Faster Support Vector Machines*

Sebastian Schlag† Matthias Schmitt‡ Christian Schulz§

Abstract

The time complexity of support vector machines (SVMs) prohibits training on huge data sets with millions of samples. Recently, multilevel approaches to train SVMs have been developed to allow for time efficient training on huge data sets. While regular SVMs perform the entire training in one - time consuming - optimization step, multilevel SVMs first build a hierarchy of problems decreasing in size that resemble the original problem and then train an SVM model for each hierarchy level benefiting from the solved models of previous levels. We present a faster multilevel support vector machine that uses a label propagation algorithm to construct the problem hierarchy. Extensive experiments show that our new algorithm achieves speed-ups of up to two orders of magnitude while having similar or better classification quality over state-of-the-art algorithms.

1 Introduction

Machine learning is an important subfield of computer science that builds and studies algorithms which are able to learn from and to understand the vast amounts of data that are available today in order to make predictions. A concrete machine learning task is the classification problem. In a classification problem, we are given unlabeled data points, e.g. information about the financial situation of a person, and want to put them into the right class out of a finite number of classes, e.g credit-worthy or not credit-worthy. Such tasks were historically done by experts in the specific field and required lots of time and man power. We can easily see why a bank would like to automate the process of checking for credit-worthiness because a machine could do this job faster, more cost-efficiently, and hopefully less error prone. With recent advances in machine learning, we now have algorithms that are able to do the work previously thought of being an exclusive competence of humans. The training is done by presenting the algorithm with labeled example data points from which it has to learn the underlying structure such that it is able to correctly classify unlabeled data afterwards.

Large margin classifiers are one approach to tackle this problem. These classifiers separate the classes of the classification problem in such a way that they are able to give the distance to a decision boundary. Support Vector Machines (SVMs) [10] are the most well-known large margin classifiers. They solve a convex optimization problem and use a maximum-margin hyperplane to separate classes. SVMs are known to show good performance when trained to solve a classification problem, but in order to achieve high quality predictions *model selection* is also required. Model selection is the process of finding the right parameters for a specific problem. This is the work-intensive part of machine learning with SVMs. Since the time complexity for solving the optimization problem underlying SVMs is between $O(n^2)$ and $O(n^3)$ [16] and every set of parameters requires the training of a new SVM model, SVM training becomes a problem on data sets that have hundreds of thousands or even millions of training points.

Model selection itself is highly parallelizable as different model parameters can be evaluated independently in parallel, but for large data sets the time complexity to solve the underlying optimization problem for a single set of parameters is still infeasible. Other model selection approaches for SVM make some training results reusable [8] so that the training time complexity shrinks but parallelization is more difficult. Even highly optimized SVM algorithms can not cope with data sets of hundreds of thousands of data points.

In practice, huge data sets can be imbalanced, i.e., classes have unequal size. An example are medical data sets where labels tell whether a person is ill or healthy. A data set like this is often imbalanced due to the fact that illnesses occur less often when the general population is considered. Different machine learning algorithms apply varying techniques to train SVMs on such data sets.

One possibility to cope with large data sets is random sampling [42]. Here, a random subset of the input data is selected and used for training. However,

*This work was partially supported by DFG SA 933/11-1 and the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

†Institute for Theoretical Informatics, Karlsruhe Institute of Technology, Karlsruhe, Germany.

‡Karlsruhe Institute of Technology, Karlsruhe, Germany.

§Faculty of Computer Science, University of Vienna, Vienna, Austria.

problems can arise when infrequently occurring important data are underrepresented in the sample. Because random sampling only reflects the distribution of training data one may miss significant regions of the testing data [48]. Hence, to train a SVM on large problems a more sophisticated approach is needed.

A new promising research path to tackle the scalability problem was recently introduced to SVMs: *the multilevel paradigm*. Already widely used in, e.g. graph partitioning [5], a multilevel framework first builds a hierarchy of the problem. Each level of the hierarchy is a problem that decreases in size but reflects the structure of the original problem. Then a regular SVM is trained on the coarsest problem. This is more feasible than training on the original input because the problem is much smaller. The training is then projected upwards in the hierarchy, i.e., the model is refined to better fit the original problem while only gradually increasing the size of the data the SVM is trained on. Overall, experiments indicate that the multilevel paradigm reduces computation time while being comparable and often better than non-hierarchical approaches in terms of prediction quality on large data sets.

Contribution and Outline. We present a faster SVM that also uses the multilevel paradigm. In contrast to previous approaches, we use a clustering-based contraction scheme that is able to shrink the input size very quickly. More precisely, we apply a near-linear time label propagation clustering algorithm to compute a clustering and then contract the clustering. Extensive experiments indicate that our approach is up to orders of magnitude faster than the previous best algorithm while having comparable classification quality.

2 Preliminaries

2.1 Basic Concepts. Let $G = (V = \{0, \ldots, n-1\}, E, \omega)$ be an *undirected graph* with edge weights $\omega : E \to \mathbb{R}_{>0}$, $n = |V|$, and $m = |E|$. We extend ω to sets, i.e., $\omega(E') := \sum_{e \in E'} \omega(e)$. $N(v) := \{u : \{v, u\} \in E\}$ denotes the *neighbors* of v. Given a set of points $P = \{p_1, \ldots, p_m\}$, with $p_i \in \mathbb{R}^d$, the *k-nearest neighbor graph* has a vertex for every point and connects two vertices p, q by an edge if the distance between them is among the k-th smallest distances from p to other points in P. An *independent set* $I \subset V$ in a graph G is a set of vertices such that no two vertices are adjacent. A *clustering* is a partition of the nodes, i.e., *blocks* of nodes $V_1, \ldots, V_k \subset V$ such that $V_1 \cup \cdots \cup V_k = V$ and $V_i \cap V_j = \emptyset$. However, k is usually not given in advance. A *size-constrained clustering* constrains the size of the blocks by a given upper bound U such that $|V_i| \leq U$. For example, when using $U = 1$, the only feasible size-constrained clustering in an unweighted graph is the singleton clustering, where each node forms a block on its own.

