Speeding Up Budgeted Stochastic Gradient Descent SVM Training with Precomputed Golden Section Search

Tobias Glasmachers(✉) and Sahar Qaadan

Institute for Neural Computation, Ruhr University Bochum,
44801 Bochum, NRW, Germany
{tobias.glasmachers,sahar.qaadan}@ini.rub.de

Abstract. Limiting the model size of a kernel support vector machine to a pre-defined budget is a well-established technique that allows to scale SVM learning and prediction to large-scale data. Its core addition to simple stochastic gradient training is budget maintenance through merging of support vectors. This requires solving an inner optimization problem with an iterative method many times per gradient step. In this paper we replace the iterative procedure with a fast lookup. We manage to reduce the merging time by up to 65% and the total training time by 44% without any loss of accuracy.

1 Introduction

The Support Vector Machine (SVM; [5]) is a widespread standard machine learning method, in particular for binary classification problems. Being a kernel method, it employs a linear algorithm in an implicitly defined kernel-induced feature space [24]. SVMs yield high predictive accuracy in many applications [6,15,16,19,28]. They are supported by strong learning theoretical guarantees [1,9,12,17].

When facing large-scale learning, the applicability of support vector machines (and many other learning machines) is limited by their computational demands. Given n training points, training an SVM with standard dual solvers takes quadratic to cubic time in n [1]. Steinwart [23] established that the number of support vectors is linear in n, and so is the storage complexity of the model as well as the time complexity of each of its predictions. This quickly becomes prohibitive for large n, e.g., when learning from millions of data points.

Due to the prominence of the problem, a large number of solutions was developed. Parallelization can help [29,33], but it does not reduce the complexity of the training problem. One promising route is to solve the SVM problem only locally, usually involving some type of clustering [14,30] or with a hierarchical divide-and-conquer strategy [8,11]. An alternative approach is to leverage the progress in the domain of linear SVM solvers [10,13,32], which scale well to large data sets. To this end, kernel-induced feature representations are approximated

G. Nicosia et al. (Eds.): LOD 2018, LNCS 11331, pp. 329–340, 2019.
https://doi.org/10.1007/978-3-030-13709-0_28

by low-rank approaches [7,20,27,31], either a-priory using random Fourier features, or in a data-dependent way using Nyström sampling.

Budget methods, introducing an a-priori limit $B \ll n$ on the number of support vectors [18,25], go one step further by letting the optimizer adapt the feature space approximation during operation to its needs, which promises a comparatively low approximation error. The usual strategy is to merge support vectors at need, which effectively enables the solver to move support vectors around in input space. Merging decisions greedily minimize the approximation error.

In this paper we propose an effective computational improvement of this scheme. Finding the best merge partners, i.e., support vectors that induce the lowest approximation error when merged, is a rather costly operation. Usually, $\mathcal{O}(B)$ candidate pairs of vectors are considered, and for each pair an optimization problem is solved with an iterative strategy. By modelling the low-dimensional space of (solutions of the) optimization problems explicitly, we can remove the iterative process entirely, and replace it with a simple and fast lookup.

Our results show that merging-based budget maintenance can account for more than half of the total training time. Therefore reducing the merging time is a promising approach to speeding up training. The speed-up can be significant; on our largest data set we reduce the merging time by 65%, which corresponds to a reduction of the total training time by 44%. At the same time, our lookup method is at least as accurate as the original iterative procedure, resulting in nearly identical merging decisions and no loss of prediction accuracy.

The remainder of this paper is organized as follows. In the next section we introduce SVMs and stochastic gradient training on a budget. Then we analyze the computational bottleneck of the solver and develop a lookup smoothed with bilinear interpolation as a remedy. In Sect. 4 we benchmark the new algorithm against "standard" BSGD, and we investigate the influence of the algorithmic simplification on different budget sizes. Our results demonstrate systematic improvements in training time at no cost in terms of solution quality.

2 Support Vector Machine Training

In this section we introduce the necessary background: SVMs for binary classification, and training with stochastic gradient descent (SGD) on a budget, i.e., with a-priori limited number of support vectors.

