# Distributed Optimization of Multi-Class SVMs

**Extended Abstract Submission for NIPS '16 XMC Workshop**

**Maximilian Alber** [*]
Berlin Big Data Center
Berlin Institute of Technology
10587 Berlin, Germany
maximilian.alber@tu-berlin.de

**Julian Zimmert** [*]
Department of Computer Science
Humboldt University of Berlin
10099 Berlin, Germany
zimmerjy@informatik.hu-berlin.de

**Urun Dogan**
Microsoft Research
Cambridge CB1 2FB, UK
urundogan@gmail.com

**Marius Kloft**
Department of Computer Science
Humboldt University of Berlin
10099 Berlin, Germany
kloft@hu-berlin.de

## Abstract

While training the one-vs.-rest SVM can be parallelized over the number of classes in a straight forward way, the so-called *all-in-one* multi-class SVMs (Crammer and Singer, Weston and Watkins, Lee et al.) are still mostly non applicable for extreme classification.

We work towards distributed algorithms for all-in-one MC-SVMs, starting – by the present paper – with the formulation by Weston and Watkins (WW), for which we propose a distributed solver that can be parallelized over the number of classes, with decent computational overhead.

## 1 Introduction

Extreme classification problems 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. The one-vs.-one (OVO) and one-vs.-rest (OVR) machines, which decompose the problem into multiple binary subproblems that are subsequently aggregated [1, 2]. Training can be distributed in a straight forward way [3, 4].
2. The *all-in-one* approaches, which extend the concept of the margin to multiple classes. Because there is no unique extension of the margin concept, multiple MC-SVM variants have been proposed, including the ones by Crammer and Singer (CS) [5], Lee, Lin, and Wahba (LLW) [6], and Weston and Watkins (WW) [7, 1]. See [2, 8–13] for conceptual and empirical comparisons.

So far, slow training time has prohibited comparing the various all-in-one MC-SVM variants in the large number of label domains. 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 space 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 the simplistic OVO and OVR SVMs as the only MC-SVM options in the big data setting.

We want to work toward comparing the various all-in-one MC-SVM variants in the large number of classes domain. In the present paper, we start by addressing the MC-SVM by *Weston and Watkins*

[*]Shared first co-authorship, sorted alphabetically.

Workshop on Extreme Classification at 29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

(WW) [7, 1], which has been shown in [13] to outperform LLW and CS. We propose a distributed algorithm for WW, where up to $\mathcal{O}(\mathcal{C})$ machines solve an all-in-one MC-SVM problem in parallel, while dividing the model evenly over the machines.

Our algorithm draws inspiration from a major result in graph theory: the solution to the 1-factorization problem of a graph [14]. The core insight that lets us distribute the coordinate ascent training of the *WW SVM* is that the optimization of a single coordinate $\alpha_{i,c}$ of the dual objective involves only the two hypotheses $w_{y_i}$ and $w_c$. We can thus form pairs of classes where the corresponding blocks of coordinates can be optimized in parallel.

We report on empirical runtime comparisons of the proposed solver with the state of the art on standard benchmark data from various domains as well as massive text classification data taken from the LSHTC corpus [3]. To our knowledge, we are the first to train WW on datasets from the LSHTC corpus [3] using the full feature resolution.

The paper is structured as follows. In the next section, we the discuss problem setting and preliminaries. In Section 3, we present the proposed distributed algorith for WW. We analyze its convergence empirically in Section 4. Section 5 concludes.

## 2 Preliminaries

We consider the following problem. We are given data $(x_1, y_1), \ldots, (x_n, y_n)$ with $x_i \in \mathbb{R}^d$ and $y_i \in \{1, ..., \mathcal{C}\}$. 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

$$\hat{y}(x) := \underset{c}{\operatorname{argmax}}\, w_c^T x, \tag{1}$$

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

To address this problem setting, a number of generalizations of the binary SVM [15] have been proposed. We are specifically studying the following formulation, dropping the bias terms. Throughout this paper, $l(x) = \max\{0, 1 - x\}$ will denote the hinge-loss.