2.2 Classification. A common supervised learning task in machine learning is the classification problem, i.e., to associate unlabeled data points with a specific class out of a finite number of classes. We do so by training an algorithm on a set of training examples $x_1, \ldots, x_n$ with associated labels $y_1, \ldots, y_n$. Once the algorithm is trained on the training set, we predict the class/label y_{n+1} for a new data point x_{n+1}. A special case of classification is binary classification, where there are only two classes.

2.3 Support Vector Machines. Support vector machines (SVMs) are supervised learning models that can be used for classification. SVMs are one of the most well-known machine learning algorithms [40]. They are large margin classifiers that find a hyperplane to decide the class for a new data point.

Given a set $\mathcal{I}$ of n data points x_i with corresponding labels $y_i \in \{+1, -1\}$, the minority class $\mathbf{C}^+$ consists of all data points with positive label ($|\mathbf{C}^+| = n^+$). All other data points are in the majority class $\mathbf{C}^-$ ($|\mathbf{C}^-| = n^-$, $n = n^+ + n^-$). Throughout this paper we assume w.l.o.g. $n^+ \leq n^-$. Every training data point x_i is interpreted as a d dimensional vector in $\mathbb{R}^d$. The SVM finds a $d-1$ dimensional hyperplane separating the two classes. The *best separating hyperplane* is the one furthest away from both classes, i.e., the one with the largest margin between the two classes, hence the name large margin classifier.

However, often the data is not linear separable in the Euclidean space. Hence in practice one uses a mapping of the data points to a higher dimensional space $\phi : \mathbb{R}^d \to \mathbb{R}^p$ ($d \leq p$) in order to make two classes separable by a hyperplane. This is also known as the kernel trick, which was originally proposed by Aizerman et al. [1] and applied to SVMs by Boser et al. [4]. In this paper, we use the Gaussian kernel (radial basis function, RBF), $k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) = \phi(x_i)^T \phi(x_j)$. This function is known to be reliable when no additional assumptions about the data are known. The standard SVM formulation is then given by the following constrained optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(w \cdot \phi(x_i) - b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

The hyperplane $f(x) = w \cdot \phi(x) + b$ with maximum margin is calculated by solving for the system and obtaining optimal parameters w and b. The slack variables

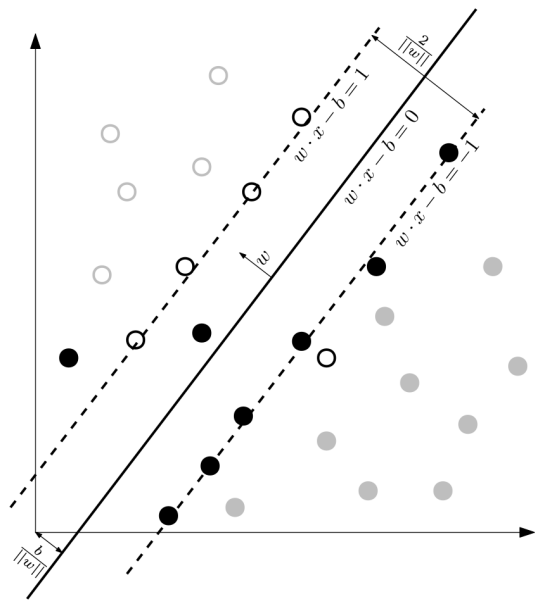

Figure 1: A non-separable binary classification problem solved with SVM and slack variables. Non-support vectors are gray shaded. The mapping ϕ is assumed to be the identity in this example.

$\xi_i = \max(0, 1 - y_i(w \cdot \phi(x_i) - b))$ are used to extend SVMs to cases where data is not linearly separable, meaning there is no hyperplane separating $\mathbf{C}^+$ and $\mathbf{C}^-$. This allows for misclassification but ensures that every x_i lies on the correct side of the margin. Since it is part of the term that is to minimize, it also penalizes misclassification. This method is known as soft margin extension [10] to the SVM. The parameter $C > 0$ controls the magnitude of the penalization.

Once the vectors w and b have been found, new data points are classified by the sign of the hyperplane equation $h(x) = \text{sign}(w \cdot \phi(x) + b)$. In simpler terms the hyperplane splits the input space into two and puts data points above the plane in $\mathbf{C}^+$ and below in $\mathbf{C}^-$.

A non-linearly separable example where slack variables are important and allow the SVM to find a separating hyperplane is shown in Fig. 1. We see that $\frac{2}{\|w\|}$ is the distance between the margins and that in order to find the maximum separating hyperplane we need to minimize $\|w\|$. An easy-to-see but important consequence by the way the hyperplane is calculated is that the maximum margin hyperplane is only determined by those x_i which lie nearest to it, e.g. the data points whose removal would result in a change of the hyperplane, known as *support vectors* (SVs).

Model Selection. Prior to training an SVM model, the model parameters have to be chosen. In the case that is the main focus of the paper there are two parameters: the penalty parameter C and the kernel parameter γ of the Gaussian kernel. The optimization problem with fixed parameters is convex [10]. The model parameters are highly instance dependent. Hence, parameter fitting is required to get optimal or near optimal parameters for a concrete instance. The parameter search is also called *model selection*. This is a general problem not only found in the context of SVMs.

Multi-class Classification. Our focus in this paper is on binary classification problems, but the concepts can be extended to support multi-class classification where the y_i are not restricted to $\{+1, -1\}$ by allowing for multiple labels directly [11] or training independent one-versus-rest binary SVMs [27].

3 Related Work

There has been a *huge* amount of research on machine learning and support vector machines so that we refer the reader to existing literature [35, 41, 44] for most of the material. Here, we focus on issues closely related to our main contributions and previous work on the problem.

There have been many approaches that attempt to improve the performance of SVMs, e.g. [15, 23, 32]. LibSVM [6] is a widely used solver that implements the sequential minimal optimization algorithm. LibLINEAR [14] is a library that performs well on data where a non-linear SVM is not required, but is typically not feasible for complex data or data with a large amount of imbalance. Another attempt to improve the performance of SVMs is parallelization [7, 12, 16, 26, 28, 45–47]. In particular, there are methods that use parallel interior point methods to solve the underlying optimization problem [7, 28]. The approach by Li et al. [26] is also a parallel interior point method, but additionally uses GPUs. Other parallelizations use parallel stochastic gradient decent methods [2, 33] or try to avoid communication [12, 16, 46].