Support Vector Machines. An SVM classifier is a supervised machine learning algorithm. In its simplest form it linearly separates two classes with a large margin. When applying a kernel function $k : X \times X \to \mathbb{R}$ over the input space X, the separation happens in a reproducing kernel Hilbert space (RKHS). For labeled data $\big((x_1, y_1), \ldots, (x_n, y_n)\big) \in (X \times \{-1, +1\})^n$, the prediction on $x \in X$ is computed as

$$\operatorname{sign}\Big(\big\langle w, \phi(x)\big\rangle + b\Big) = \operatorname{sign}\left(\sum_{j=1}^{n} \alpha_j k(x_j, x) + b\right)$$

with $w = \sum_{j=1}^{n} \alpha_j \phi(x_j)$, where $\phi(x)$ is an only implicitly defined feature map (due to Mercer's theorem, see also [24]) corresponding to the kernel function fulfilling $k(x, x') = \langle \phi(x), \phi(x') \rangle$. Training points x_j with non-zero coefficients $\alpha_j \neq 0$ are called support vectors; the summation in the predictor can obviously be restricted to this subset. The SVM model is obtained by minimizing the following (primal) objective function:

$$P(w, b) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} L\Big(y_i, \big\langle w, \phi(x_i)\big\rangle + b\Big). \tag{1}$$

Here, $\lambda > 0$ is a user-defined regularization parameter and $L(y, \mu) = \max\{0, 1 - y \cdot \mu\}$ denotes the hinge loss, which is a prototypical large margin loss, aiming to separate the classes with a functional margin $y \cdot \mu$ of at least one. The incorporation of other loss functions allows to generalize SVMs to other tasks like multi-class classification, regression, and ranking.

Primal Training. Problem (1) is a convex optimization problem without constraints. It has an equivalent dual representation as a quadratic program (QP), which is solved by several state-of-the-art "exact" solvers like LIBSVM [4] and thunder-SVM [26]. The main challenge is the high dimensionality of the problem, which coincides with the training set size n and can hence easily grow into the millions.

A simple method is to solve problem (1) directly with stochastic gradient descent (SGD), similar to neural network training. When presenting one training point at a time, as done in Pegasos [22], the objective function $P(w, b)$ is approximated by the unbiased estimate

$$P_i(w, b) = \frac{\lambda}{2}\|w\|^2 + L\Big(y_i, \big\langle w, \phi(x_i)\big\rangle + b\Big),$$

where the index $i \in \{1, \ldots, n\}$ follows a uniform distribution. The stochastic gradient $\nabla P_i(w, b)$ is an unbiased estimate of the "batch" gradient $\nabla P(w, b)$ but faster to compute by a factor of n, since it involves only a single training point. Starting from $(w, b) = (0, 0)$, SGD updates the weights according to

$$(w, b) \leftarrow (w, b) - \eta_t \cdot \nabla P_{i_t}(w, b),$$

where t is the iteration counter. With a learning rate $\eta_t \in \Theta(1/t)$ it is guaranteed to converge to the optimum of the convex training problem [2].

With a sparse representation $w = \sum_{(\alpha, \tilde{x}) \in M} \alpha \cdot \phi(\tilde{x})$ the SGD update decomposes into the following algorithmic steps. We scale down all coefficients α uniformly by the factor $1 - \lambda \cdot \eta_t$. If the margin $y_i(\langle w, \phi(x_i)\rangle + b)$ happens to be less than one, then we add a new point $\tilde{x} = x_i$ with coefficient $\alpha = \eta_t \cdot y_i$ to the

model M. With a dense representation holding one coefficient α_i per data point (x_i, y_i) we would add the above value to α_i. The most costly step is the computation of $\langle w, \phi(x_i) \rangle$, which is linear in the number of support vectors (SVs), and hence generally linear in n [23].

SVM Training on a Budget. Budgeted Stochastic Gradient Descent (BSGD) breaks the unlimited growth in model size and update time for large data streams by bounding the number of support vectors during training. The upper bound $B \ll n$ is the budget size. Per SGD step the algorithm can add at most one new support vector; this happens exactly if (x_i, y_i) does not meet the target margin of one (and α_i changes from zero to a non-zero value). After $B + 1$ such steps, the budget constraint is violated and a dedicated budget maintenance algorithm is triggered to reduce the number of support vectors to at most B. The goal of budget maintenance is to fulfill the budget constraint with the smallest possible change of the model, measured by $\|\Delta\|^2 = \|w' - w\|^2$, where w is the weight vector before and w' is the weight vector after budget maintenance. $\Delta = w' - w$ is referred to as the weight degradation.