**Weston and Watkins (WW) [7]**

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

The dualization reads as follows:

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

## 3 Algorithm

In this section, we derive an algorithm that solves (WW) in a distributed manner, using dual block coordinate ascent (DBCA). Our solver runs dual coordinate ascent as proposed in [16, Algorithm 3.1]. Our contribution is that – by a trick to be explained below – we are able to develop a scheme that distributes the computations.

### 3.1 Preliminaries

We recapitulate [16, Algorithm 3.1] as Algorithm 1 in the table to the right. The core idea of the algorithm is to optimizes one dual variable $\alpha_{i,c}$ at a time as follows.

**Algorithm 1** Solving 1-dim WW sub-problem induced by a single dual variable $\alpha_{i,c}$.

```
1:  function SOLVE1DIMWW(i,c)
2:      global $C, X, w_{y_i}, w_c, \alpha_c$, 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 ← False
7:      if $g > \epsilon$ and $\alpha_{i,c} > 0$ then
8:          $\delta \leftarrow \max\{-\alpha_{i,c}, -g/2k_i\}$
9:          optimal ← False
10:     $w_{y_i} \leftarrow w_{y_i} + \delta x_i$
11:     $w_c \leftarrow w_c - \delta x_i$
12:     $\alpha_{i,c} \leftarrow \alpha_{i,c} + \delta$
```

Denote the objective in WW by $D(\alpha)$ and recall from [16] that optimizing $\alpha_{i,c}$ is solving the problem

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

Setting

$$w_c = \sum_{i:y_i \neq c} \left( -x_i \alpha_{i,c} + \sum_{c:c \neq y_i} x_i \alpha_{i,c} \right),$$

the gradient for $\delta$ 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

$$\delta = \min\{C - \alpha_{i,c}, \max\{-\alpha_{i,c}, \frac{x_i^T(w_{y_i} - w_c) - 1}{2x_i^T x_i}\}\}. \tag{3}$$

### 3.1.1 Core Observation

We observe in the algorithm described 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 (3)—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.

## 3.2 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 $(\alpha_i, c_1)_{i:y_i=c_2}, \ldots, (\alpha_i, c_{\mathcal{C}-1})_{i:y_i=\mathcal{C}}$ 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., 14]. The solution to this problem, illustrated in Figure 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. In case of an uneven number of classes, we skip the middle node and optimize it in round r $r$. The pseudocode to compute the partner of a given node $c$ at a certain round $r$ is given in Algorithm 3 in the appendix.

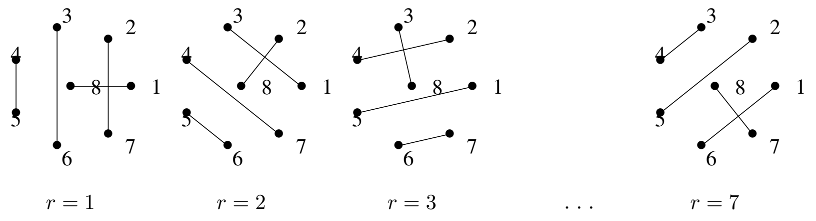

Figure 1: Illustration of the solution of the 1-factorization problem of a graph with $\mathcal{C} = 8$ many nodes.

### 3.3 Algorithm

We are now ready to formulate our algorithm to solve WW. The algorithm, shown in Algorithm 2, performs DCA over the variables $\alpha_{i,c}$ using a schedule inspired from the 1-factorization problem in graph theory and the coordinate updates derived in Section 3.1.

**Algorithm 2** Watkins-Weston

```
 1: function SOLVE-WW(c,X,Y)
 2:     for $c = 1..\mathcal{C}$ 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 ← True
 9:         shuffleData()
10:         for $r = 1..\mathcal{C} - 1$ do
11:             for $c = 1..\mathcal{C}$ do in parallel
12:                 $\tilde{c} \leftarrow$ matchClass($\mathcal{C}, c, r$)
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($i, c$)
```

#### 3.3.1 Convergence

It is well know that the block coordinate ascent method converges under suitable regularity conditions [e.g., 17, 18]. 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 2 traverses the two blocks in cyclic order. Under these conditions, the DBCA method provably converges [e.g, 18, Prop. 2.7.1]. Note that in practice, we observed speedups by updating $\bar{w}$ in Algorithm 2 after each tenth of an epoche, breaking the cyclic order. Further, we drop samples from the training set of a class once they are not updated three times in a row. Once the stopping condition holds we start again with the full set for each class. *The final run must fulfill the stopping condition traversing all samples.* The blocks of coordinates are then traversed in so-called *essentially cyclic order* [e.g., 17, Section 2], 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 [e.g, 18, Prop. 2.7.1] reveals that the result holds also under this slightly more general assumption.