Another successful approach to improve the performance of SVMs are hierarchical techniques. The general multilevel paradigm originated from multigrid solvers for solving systems of linear equations. Currently, there are two types of that scheme, i.e., early approaches that work in the feature space and later approaches that build a graph representation of the data first and then work with that representation to build a problem hierarchy. The most general multilevel approach for SVMs consists of three main phases: training set coarsening, coarsest support vector learning, and support vector refinement. In the *contraction* (coarsening) phase, one iteratively decreases the problem size by performing contractions. Contraction should quickly reduce the size of the input and each computed level

should reflect the global structure of the input. Contraction is stopped when the problem is small enough so that a model can be trained by some other potentially more expensive algorithm. This is also called *initial training* phase. In the *local search* (or uncoarsening) phase, contraction is iteratively undone, model parameters are inherited and at each level a local improvement algorithm is used to improve the quality of the result. In current multilevel SVMs the support vectors and optionally their neighbors are uncontracted and used to train the SVM again using insights from the parameter search of the initial training.

Yu et al. [48] were among the first to explore hierarchical SVM techniques. The authors create a hierarchically clustered representation of the data by merging data points based on distance. However, only linear classifiers have been considered and model parameters are neither inherited nor improved throughout the hierarchy. Later, non-linear kernels have also been considered [18] in an advanced intrusion detection system. Hsieh et al. [19] also present an hierarchical approach that uses a different geometric clustering scheme.

Instead of using the feature space representation as previous works [18, 19, 48], Razzaghi and Safro [37] use a graph representation. Their algorithm starts by building two graphs, one for each classification class, using approximate k-nearest neighbors. The multilevel algorithm to train the SVM then uses this graph representation, i.e., the k-nearest neighbor graphs are gradually coarsened by performing contractions based on independent sets. Moreover, the algorithm is the first to include *refinement*, improving the initial trained model throughout the hierarchy from coarsest to finest. The support vectors of the previous coarser level of the hierarchy are used to train on the current level. This approach is less sensitive to imbalanced data. This is due to the fact that on the coarsest level the data classes are of roughly equal size because the coarsening uses the same size-constraint for both classes individually. This helps the initial SVM algorithm since the coarsest level is not imbalanced anymore but still resembles the imbalanced original problem. The approach is substantially faster with no loss of quality in the performance measures, when compared the underlying SVM solver LibSVM [37]. Sadrfaridpour et al. [38, 39] improved the result by utilizing an algebraic multigrid (AMG) multilevel scheme, i.e., instead of independent set contractions an AMG algorithm is used in the coarsening phase.

DC-SVM [19] is a multilevel divide-and-conquer SVM that uses adaptive clustering. It is one of the fastest SVMs, however, the approach by Sadrfaridpour et al. [38, 39] demonstrates significantly better running time than DC-SVM on almost all data sets [40].

Algorithm 1 Overview

```
preprocess data
build k-nearest neighbor graph for C+ and C−
contract graphs recursively, build hierarchy
initial training on coarsest problem
while levels in the hierarchy do
    train SVM model on uncontracted support vectors
    of previous level
return best model of all levels
```

4 Scalable SVMs by Cluster Contraction

We now give a full description of our algorithm. Note that our algorithm is similar to the one of Razzaghi and Safro [37], i.e., we transfer the problem to a graph problem and then use a graph-based multilevel algorithm to solve the underlying SVM optimization problem. Our algorithm, however, uses a much simpler and more efficient coarsening strategy that is able to shrink the problem size very quickly and thus is much faster while giving similar classification results.

4.1 Overview. An overview of our algorithm is shown in Algorithm 1. Our algorithm starts by preprocessing the data and afterwards splitting it into the two given classes $\mathbf{C}^+$ and $\mathbf{C}^-$. We then build *two* k-nearest neighbor graphs, one for each class. The rest of the algorithm is a multilevel approach that works on those two graphs. Our algorithm starts coarsening by computing coarse versions of each of the two graphs independently using a *clustering contraction* scheme. More precisely, we use a label propagation algorithm to compute a clustering in each graph and then contract the clusters. We continue to do this recursively. For each of the two graphs, coarsening is stopped once the associated graph has a predefined size. Note that since the coarsening process is done independently on each of the graphs, the coarsest graphs are roughly balanced in the number of vertices they contain. When coarsening has been stopped on both graphs, we perform *initial training* using the coarsest level of the computed hierarchies, i.e., we solve the SVM optimization problem on the coarsest graphs of that hierarchy. Once the initial model is trained we recursively uncontract the next finer hierarchy level and train a new SVM model on the support vectors of the previous model. After the finest level is processed and we trained models for every level of the hierarchy, we choose the overall best model as the final model of the complete multilevel process. We now give full details.

4.2 Preprocessing Data. Our algorithm starts with two commonly used preprocessing routines: categorical feature preprocessing and feature scaling.

Categorical Features. SVMs require data to be represented by real valued numbers. Often times features such as the country someone lives in or the marital status are given as categorical variables. Categorical means that the feature can take one of a limited and fixed number of possible values. For categorical attributes an extra preprocessing step is needed to convert them into numerical data. We do so by using one hot encoding [17]. An ℓ-category attribute is represented by ℓ numbers where one of the ℓ numbers is one and the others are zero. If the number of values in an attribute is not too large, this coding is more stable than using a single number [20].

Scaling. The purpose of scaling is to avoid that attributes in greater numeric ranges dominate those in smaller numeric ranges [20]. The problem can be avoided by scaling the feature columns independently. Our algorithm uses the standardization technique [43]. Here each of the feature columns is transformed by subtracting the mean and then dividing by the standard deviation such that all feature columns have zero-mean and unit-variance.