Budget maintenance strategies are investigated in detail in [25]. It turns out that *merging* of two support vectors into a single new point is superior to alternatives like removal of a point and projection of the solution onto the remaining support vectors. Merging was first proposed in [18] as a way to efficiently reduce the complexity of an already trained SVM. With merging, the complexity of budget maintenance is governed by the search for suitable merge partners, which is $\mathcal{O}(B^2)$ for all pairs, while it is common to apply the $\mathcal{O}(B)$ heuristic resulting from fixing the point with smallest coefficient α_i as a first partner.

When merging two support vectors x_i and x_j, we aim to approximate $\alpha_i \cdot \phi(x_i) + \alpha_j \cdot \phi(x_j)$ with a new term $\alpha_z \cdot \phi(z)$ involving only a single point z. Since the kernel-induced feature map is usually not surjective, the pre-image of $\alpha_i \phi(x_i) + \alpha_j \phi(x_j)$ under ϕ is empty [3,21] and no exact match z exists. Therefore the weight degradation $\Delta = \alpha_i \phi(x_i) + \alpha_j \phi(x_j) - \alpha_z \phi(z)$ is non-zero. For the Gaussian kernel $k(x, x') = \exp(-\gamma \|x - x'\|^2)$, due to its symmetries, the point z minimizing $\|\Delta\|^2$ lies on the line connecting x_i and x_j and is hence of the form $z = hx_i + (1 - h)x_j$. For $y_i = y_j$ we obtain a convex combination $0 < h < 1$, otherwise we have $h < 0$ or $h > 1$. In this paper we merge only vectors of equal label. For each choice of z, the optimal value of α_z is obtained in closed form: $\alpha_z = \alpha_i k(x_i, z) + \alpha_j k(x_j, z)$. This turns minimization of $\|\Delta\|^2 = \alpha_i^2 + \alpha_j^2 - \alpha_z^2 + 2k(x_i, x_j)$ into a one-dimensional non-linear optimization problem, which is solved in [25] with golden section line search. The calculations are further simplified by the relations $k(x_i, z) = k(x_i, x_j)^{(1-h)^2}$ and $k(x_j, z) = k(x_i, x_j)^{h^2}$, which save costly kernel functions evaluations.

Budget maintenance in BSGD usually works in the following sequence of steps, see Algorithm 1: First, x_i is fixed to the support vector with minimal coefficient $|\alpha_i|$. Then the best merge partner x_j is determined by testing B pairs (x_i, x_j), $j \in \{1, \ldots, B + 1\} \setminus \{i\}$. Golden section search is run for each of these steps to determine h to fixed precision $\varepsilon = 0.01$. The weight degradation is computed using the shortcuts mentioned above. Finally, the candidate with minimal

weight degradation is selected and the vectors are merged. Hence, although a single golden search search is fast, the need to run it many times per SGD iteration turns it into a rather costly operation.

Algorithm 1. Procedure Budget Maintenance for a sparse model M

Input/Output: model M
$(\alpha_{\min}, \tilde{x}_{\min}) \leftarrow \arg\min \{|\alpha| \,|\, (\alpha, \tilde{x}) \in M\}$
$WD^* \leftarrow \infty$
for $(\alpha, \tilde{x}) \in M \setminus \{(\alpha_{\min}, \tilde{x}_{\min})\}$ **do**
  $m \leftarrow \alpha/(\alpha + \alpha_{\min})$
  $\kappa \leftarrow k(\tilde{x}, \tilde{x}_{\min})$
  $h \leftarrow \arg\max \{m\kappa^{(1-h')^2} + (1-m)\kappa^{h'^2} \,|\, h' \in [0,1]\}$
  $\alpha_z \leftarrow \alpha_{\min} \cdot \kappa^{(1-h)^2} + \alpha \cdot \kappa^{h^2}$
  $WD \leftarrow \alpha_{\min}^2 + \alpha^2 - \alpha_z^2 + 2 \cdot \alpha_{\min} \cdot \alpha \cdot \kappa$
  if $(WD < WD^*)$ **then**
    $WD^* \leftarrow WD$
    $(\alpha^*, \tilde{x}^*, h^*, \kappa^*) \leftarrow (\alpha, \tilde{x}, h, \kappa)$
$z \leftarrow h^* \cdot \tilde{x}_{\min} + (1-h^*) \cdot \tilde{x}^*$
$\alpha_z \leftarrow \alpha_{\min} \cdot (\kappa^*)^{(1-h^*)^2} + \alpha^* \cdot (\kappa^*)^{(h^*)^2}$
$M \leftarrow M \setminus \{(\alpha_{\min}, \tilde{x}_{\min}), (\alpha^*, \tilde{x}^*)\} \cup \{(\alpha_z, z)\}$