## 4 Experiments

This section is structured as follows. First we empirically show the soundness of our algorithms. We present the used datasets and show the convergence and speedup properties of the proposed solutions. Finally, we analyze the classification and time performance on the LSHTC datasets.

For the experimental setup we refer to Appendix A. Repetitions were trained 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 DCA algorithm).

We compare our WW solvers with the state-of-the-art implementation contained in the ML library Shark [19] on a series of small standard benchmarks. We observe good accordance of the results and model sparsity of the proposed solvers and the reference implementation from the Shark toolbox, thus confirming that our respective solver is indeed an exact solver of the WW MC-SVM problem. The results can be found in Table C.1 in the supplement.

### 4.1 Datasets

| Dataset | # Training | # Test | # Classes | # Features |
|---|---|---|---|---|
| **LSHTC1-small** | 4,463 | 1,858 | 1,139 | 51,033 |
| **LSHTC1-large** | 128,710 | 34,880 | 12,294 | 381,581 |
| **LSHTC3** | 383,408 | 103,435 | 11,947 | 575,555 |
| **LSHTC2** | 394,754 | 104,263 | 27,875 | 594,158 |

Table 1: The used datasets and their properties.

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 1. The LSHTC-* datasets are high-dimensional text datasets taken from the well-known LSHTC corpus [3]. The datasets correspond respectively to the released competition rounds. LSHTC2 and LSHTC3 originate from the DMOZ corpus. The features were extracted using TF/IDF representation and we use the full feature resolution for training.

## 4.2 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 LSHTC1-small and LSHTC1-large dataset. We use 10 runs over 10 iterations with a fixed parameter C equal 1. The results are shown in Figure 2. We observe a near linear speedup until 8 nodes. The plot indicates a sublinear (rootish) speedup afterwards, but because of computational constraint we were not (yet) able to assess this properly.

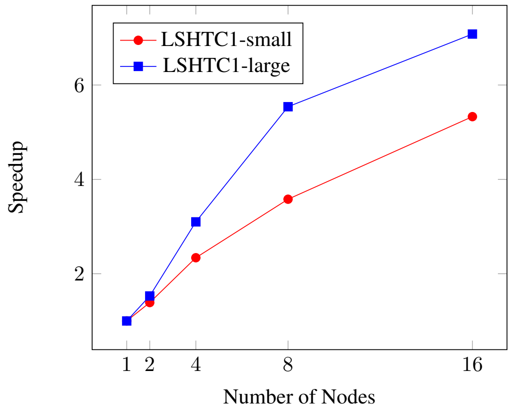

Figure 2: Speed-up averaged over 3 repetitions in the number of cores.

## 4.3 Timing and Classification Results

Now we evaluate and compare our solution on the LSHTC datasets for a range of C values, i.e. we perform no cross-validation. For comparison we use solvers from the well-known *LIBLINEAR* package, namely the multi-core implementation with L2L1-loss (OVR, [20]) and the Crammer-Singer implementation (CS, [21]). For the multi-core solvers we use 16 cores. Table 2 shows the error and the model sparsity for the compared solutions.