4.3 Graph Model. We transform the training problem into a graph problem by using the technique of Razzaghi and Safro [37]. More precisely, we compute two graphs: one for each of the classes $\mathbf{C}^+$ and $\mathbf{C}^-$. For a class, the graph is constructed by applying an approximate k-nearest neighbor algorithm on the data. Thus we obtain a graph with information about the *proximity* of every vertex. This information is also incorporated into the graph, i.e., the weight of an edge is set to $1/\text{dist}(p, q)$ for two points that are adjacent in the graph, where $\text{dist}(\cdot, \cdot)$ is the Euclidean distance. We use the AkNN library FLANN [31] to perform this task.

4.4 Coarsening by Cluster-Contraction. Our algorithm computes coarse versions of each of the two graphs independently using a *clustering contraction* scheme. To compute a clustering, we use a variation of the *label propagation* algorithm proposed by Raghavan et al. [36], which is a very fast, near linear-time algorithm that locally optimizes the number of edges cut. Initially, each node is in its own cluster/block, i.e., the initial block ID of a node is set to its node ID. The algorithm then works in rounds.

In each round, the original algorithm traverses the nodes of the graph in random order. However, previous work [29] has shown that using the ordering induced by the node degree (increasing order) improves the overall solution quality *and* running time. Using this node ordering means that in the first round of the label propagation algorithm, nodes with small node degree can change their cluster before nodes with a large node degree. We use this node ordering in our algorithm.

When a node v is visited, it is *moved* to the cluster that has the strongest connection to v, i.e., it is moved to the cluster V_i that maximizes $\omega(\{(v, u) \mid u \in N(v) \cap V_i\})$. Ties are broken randomly. Originally, the process is repeated until it has converged. We perform at most ℓ rounds of the algorithm instead, where ℓ is a tuning parameter. One round can be implemented to run in linear time. Note that due to the way weights are defined in our graphs, i.e., edge weights are anti-proportional to the distance of vertices, our algorithm finds clusters of vertices that are *close* to each other.

We also tried a variant of the label propagation algorithm that, in contrast to the original algorithm [36], ensures that each block of the clustering fulfills a size constraint. However, we observed that this slows down the overall algorithm and had almost no effect on the observed classification quality.

To compute a graph hierarchy, the clustering is contracted by replacing each cluster/block by a single node, and the process is repeated recursively until the graph is small. This way the inherent cluster hierarchy of the networks is detected and the contraction of important edges in small cuts is unlikely. For the original problem this means that we contract groups of data points that are close into one single data point. Note that cluster contraction is an aggressive coarsening strategy. In contrast to most previous approaches, it can drastically shrink the size of irregular networks. Experiments in [29] indicate that already one contraction step can shrink the graph size by orders of magnitude.

Contracting a clustering works as follows: each block of the clustering is contracted into a single node. Note that each node has an associated feature vector. The feature vector of the coarse representative is computed as follows: for each feature column, the value of the feature of the coarse representative is set to the arithmetic mean of the values of the feature of all nodes that are contracted into that node. There is an edge between two nodes u and v in the contracted graph if the two corresponding blocks in the clustering are adjacent to each other in G, i.e., block u and block v are connected by at least one edge. The weight of an edge (A, B) is set to the sum of the weights of edges that run between block A and block B of the clustering.

4.5 Initial Training. As soon as the graphs are small enough, we train the initial model. We perform

model selection and solve the SVM optimization problems by using the C-SVM routine of the LibSVM [6] library. As our algorithm uses the Gaussian kernel to allow for non-linear classification, we have to adjust the model parameters C and γ.

A common way to find the best parameters in SVM learning is to use grid search [34] over the parameter space where a regular grid is put on the parameter space and the parameter combinations corresponding to the grid vertices are evaluated. However, while grid search is robust it is work intensive. In contrast to grid search, Uniform Design (UD) [21] uses a pre-computed set of parameters that have maximal distance to each other. This ensures that the parameter space is covered well but reduces the number of parameters that have to be evaluated. Our implementation of UD is similar to the one used in [37] and performs a two sweep search. That is we begin the search using points that are well distributed and in a second sweep we use points close to the best parameters found during the first sweep.

4.6 Uncoarsening. After initial training, our algorithm improves the solution on every level of the hierarchy. We uncontract the hierarchies for the majority and minority class at the same time. Recall that support vectors are the important data points for the construction of the maximum separating hyperplane. We use this property to decrease the problem size.

Consider an SVM model trained on the previous, coarser level. The support vectors of the model are data points of the previous levels' problem. For the current level we uncontract those support vectors and use the resulting data points as input to train a new SVM model. Solving the optimization problem is again done using LibSVM. In the case that there are more hierarchy levels for the majority class than for the minority class, which is often the case for imbalanced data, we only uncontract the majority hierarchy until both hierarchies have similar size.

Our algorithm also adopts the model parameters from the previous level. The assumption is that model parameters of coarser levels are already good model parameters for the current level. We only perform a second, more fine-grained, UD sweep around the model parameters of the previous level to improve the model. When the problem size exceeds 10 000 data points, we adopt the best parameters from the previous level and skip the model selection process. We proceed uncontraction recursively until the complete hierarchy is processed. We return the best model that we found during uncoarsening. Note this does not have to be the SVM model of the last level.

4.7 Evaluation of Solutions. At different stages of the algorithm we need to assess the quality of the current solution. This is usually done by using data points with known labels and comparing the known correct label to the label which the trained SVM predicts for the data point. More precisely, in machine learning it is common to split a given instance into two parts: the training set and the test set [3]. Learning is conducted on the training set and evaluation is done on the test set. However, we can not use test data for the evaluation tasks during the training phase, otherwise the result of the algorithm would not be meaningful.

Sadrfaridpour et al. [40] discuss several different evaluation approaches. We use the approach that worked best in their studies: Rather than using the data of the current level that our algorithm works on, we use the original problem (i.e., the training set) as validation set in our algorithm. Since we aim at classifying large problems we can not validate every parameter combination on the entire training set, which is why we use a random subset of the training set for evaluation. The size of the set is 10% of the original training data.

