetitions per C for the various solvers.

<https://doi.org/10.1371/journal.pone.0178161.g003>

Table 5. Results for LLW-Solver.

log(C):	2	3	4
Error:	87.73	66.74	59.31
Micro F1:	2.08	15.07	40.69
Macro F1:	12.27	33.26	24.58
W-Density:	92.74	92.74	92.74
α-Density:	99.88	99.87	99.90

Error, Micro-F1, and Macro-F1 on the test set and model density in % of the LLW solver on the LSHTC-small dataset.

<https://doi.org/10.1371/journal.pone.0178161.t005>

model size is vary large. Because each node holds the whole model in memory, this algorithm hardly scales with large label spaces. E.g. consider Table 2; the model for LSHTC-2011 contains $\approx 16 * 10^9$ parameters. Note also that it is unclear at this point whether the approach of [42] could be adapted to LLW and WW, which are the object of study in the present paper.

Note that beyond SVMs there is a large body of work on distributed multi-class [44, 45] and multi-label learning algorithms [46], which is outside of the scope of the present paper.

When the performance of single solvers saturates one can consider to refine them, e.g. by considering the reliability of single class estimates [47] to enhance the prediction accuracy, or to learn ensembles of SVMs to get better prediction by combining several estimates. This can be done by bagging [48] or boosting [49] them or by exploiting smart compositions of solvers, e.g. with LibD3C [50].

Conclusion

We proposed distributed algorithms for solving the multi-class SVM formulations by Lee et al. (LLW) and Weston and Watkins (WW). The algorithm addressing LLW takes advantage of an auxiliary variable, while our approach to optimizing WW in parallel is based on the 1-factorization problem from graph theory.

The experiments confirmed the correctness of the solver (in the sense of an exact solver) and show linear speedup when the number of cores is increased. This speedup allows us to train LLW and WW on LSHTC datasets, for which results were lacking in the literature.

Our analysis contributed to comparing MC-SVM formulations on rather large data sets, where comparisons were still lacking. In comparison to OVR we showed that WW can achieve competitive classification results in less time, while still leading to a much sparser model. Unexpectedly, LLW shows clear disadvantages over the other MC-SVMs. Yet the favorable scaling properties make further research interesting, for instance regarding the development of an unconstrained algorithm. We ease further research by publishing the source code under <https://github.com/albermax/xcsvm>.

Overcoming the limitations of a single machine, i.e. distribution, is a key problem and a key enabler in large scale learning. To best of our knowledge, we are the *first to train an exact, all-in-one MC-SVMs in a distributed manner*. We hope this first step inspires further research in this context.

In the future, we would like to study extensions of the concepts presented in this paper to various more MC-SVMs, including the Crammer and Singer MC-SVM [51], the l_p -norm MC-SVM [52], and scatter based MC-SVMs [53].

Acknowledgments

We thank Rohit Babbar, Shinichi Nakajima, and Klaus-Robert Müller for helpful discussions. We thank Giancarlo Kerg for pointing us to the graph 1-factorization problem. We thank Ioannis Partalas for help regarding the LSHTC datasets.

Author Contributions

Conceptualization: MA JZ UD MK.

Data curation: MA JZ.

Methodology: MA JZ MK.

Software: MA.

Writing – original draft: MA JZ MK.

Writing – review & editing: MA MK.