| | **OVR** | | **CS** | | **WW** | |
|---|---|---|---|---|---|---|
| **Dataset:** | Err. | Spar. | Err. | Spar. | Err. | Spar. |
| **LSHTC1-small** | | | | | | |
| log$(C)$: *-3* | 93.00 | 92.74 | 59.74 | 11.11 | 72.82 | 69.73 |
| *-2* | 85.36 | 81.54 | 59.74 | 11.13 | 65.34 | 16.44 |
| *-1* | 74.54 | 46.76 | 59.74 | 11.12 | 57.59 | 6.06 |
| *0* | 64.37 | 38.20 | 55.49 | 11.76 | 54.57 | 5.74 |
| *1* | 57.75 | 38.63 | 54.57 | 11.69 | 54.41 | 5.73 |
| **LSHTC1-large** | | | | | | |
| log$(C)$: *-3* | 88.12 | 75.26 | 58.57 | 2.53 | 66.47 | 18.50 |
| *-2* | 85.21 | 45.14 | 58.57 | 2.53 | 60.58 | 4.45 |
| *-1* | 77.96 | 25.28 | 57.82 | 2.55 | 55.28 | 1.71 |
| *0* | 63.11 | 18.33 | 53.61 | 2.69 | 53.98 | 1.61 |
| *1* | 57.18 | 18.55 | 54.18 | 2.67 | 54.41 | 1.66 |
| **LSHTC3** | | | | | | |
| log$(C)$: *-3* | 83.66 | 72.60 | 49.81 | 1.73 | 58.02 | 16.97 |
| *-2* | 75.15 | 46.20 | 49.65 | 1.71 | 50.20 | 4.06 |
| *-1* | 60.38 | 25.87 | 46.14 | 1.76 | 44.94 | 1.52 |
| *0* | 47.33 | 18.20 | 42.67 | 2.06 | 44.01 | 1.42 |
| *1* | 46.83 | 18.46 | 45.60 | 2.09 | 46.15 | 1.47 |
| **LSHTC2** | | | | | | |
| log$(C)$: *-3* | 87.95 | 72.38 | 59.09 | 1.57 | 68.19 | 13.49 |
| *-2* | 85.85 | 45.97 | 59.09 | 1.57 | 62.14 | 3.16 |
| *-1* | 76.78 | 25.97 | 58.18 | 1.55 | 57.31 | 1.19 |
| *0* | 63.11 | 18.24 | 55.58 | 1.69 | 56.94 | 1.11 |
| *1* | 60.01 | 18.46 | 57.78 | 1.70 | 58.32 | 1.14 |

Table 2: Test set error and model sparsity (in %) as achieved by the OVR, CS, and WW solvers on the LSHTC datasets. The values of the regularization parameter $C$ are shown on $\log_{10}$ scale.

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. 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.

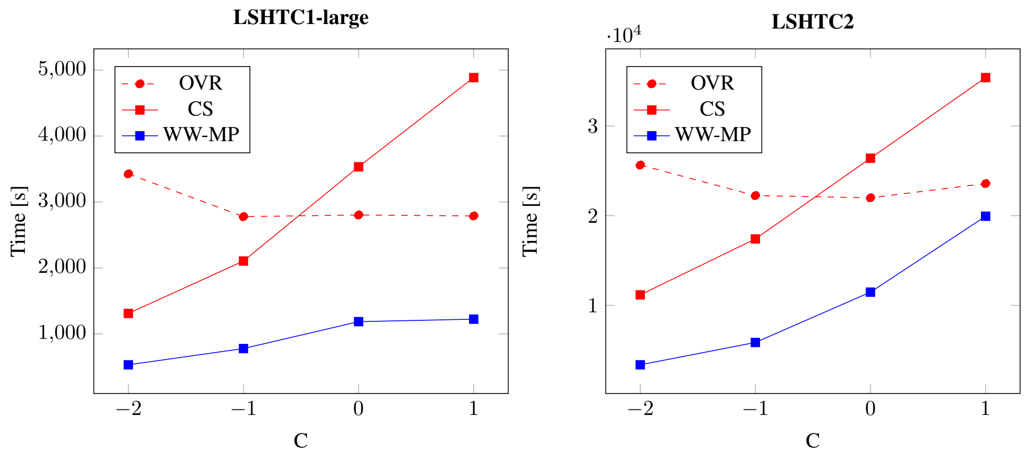

Figure 3: Training time averaged over 3 repetitions per C.

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 Figure 3, we observe that the runtime of our solver outperforms the one of OVR and CS by up to two orders of magnitude.