5 Experimental Evaluation

System and Methodology. We implemented the algorithm described in the previous section using C++. Our code uses FLANN 1.8.4 [31], LibSVM 3.22 [6], and is compiled with gcc-8.1.1. All of our experiment are executed on a machine with an AMD Opteron 6168 with 1.9GHz and 256GB of RAM.

Name	Size	Feat.	$\mathbf{C}^+$	$\mathbf{C}^-$	Imb.
Advertisement	3 279	1 558	459	2 820	0.86
Buzz	140 707	77	27 775	112 932	0.80
Clean (Musk)	6 598	166	1 017	5 581	0.85
Cod-rna	59 535	8	19 845	39 690	0.67
EEG Eye State	14 980	14	6 723	8 257	0.55
Forest (Class 3)	581 012	54	35 754	369 172	0.94
Forest (Class 5)	581 012	54	9 493	571 519	0.98
Forest (Class 7)	581 012	54	20 510	560 502	0.96
Hypothyroid	3 919	21	240	3 679	0.94
Isolet (Class A)	6 919	617	240	5 998	0.96
Letter (Class Z)	20 000	16	734	19 266	0.96
Nursery	12 960	8	4 320	8 640	0.67
Protein	145 751	74	1 296	144 455	0.99
Ringnorm	7 400	20	3 664	3 736	0.50
Twonorm	7 400	20	3 703	3 697	0.50
APS failure	76 000	170	1 375	74 625	0.98
Census	299 285	41	18 568	280 717	0.94
Letter (Class A)	20 000	16	786	19 266	0.96
Letter (Class B)	20 000	16	766	19 266	0.96
Letter (Class H)	20 000	16	734	19 266	0.96
Skin	245 057	3	50 859	194 198	0.79
Sleep (Class 1)	105 908	13	9 052	96 856	0.91

Table 1: Properties of instances in our benchmark set. The upper part of the table shows all instances that have been employed in [40] and the lower part shows newly added instances.

	mlsvm-AMG	LPSVM	$\text{LPSVM}_{\text{fast}}$	LibSVM
	running time [s]			
Advertisement	343	192	70	557
APS failure	1 473	109	13	-
Buzz	110	121	19	-
Census	3 047	657	38	-
Clean (Musk)	14	8	4	320
Cod-rna	80	43	7	15 700
EEG Eye State	123	1 320	0.9	2 700
Forest (Class 3)	10 156	744	99	-
Forest (Class 5)	6 986	1 090	158	-
Forest (Class 7)	5 393	1 990	114	-
Hypothyroid	2	3	0.9	13
Isolet (Class A)	1 627	23	7	856
Letter (Class A)	17	4	2	1 930
Letter (Class B)	55	4	2	1 590
Letter (Class H)	74	9	2	1 970
Letter (Class Z)	31	3	2	1 710
Nursery	7	2	0.7	998
Protein	3 654	41	17	-
Ringnorm	10	13	0.6	161
Skin	81	18	12	38 200
Sleep (Class 1)	1 594	3 080	12	-
Twonorm	7	0.7	0.5	109

Table 2: Running times of different algorithms to train an SVM.

k-Fold Cross-Validation. In order to evaluate the performance of the algorithms a given instance is split into the training set, which is the only data used during training, and the test set. After solving the SVM optimization problem for the training set, one then compares labels of the *test* data computed by the final SVM with the given labels to measure prediction quality. We use k-fold cross-validation [24] to get a more accurate estimate of the prediction performance. More precisely, we first shuffle the entire data set and split it into k parts of equal size. We then perform k repetitions of the training algorithm. In every run one of the parts is used as the *test set* while the other $k-1$ parts constitute the *training set*. The test set is *never* used to train or validate any of the intermediate results of the hierarchy; only the final result of a run is evaluated using the test set. After all k runs are finished, we average the results of all runs and present that as the overall quality of our training process. We use $k = 5$ in our experiments, since this is the default value in mlsvm-AMG [40]. In our experiments, all algorithms operate on the same k-folds in order to achieve meaningful comparisons. By default we perform five k-folds for each algorithm using different random seeds. Hence, in total we perform 25 different runs for each test instance and report the arithmetic mean of solution quality and running time. When further averaging over multiple instances, we use the geometric mean in order to give every instance a comparable influence on the final score.

Instances. We use the same set of instances that was used by Sadrfaridpour et al. [40] and extend it with large problem instances from the UC Irvine Machine Learning Repository [13]. Basic properties of the instances in our benchmark set are shown in Table 1. The first part of the table shows previously used instances, while the second part shows newly added benchmark instances.

Algorithm Configuration. Any multilevel algorithm has a considerable number of choices between algorithmic components and tuning parameters. The model selection parameters C and γ are not part of the algorithm configuration as they are determined during the initial SVM training. The number ℓ of label propagation iterations during coarsening is fixed to ten, as more iterations rarely found better clusterings for contraction in previous studies such as [30]. We stop the coarsening process, when the graph size is smaller than 500, as this is the default value in [37]. Our algorithm uses the same value $k = 10$ for the construction of the k-nearest neighbor graphs as mlsvm-AMG [40]. This configuration of our algorithm is called LPSVM or quality configuration unless otherwise mentioned. We also use a fast configuration of our algorithm, $\text{LPSVM}_{\text{fast}}$. This configuration