References

1. Vapnik V. *Statistical Learning Theory*. John Wiley and Sons; 1998.
2. Rifkin R, Klautau A. In defense of one-vs-all classification. *Journal of Machine Learning Research*. 2004; 5:101–141.
3. Crammer K, Singer Y. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*. 2002; 2:265–292.
4. Lee Y, Lin Y, Wahba G. Multicategory Support Vector Machines: Theory and Application to the Classification of Microarray Data and Satellite Radiance Data. *Journal of the American Statistical Association*. 2004; 99(465):67–82. <https://doi.org/10.1198/016214504000000098>
5. Weston J, Watkins C. Support vector machines for multi-class pattern recognition. In: Verleysen M, editor. *Proceedings of the Seventh European Symposium On Artificial Neural Networks (ESANN)*. Evere, Belgium: d-side publications; 1999. p. 219–224.
6. Allwein EL, Schapire RE, Singer Y. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*. 2001; 1:113–141.
7. Hsu CW, Lin CJ. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*. 2002; 13(2):415–425. <https://doi.org/10.1109/72.991427> PMID: 18244442
8. Hill SI, Doucet A. A framework for kernel-based multi-category classification. *Journal of Artificial Intelligence Research*. 2007; 30(1):525–564.
9. Liu Y. Fisher consistency of multicategory support vector machines. In: Meila M, Shen X, editors. *Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS)*. vol. 2 of *JMLR W&P*; 2007. p. 289–296.
10. Guermeur Y. VC Theory for Large Margin Multi-Category Classifiers. *Journal of Machine Learning Research*. 2007; 8:2551–2594.
11. Doğan Ü, Glasmachers T, Igel C. A Unified View on Multi-class Support Vector Classification. *Journal of Machine Learning Research*. 2016; 17(45):1–32.
12. Bondy JA, Murty USR. *Graph theory with applications*. vol. 290. Macmillan London; 1976.
13. Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*. 2008; 9:1871–1874.
14. Partalas I, Kosmopoulos A, Baskiotis N, Artières T, Palioras G, Gaussier É, et al. LSHTC: A Benchmark for Large-Scale Text Classification. *CoRR*. 2015;abs/1503.08581.
15. Szedmak S, Shawe-Taylor J, Parado-Hernandez E. Learning via linear operators: Maximum margin regression. *PASCAL*, Southampton, UK; 2006.
16. Liu Y, Yuan M. Reinforced Multicategory Support Vector Machines. *Journal of Computational and Graphical Statistics*. 2011; 20(4):901–919. <https://doi.org/10.1198/jcgs.2010.09206>
17. Cortes C, Vapnik V. Support-vector networks. *Machine Learning*. 1995; 20(3):273–297. <https://doi.org/10.1007/BF00994018>

18. Keerthi SS, Sundararajan S, Chang KW, Hsieh CJ, Lin CJ. A Sequential Dual Method for Large Scale Multi-class Linear Svms. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD'08. New York, NY, USA: ACM; 2008. p. 408–416. Available from: <http://doi.acm.org/10.1145/1401890.1401942>
19. Tseng P. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*. 2001; 109(3):475–494. <https://doi.org/10.1023/A:1017501703105>
20. Bertsekas DP, Homer ML, Logan DA, Patek SD. *Nonlinear programming*. Athena scientific. 1995;.
21. Gropp W, Lusk E, Doss N, Skjellum A. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing*. 1996; 22(6):789–828. [https://doi.org/10.1016/0167-8191\(96\)00024-5](https://doi.org/10.1016/0167-8191(96)00024-5)
22. Dagum L, Enon R. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering*. IEEE. 1998; 5(1):46–55. <https://doi.org/10.1109/99.660313>
23. Van Der Walt S, Colbert SC, Varoquaux G. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*. 2011; 13(2):22–30. <https://doi.org/10.1109/MCSE.2011.37>
24. Behnel S, Bradshaw R, Citro C, Dalcin L, Seljebotn DS, Smith K. Cython: The best of both worlds. *Computing in Science & Engineering*. 2011; 13(2):31–39. <https://doi.org/10.1109/MCSE.2010.118>
25. Dalcin LD, Paz RR, Kler PA, Cosimo A. Parallel distributed computing using python. *Advances in Water Resources*. 2011; 34(9):1124–1139. <https://doi.org/10.1016/j.advwatres.2011.04.013>
26. Asuncion A, Newman D. UCI machine learning repository; 2007.
27. Igel C, Glasmachers T, Heidrich-Meisner V. *Shark*. *Journal of Machine Learning Research*. 2008; 9:993–996.
28. Chiang WL, Lee MC, Lin CJ. Parallel Dual Coordinate Descent Method for Large-scale Linear Classification in Multi-core Environments. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD'16. New York, NY, USA: ACM; 2016. p. 1485–1494. Available from: <http://doi.acm.org/10.1145/2939672.2939826>
29. Babbar R, Maundet K, Schölkopf B. TerseSVM: A Scalable Approach for Learning Compact Models in Large-scale Classification. In: Proceedings of the 2016 SIAM International Conference on Data Mining. SIAM; 2016. p. 234–242.
30. Bengio S, Weston J, Grangier D. Label embedding trees for large multi-class tasks. In: *Advances in Neural Information Processing Systems*; 2010. p. 163–171.
31. Deng J, Satheesh S, Berg AC, Li F. Fast and balanced: Efficient label tree learning for large scale object recognition. In: *Advances in NIPS*; 2011. p. 567–575.
32. Gao T, Koller D. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In: *ICCV*. IEEE; 2011. p. 2072–2079.
33. Choromanska AE, Langford J. Logarithmic Time Online Multiclass prediction. In: *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc.; 2015. p. 55–63.
34. Zhou D, Xiao L, Wu M. Hierarchical classification via orthogonal transfer. In: *ICML*; 2011. p. 801–808.
35. Gopal S, Yang Y. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In: *ACM SIGKDD*. ACM; 2013. p. 257–265.
36. Madzarov G, Gjorgjevikj D, Chorbev I. A Multi-class SVM Classifier Utilizing Binary Decision Tree. *Informatica (Slovenia)*. 2009; 33(2):225–233.
37. Govada A, Gauri B, Sahay SK. Distributed Multi Class SVM for Large Data Sets. In: *Proceedings of the Third International Symposium on Women in Computing and Informatics*. WCI'15. New York NY, USA: ACM; 2015. p. 54–58. Available from: <http://doi.acm.org/10.1145/2791405.2791534>
38. Govada A, Ranjani S, Viswanathan A, Sahay S. A Novel Approach to Distributed Multi-Class SVM. *arXiv preprint arXiv:151201993*. 2015;.
39. Lodi S, Nanculef R, Sartori C. Single-pass distributed learning of multi-class svms using core-sets. *methods*. 2010; 14(27):2.
40. Gupta MR, Bengio S, Weston J. Training highly multiclass classifiers. *Journal of Machine Learning Research*. 2014; 15(1):1461–1492.
41. Do TN. Parallel multiclass stochastic gradient descent algorithms for classifying million images with very-high-dimensional signatures into thousands classes. *Vietnam Journal of Computer Science*. 2014; 1(2):107–115. <https://doi.org/10.1007/s40595-013-0013-2>
42. Han X, Berg AC. DCMSVM: Distributed parallel training for single-machine multiclass classifiers. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE; 2012. p. 3554–3561.