## 5 Conclusion

Based on DCA and the 1-factorization of a graph problem, we proposed an provably converging algorithm for solving the WW MC-SVM, where the computation is distributed over the number of classes. The experiments confirmed the correctness of the solver (in the sense of an exact solver) and show speedup when the number of cores is increased. This speedup allowed us to train WW on LSHTC datasets, for which results were lacking in the literature.

In the future, we want to experiment with distributing the solver not only on different cores but different machines. Further, we would like to push our agenda forward to parralize also the MC-SVMs by Crammer and Singer [22], Lee at al. [6], multi-class maximum margin regression [23], and the reinforced multicategory SVM[24].

### Acknowledgments

We thank Rohit Babbar 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. 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.

## References

[1] V. Vapnik, *Statistical Learning Theory*. John Wiley and Sons, 1998.

[2] R. Rifkin and A. Klautau, “In defense of one-vs-all classification,” *Journal of Machine Learning Research*, vol. 5, pp. 101–141, 2004.

[3] I. Partalas, A. Kosmopoulos, N. Baskiotis, T. Artières, G. Paliouras, É. Gaussier, I. Androutsopoulos, M. Amini, and P. Gallinari, “LSHTC: A benchmark for large-scale text classification,” *CoRR*, vol. abs/1503.08581, 2015.

[4] G. Balikas, A. Kosmopoulos, A. Krithara, G. Paliouras, and I. Kakadiaris, “Results of the bioasq tasks of the question answering lab at clef 2015,” in *CLEF 2015*, 2015.

[5] K. Crammer and Y. Singer, “On the algorithmic implementation of multiclass kernel-based vector machines,” *Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2002.

[6] Y. Lee, Y. Lin, and G. Wahba, “Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data,” *Journal of the American Statistical Association*, vol. 99, no. 465, pp. 67–82, 2004.

[7] J. Weston and C. Watkins, “Support vector machines for multi-class pattern recognition,” in *Proceedings of the Seventh European Symposium On Artificial Neural Networks (ESANN)* (M. Verleysen, ed.), pp. 219–224, Evere, Belgium: d-side publications, 1999.

[8] E. L. Allwein, R. E. Schapire, and Y. Singer, “Reducing multiclass to binary: A unifying approach for margin classifiers,” *Journal of Machine Learning Research*, vol. 1, pp. 113–141, 2001.

[9] C. W. Hsu and C. J. Lin, “A comparison of methods for multiclass support vector machines,” *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[10] S. I. Hill and A. Doucet, “A framework for kernel-based multi-category classification,” *Journal of Artificial Intelligence Research*, vol. 30, no. 1, pp. 525–564, 2007.

[11] Y. Liu, “Fisher consistency of multicategory support vector machines,” in *Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS)* (M. Meila and X. Shen, eds.), vol. 2 of *JMLR W&P*, pp. 289–296, 2007.

[12] Y. Guermeur, “VC theory for large margin multi-category classifiers,” *Journal of Machine Learning Research*, vol. 8, pp. 2551–2594, 2007.

[13] Ü. Doğan, T. Glasmachers, and C. Igel, “A Unified View on Multi-class Support Vector Classification,” *Journal of Machine Learning Research*, vol. 17, no. 45, pp. 1–32, 2016.

[14] J. A. Bondy and U. S. R. Murty, *Graph theory with applications*, vol. 290. Macmillan London, 1976.

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

[16] S. S. Keerthi, S. Sundararajan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, “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), pp. 408–416, ACM, 2008.

[17] P. Tseng, “Convergence of a block coordinate descent method for nondifferentiable minimization,” *Journal of optimization theory and applications*, vol. 109, no. 3, pp. 475–494, 2001.

[18] D. P. Bertsekas, M. L. Homer, D. A. Logan, and S. D. Patek, “Nonlinear programming,” *Athena scientific*, 1995.

[19] C. Igel, T. Glasmachers, and V. Heidrich-Meisner, “Shark,” *Journal of Machine Learning Research*, vol. 9, pp. 993–996, 2008.

[20] W.-L. Chiang, M.-C. Lee, and C.-J. Lin, “Parallel dual coordinate descent method for large-scale linear classification in multi-core environments,”

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

[22] K. Crammer and Y. Singer, “On the learnability and design of output codes for multiclass problems,” *Machine Learning*, vol. 47, no. 2, pp. 201–233, 2002.