Instance	mlsvm-AMG		LPSVM		$\text{LPSVM}_{\text{fast}}$		LibSVM	
	ACC	G-mean	ACC	G-mean	ACC	G-mean	ACC	G-mean
Advertisement	0.93	**0.91**	0.91	**0.91**	0.82	0.87	0.989	0.941
APS failure	0.94	**0.94**	0.95	0.92	0.95	**0.94**	-	-
Buzz	0.94	**0.95**	0.94	0.94	0.94	0.93	-	-
Census	0.76	0.81	0.85	**0.83**	0.78	0.81	-	-
Clean (Musk)	0.90	0.88	0.96	**0.96**	0.92	0.88	0.998	0.996
Cod-rna	0.94	**0.94**	0.93	**0.94**	0.92	0.93	0.911	0.855
EEG Eye State	0.76	0.75	0.84	**0.83**	0.64	0.63	0.930	0.929
Forest (Class 3)	0.91	0.94	0.94	**0.95**	0.94	0.94	-	-
Forest (Class 5)	0.73	0.79	0.89	**0.90**	0.88	**0.90**	-	-
Forest (Class 7)	0.81	0.86	0.95	**0.93**	0.93	0.91	-	-
Hypothyroid	0.96	**0.94**	0.97	0.90	0.97	**0.94**	0.989	0.927
Isolet (Class A)	0.96	0.00	0.98	**0.99**	0.78	0.88	0.999	0.990
Letter (Class A)	0.93	0.94	0.96	**0.96**	0.95	0.95	0.999	0.995
Letter (Class B)	0.82	0.88	0.93	**0.92**	0.91	**0.92**	0.997	0.979
Letter (Class H)	0.76	0.76	0.88	**0.89**	0.86	0.86	0.996	0.970
Letter (Class Z)	0.89	0.92	0.95	**0.95**	0.94	**0.95**	0.999	0.991
Nursery	1.00	**1.00**	1.00	**1.00**	1.00	**1.00**	1.000	1.000
Protein	0.92	0.92	0.93	**0.93**	0.86	0.91	-	-
Ringnorm	0.98	**0.98**	0.97	0.97	0.82	0.82	0.987	0.987
Skin	0.99	0.99	1.00	**1.00**	0.99	**1.00**	1.000	1.000
Sleep (Class 1)	0.62	0.69	0.73	**0.70**	0.87	0.41	-	-
Twonorm	0.97	**0.97**	0.96	0.96	0.96	0.96	0.981	0.981

Table 3: Computational results of different algorithms to train an SVM.

returns the model that has been found after the initial training phase and does not perform any further refinement during uncoarsening.

Performance Measures. We employ the commonly used performance measures *sensitivity* (SN), *specificity* (SP), *G-mean*, and *accuracy* (ACC) to evaluate our prediction results [40]. The definition for those are

$$\begin{aligned} ACC &= \frac{TP+TN}{FP+TN+TP+FN} \\ SN &= \frac{TP}{TP+FN} \\ SP &= \frac{TN}{TN+FP} \\ G\text{-mean} &= \sqrt{SP \cdot SN}, \end{aligned}$$

where TP are the true positives (the correctly classified points of $\mathbf{C}^+$), FN the false negatives (wrongly classified points of the minority class $\mathbf{C}^+$), TN the true negatives (correctly classified point of $\mathbf{C}^-$), and FP the false positives (points of $\mathbf{C}^-$ wrongly classified as points of the minority class). These metrics are common in statistical analysis with accuracy being the most used metric in machine learning [3]. Note that on large imbalanced data sets getting high accuracy is trivial by only predicting the larger class. Hence, as we work with imbalanced instances, we cannot use accuracy as primary measure for prediction quality. Instead, we use the geometric mean of the sensitivity and the specificity. This is more sensitive to wrong classification in general and yields more informative results on imbalanced data sets. Moreover, when comparing trained SVM models, e.g. in model selection during training or later when searching for the best overall SVM model, we use the G-mean as primary decision criterion. If two models have roughly the same G-mean we choose the model with less support vectors.

5.1 Comparison with State-of-the-Art. We now compare our algorithm to the state-of-the-art algorithm mlsvm-AMG (the previously best system), and to LibSVM. As LibSVM is fairly slow, we stopped computations if the running time exceeded 24 hours. We do not perform additional comparisons with mlsvm-IS [37], DC-SVM [19] and EnsembleSVM [9] as mlsvm-AMG computes similar or improved classification results while being up to one to two orders of magnitude faster on large instances in case of the first three algorithms, or in case of EnsembleSVM [9] the reported classification quality of mlsvm-AMG is significantly higher. We run the algorithms on our machine without tuning parameters on a per instance basis, i.e, we use the same set

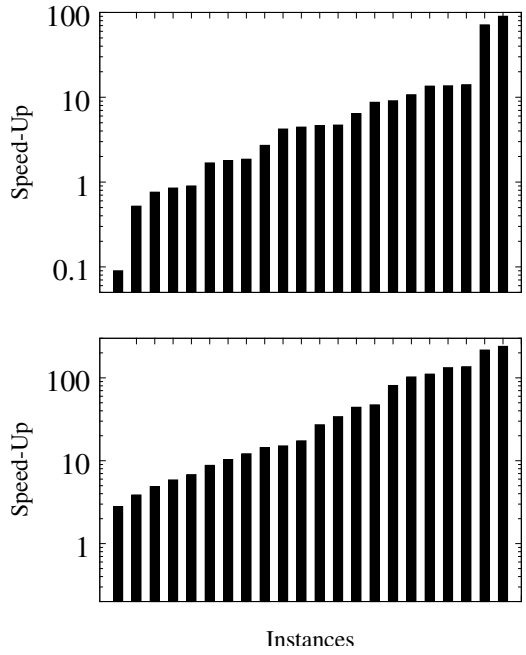

Figure 2: Distribution of speed-ups of our algorithm over mlsvm-AMG when running algorithms on the same k-folds. Top: speed-ups of LPSVM over mlsvm-AMG. Bottom: speed-ups of $\text{LPSVM}_{\text{fast}}$ over mlsvm-AMG.

Dataset	LPSVM	$\text{LPSVM}_{\text{fast}}$
`Advertisement`	1.79	4.88
`APS failure`	13.52	110.85
`Buzz`	0.90	5.86
`Census`	4.64	80.79
`Clean (Musk)`	1.68	3.86
`Cod-rna`	1.86	12.14
`EEG Eye State`	0.09	132.65
`Forest (Class 3)`	13.65	102.49
`Forest (Class 5)`	6.41	44.21
`Forest (Class 7)`	2.71	47.22
`Hypothyroid`	0.85	2.80
`Isolet (Class A)`	71.38	239.67
`Letter (Class A)`	4.70	8.79
`Letter (Class B)`	14.07	27.17
`Letter (Class H)`	8.72	33.99
`Letter (Class Z)`	9.09	15.10
`Nursery`	4.22	10.34
`Protein`	90.21	216.31
`Ringnorm`	0.76	17.38
`Skin`	4.45	6.80
`Sleep (Class 1)`	0.52	136.25
`Twonorm`	10.72	14.38

Table 4: Speed-ups of our quality and fast algorithm configuration over mlsvm-AMG on a per instance basis.

of parameters as described above for every instance. In the original work, mlsvm-AMG [38, 39] uses instance-based parameters, i.e., different instances use different parameters of the multilevel algorithm[1]. Here, we use the same good parameters (provided by one of the co-authors Ehsan Sadrfaridpour [38, 39]) for all instances. Tables 2–4 as well as Figure 2 summarize the results of our study.