43. Boyd S, Parikh N, Chu E, Peleato B, Eckstein J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*. 2011; 3(1):1–122. <https://doi.org/10.1561/2200000016>
44. Agarwal A, Chapelle O, Dudík M, Langford J. A Reliable Effective Terascale Linear Learning System. *CoRR*. 2011;abs/1110.4198.
45. Gopal S, Yang Y. Distributed training of Large-scale Logistic models. In: *ICML (2)*; 2013. p. 289–297.
46. Prabhu Y, Varma M. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM; 2014. p. 263–272.
47. Liu Y, Zheng YF. One-against-all multi-class SVM classification using reliability measures. In: *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*. vol. 2. IEEE; 2005. p. 849–854.
48. Breiman L. Bagging predictors. *Machine learning*. 1996; 24(2):123–140. <https://doi.org/10.1007/BF00058655>
49. Freund Y, Schapire R. A desicion-theoretic generalization of on-line learning and an application to boosting. In: *Computational Learning Theory*. Springer; 1995. p. 23–37.
50. Lin C, Chen W, Qiu C, Wu Y, Krishnan S, Zou Q. LibD3C: ensemble classifiers with a clustering and dynamic selection strategy. *Neurocomputing*. 2014; 123:424–435. <https://doi.org/10.1016/j.neucom.2013.08.004>
51. Crammer K, Singer Y. On the learnability and design of output codes for multiclass problems. *Machine Learning*. 2002; 47(2):201–233. <https://doi.org/10.1023/A:1013637720281>
52. Lei Y, Dogan U, Binder A, Kloft M. Multi-class svms: From tighter data-dependent generalization bounds to novel algorithms. In: *Advances in Neural Information Processing Systems*; 2015. p. 2035–2043.
53. Jenssen R, Kloft M, Zien A, Sonnenburg S, Müller KR. A scatter-based prototype framework and multi-class extension of support vector machines. *PloS one*. 2012; 7(10):e42947. <https://doi.org/10.1371/journal.pone.0042947> PMID: 23118845