[23] S. Szedmak, J. Shawe-Taylor, and E. Parado-Hernandez, “Learning via linear operators: Maximum margin regression,” tech. rep., PASCAL, Southampton, UK, 2006.

[24] Y. Liu and M. Yuan, “Reinforced multicategory support vector machines,” *Journal of Computational and Graphical Statistics*, vol. 20, no. 4, pp. 901–919, 2011.

[25] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: a structure for efficient numerical computation,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[26] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, “Cython: The best of both worlds,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011.

[27] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, “Parallel distributed computing using python,” *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011.

# Appendix

## A Experimental 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 DCA algorithm). For *LIBLINEAR* solvers we use the newest available version as of April 2016 with the default settings.

We implemented our solveres using OpenMP, OpenMPI, and the Python-ecosystem. In more detailed we used [25], [26], and [27].

## B 1-factorization of a Graph Algorithm

The algorithm table below complements the description of the matching algorithm in Section 3.2 of the main text.

**Algorithm 3** Solving the graph 1-factorization problem. Indices start with one.

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

## C Validation Experiments

The following table shows the results of a validation experiment, where we compare the solution achieved by our proposed solver with reference implementations from ML library shark [19].

| | **D-WW** | | **S-WW** | |
|---|---|---|---|---|
| **Dataset:** | Err. | Spar. | Err. | Spar. |
| **combined** | | | | |
| log$(C)$: *-1* | 19.88 | 100.0 | 19.88 | 100.0 |
| *0* | 19.51 | 100.0 | 19.51 | 100.0 |
| *1* | 19.38 | 100.0 | 19.38 | 100.0 |
| **glass** | | | | |
| log$(C)$: *-1* | 38.10 | 100.0 | 38.10 | 100.0 |
| *0* | 19.05 | 100.0 | 19.05 | 100.0 |
| *1* | 19.05 | 100.0 | 19.05 | 100.0 |
| **iris** | | | | |
| log$(C)$: *-1* | 6.67 | 100.0 | 6.67 | 100.0 |
| *0* | 13.33 | 100.0 | 13.33 | 100.0 |
| *1* | 13.33 | 100.0 | 13.33 | 100.0 |
| **letter** | | | | |
| log$(C)$: *-1* | 28.25 | 100.0 | 28.26 | 100.0 |
| *0* | 29.04 | 100.0 | 29.03 | 100.0 |
| *1* | 28.92 | 100.0 | 28.93 | 100.0 |
| **news20** | | | | |
| log$(C)$: *-1* | 15.32 | 51.16 | 15.30 | 49.72 |
| *0* | 14.80 | 44.74 | 14.80 | 42.70 |
| *1* | 15.98 | 45.97 | 15.98 | 43.47 |
| **rcv1** | | | | |
| log$(C)$: *-1* | 11.31 | 26.42 | 11.31 | 23.45 |
| *0* | 11.52 | 22.93 | 11.52 | 20.12 |
| *1* | 12.03 | 23.05 | 12.03 | 20.06 |
| **satimage** | | | | |
| log$(C)$: *-1* | 15.80 | 100.0 | 15.80 | 100.0 |
| *0* | 15.47 | 100.0 | 15.53 | 100.0 |
| *1* | 15.96 | 100.0 | 16.00 | 100.0 |
| **splice** | | | | |
| log$(C)$: *-1* | 16.16 | 100.0 | 16.16 | 100.0 |
| *0* | 16.37 | 100.0 | 16.28 | 100.0 |
| *1* | 16.32 | 100.0 | 16.24 | 100.0 |
| **usps** | | | | |
| log$(C)$: *-1* | 8.17 | 100.0 | 8.17 | 100.0 |
| *0* | 9.37 | 100.0 | 9.37 | 100.0 |
| *1* | 10.51 | 100.0 | 10.51 | 100.0 |

Table C.1: Comparison of the Shark solver (denoted S) and the proposed solver (denoted D) in terms of test set error and model sparsity (in %) on standard benchmark data.