First of all, on 17 out of the 22 instances LPSVM computes a better or equal G-mean than mlsvm-AMG. On the remaining 5 instances our algorithm computes a result that is 0.02 worse on average. On the `Isolet` instance mlsvm-AMG computes a model that puts all data points on a single side. Hence, the G-mean value is zero on this instance. We exclude the instance from the following geometric mean computations. Overall, the geometric average improvement in classification quality of LPSVM over mlsvm-AMG is 3% when considering G-mean. We therefore conclude that classification quality is comparable or often better than mlsvm-AMG.

Our fast configuration computes slightly worse models, but still has very good classification results. The geometric average in G-mean of our fast configuration is 2.6% below the value of mlsvm-AMG. Note that the fast and quality configuration of our algorithm use the same random seeds during coarsening and initial training. Still the classification quality on the input test data by $\text{LPSVM}_{\text{fast}}$ is sometimes better than the classification quality computed by the quality configuration, i.e., the configuration that uses additional refinement during uncoarsening. On first sight, this is somewhat surprising since the full algorithm returns the best model that been found during uncoarsening over all levels. However, this effect is due to the fact that our algorithm does not evaluate the model on the entire input data (since this would be too expensive), but uses a subset of the training set to perform the evaluations. Hence, the best model selected by the quality configuration is not necessarily the best model for the input training data. In terms of accuracy, LPSVM computes a better or equal result on 18 out of 22 instances compared to mlsvm-AMG, and our fast configuration does so in 15 out of the 22 instances under consideration.

We now look at the running time spend to train the SVM by different algorithms. Figure 2 shows the distribution of speed-ups of our algorithms over mlsvm-AMG and Table 4 gives detailed per instance results. In terms of running time, already our higher quality configuration LPSVM is faster on almost all instances. The exceptions are `Buzz`, `EEG Eye State`, `Hypothyroid`, `Ringnorm` and `Sleep (Class 1)`. In these cases, most of the time of our algorithm is spend in

[1]Personal communication with Ehsan Sadrfaridpour

the refinement phase. The hierarchy of those instances yields a lot of support vectors for *both* algorithms. That means that a lot of time is spend in solving the optimization problems on the different hierarchy levels. The better running time of the mlsvm-AMG solver is due to a partitioned training technique that we did not incorporate in our algorithm. This technique partitions the training set into k blocks of roughly equal size in order to speedup computations. In general, *positive* speed-ups for LPSVM range from 1.68 up to almost two orders of magnitude. The largest observed speed-ups are on the instances `Isolet`, speed-up 71.38, and `Protein`, speed-up 90.21. The speedup on the `Protein` instance is due to the fact that our algorithm only has three levels in the multilevel hierarchy and there is only a very limited number of support vectors during uncoarsening. In contrast, the mlsvm-AMG algorithm has seven hierarchy levels on that instance.

When considering the faster version of our algorithm, we observe that it is faster on every instance. This is not surprising since most of the time is spend during the refinement phase. However, since the observed classification quality of this approach is not much worse than mlsvm-AMG, we consider this a feasible alternative. For this configuration speed-ups range from 2.8 up to 239.67. The geometric average speed-up for the fast configuration over mlsvm-AMG is 26.1.

Comparing to LibSVM, we note that our algorithms are multiple orders of magnitude faster on all except once instance that LibSVM could solve within the time limit. However, this is at the expense of solution quality, i.e., the solutions computed by LibSVM are mostly better than the ones computed by our algorithm.

6 Conclusion

We present a very fast multilevel support vector machine that uses a label propagation algorithm to construct the problem hierarchy. In contrast to previous approaches, our clustering-based contraction scheme is able to shrink the input size very quickly. Extensive experiments indicate that our algorithm is less affected by the number of features of the data set. Moreover, our new algorithm achieves speed-ups up to an order of magnitude while having similar or better classification quality over state-of-the-art algorithms. Our implementation is publicly available in the open source framework KaSVM[2].

Important future work includes parallelization, both, for shared-memory parallel and distributed memory parallel architectures. Moreover, we want to try different libraries to solve the underlying optimization problem during initial training and uncoarsening such as GENO [25]. Another important aspect is to find clusterings where each cluster has a small diameter. This may have a positive effect on the number of support vector used during uncoarsening and hence further improve the overall running time. Lastly, it will make sense to incorporate automatic algorithm configuration tools such as [22] that are able to predict good algorithm parameters based on properties of the input instance.

[2] `https://algo2.iti.kit.edu/kasvm/`

References

[1] M. A. Aizerman, E. M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] Z. Allen Zhu, W. Chen, G. Wang, C. Zhu, and Z. Chen. P-packsvm: Parallel primal gradient descent kernel SVM. In *Proc. of Ninth IEEE Intl. Conf. on Data Mining*, pages 677–686, 2009.

[3] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proc. of the Fifth Ann. Workshop on Computational Learning Theory*, COLT '92, pages 144–152. ACM, 1992.

[5] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent Advances in Graph Partitioning. In *Algorithm Engineering - Selected Results and Surveys*, pages 117–158, 2016.

[6] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] E. Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, Z. Qiu, and H. Cui. Parallelizing support vector machines on distributed computers. In *Proc. of the Twenty-First Ann. Conf. on Neural Information Processing Systems*, pages 257–264. Curran Associates, Inc., 2007.

[8] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1-3):131–159, 2002.

[9] M. Claesen, F. D. Smet, J. A. K. Suykens, and B. D. Moor. Ensemblesvm: a library for ensemble learning using support vector machines. *J. of Machine Learning Research*, 15(1):141–145, 2014.

[10] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[11] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. of Machine Learning Research*, 2:265–292, 2001.

[12] L. Cui, C. Wang, W. Li, L. Tan, and Y. Peng. Multi-modes cascade svms: Fast support vector machines in distributed system. In *Intl. Conf. on Information Science and Applications*, pages 443–450. Springer, 2017.

[13] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.

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

[15] R. Fan, P. Chen, and C. Lin. Working set selection using second order information for training support vector machines. *J. of Machine Learning Research*, 6:1889–1918, 2005.

[16] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik. Parallel support vector machines: The cascade SVM. In *Advances in Neural Information Processing Systems (NIPS)*, pages 521–528, 2004.

[17] D. M. Harris and S. Harris. Introductory digital design & computer architecture curriculum. In *2013 IEEE Intl. Conf. on Microelectronic Systems Education*, pages 14–16. IEEE, 2013.

[18] S. Horng, M. Su, Y. Chen, T. Kao, R. Chen, J. Lai, and C. D. Perkasa. A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert Syst. Appl.*, 38(1):306–313, 2011.

[19] C. Hsieh, S. Si, and I. S. Dhillon. A divide-and-conquer solver for kernel support vector machines. In *Proc. of the 31th Intl. Conf. on Machine Learning, ICML*, pages 566–574, 2014.

[20] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification, 2010.

[21] C.-M. Huang, Y.-J. Lee, D. Lin, and S.-Y. Huang. Model selection for support vector machine via uniform design. 52:335–346, 09 2007.

[22] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. Paramils: An automatic algorithm configuration framework. *J. Artif. Intell. Res.*, 36:267–306, 2009.

[23] T. Joachims. Making large-scale svm learning practical. Technical report, Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund, 1998.

[24] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of the Fourteenth Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1137–1145, 1995.

[25] S. Laue, M. Mitterreiter, and J. Giesen. GENO Project. `http://www.geno-project.org`.

[26] T. Li, X. Liu, Q. Dong, W. Ma, and K. Wang. HPSVM: heterogeneous parallel SVM with factorization based IPM algorithm on CPU-GPU cluster. In *24th Euromicro Intl. Conf. on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 74–81. IEEE Computer Society, 2016.

[27] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[28] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. on Optimization*, 2(4):575–601, 1992.

[29] H. Meyerhenke, P. Sanders, and C. Schulz. Partitioning Complex Networks via Size-constrained Clustering. In *Proc. of the 13th Int. Symp. on Experimental Algorithms*, LNCS. Springer, 2014.

[30] H. Meyerhenke, P. Sanders, and C. Schulz. Parallel graph partitioning for complex networks. *IEEE Trans. Parallel Distrib. Syst.*, 28(9):2625–2638, 2017.

[31] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2227–2240, 2014.

[32] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing VII*, pages 276–285. IEEE, 1997.

[33] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods*, pages 185–208, 1999.

[34] A. Popov and A. Sautin. Selection of support vector machines parameters for regression using nested grids. In *2008 Third Intl. Forum on Strategic Technologies*, pages 329–331, June 2008.

[35] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng. A survey of machine learning for big data processing. *EURASIP J. on Advances in Signal Processing*, 2016(1):67, May 2016.

[36] U. N. Raghavan, R. Albert, and S. Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. *Physical Review E*, 76(3), 2007.

[37] T. Razzaghi and I. Safro. Scalable multilevel support vector machines. In *Proc. of the Intl. Conf. on Computational Science (ICCS)*, volume 51 of *Procedia Computer Science*, pages 2683–2687. Elsevier, 2015.

[38] E. Sadrfaridpour, S. Jeereddy, K. Kennedy, A. Luckow, T. Razzaghi, and I. Safro. Algebraic multigrid support vector machines. *CoRR*, abs/1611.05487, 2016.

[39] E. Sadrfaridpour, S. Jeereddy, K. Kennedy, A. Luckow, T. Razzaghi, and I. Safro. Algebraic multigrid support vector machines. In *European Symposium on Artificial Neural Networks (ESANN)*, pages 35–40, 2017.

[40] E. Sadrfaridpour, T. Razzaghi, and I. Safro. Engineering fast multilevel support vector machines. *CoRR*, abs/1707.07657, 2017.

[41] S. Salcedo-Sanz, J. L. Rojo-Álvarez, M. Martínez-Ramón, and G. Camps-Valls. Support vector machines in engineering: an overview. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 4(3):234–267, 2014.

[42] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *ICML*, pages 839–846. Citeseer, 2000.

[43] A. Stolcke, S. S. Kajarekar, and L. Ferrer. Nonparametric feature normalization for svm-based speaker verification. In *Proc. of the IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1577–1580. IEEE, 2008.

[44] M. Stolpe, K. Bhaduri, and K. Das. Distributed support vector machines: An overview. In *Solving Large Scale Learning Tasks. Challenges and Algorithms - Essays Dedicated to Katharina Morik on the Occasion of Her 60th Birthday*, volume 9580 of *Lecture Notes in Computer Science*, pages 109–138. Springer, 2016.

[45] Y. Torii and S. Abe. Fast training of linear programming support vector machines using decomposition techniques. In *Artificial Neural Networks in Pattern Recognition, Second IAPR Workshop*, volume 4087 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2006.

[46] Y. You, J. Demmel, K. Czechowski, L. Song, and R. W. Vuduc. CA-SVM: communication-avoiding support vector machines on distributed systems. In *2015 IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS)*, pages 847–859. IEEE Computer Society, 2015.

[47] Y. You, H. Fu, S. L. Song, A. P. Randles, D. J. Kerbyson, A. Marquez, G. Yang, and A. Hoisie. Scaling support vector machines on modern HPC platforms. *J. Parallel Distrib. Comput.*, 76:16–31, 2015.

[48] H. Yu, J. Yang, and J. Han. Classifying large data sets using svms with hierarchical clusters. In *Proc. of the Ninth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 306–315. ACM, 2003.