A theoretical analysis of BSGD is provided by [25]. Their Theorem 1 establishes a bound on the error induced by the budget, ensuring that asymptotically the error is governed only by the (unavoidable) weight degradation.

3 Precomputing the Merging Problem

The merging problem for given support vectors x_i and x_j with coefficients α_i and α_j is illustrated in Fig. 1. Our central observation is that the geometry depends only on the (cosine of the) angle between $\alpha_i\phi(x_i)$ and $\alpha_j\phi(x_j)$, and on the relative lengths of the two vectors. These two quantities are captured by the parameters

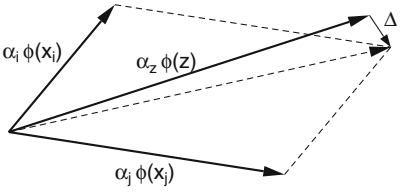


Fig. 1. The merging problem.

- relative length $m = \alpha_i/(\alpha_i + \alpha_j)$
- cosine of the angle $\kappa = k(x_i, x_j)$,

both of which take values in the unit interval. The optimal merging coefficient h is a function of m and κ, and so is the resulting weight degradation $WD = \|\Delta\|^2$. Therefore we can express h and WD as functions of m and κ, denoted as $h(m, \kappa)$

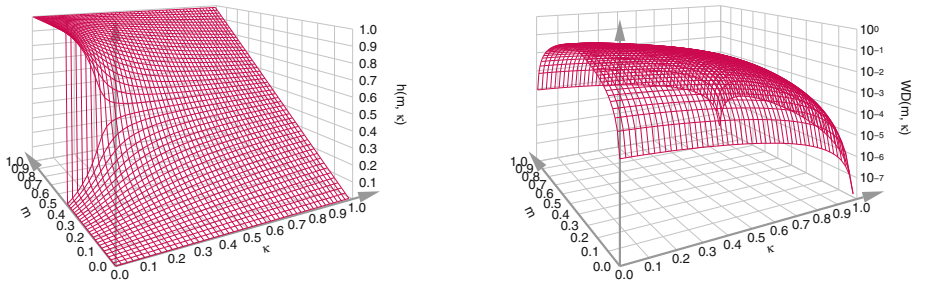
and $WD(m, \kappa)$ in the following. The functions can be evaluated to any given target precision by running the golden section search. Their graphs are plotted in Figs. 2a and b.

If the functions h or WD can be approximated efficiently then there is no need to run a potentially costly iterative procedure like golden section search. This is our core technique for speeding up the BSGD method.

The functions blend between different budget maintenance strategies. While for $\kappa \gg 0$ and for $m \approx 1/2$ it is beneficial to merge the two support vectors, resulting in $h \in (0, 1)$, this is not the case for $\kappa \ll 1$ and $m \approx 0$ or $m \approx 1$, resulting in $h \approx 0$ or $h \approx 1$, which is equivalent to removal of the support vector with smaller coefficient. This means that in order to obtain a close fit that works well in both regimes we may need a quite flexible function class like a kernel method or a neural network, while a simple polynomial function can give poor fits, with large errors close to the boundaries.

A much simpler and computationally very cheap approach is to pre-compute the function on a grid covering the domain $[0, 1] \times [0, 1]$. The values need to be pre-computed only once, and here we can afford to apply golden section search with high precision; we use $\varepsilon = 10^{-10}$. Then, given two merge candidates, we can look up an approximate solution by rounding m and κ to the nearest grid point. The approximation quality can be improved significantly through bilinear interpolation. On modern PC hardware we can easily afford a large grid with millions of points, however, this is not even necessary to obtain excellent results. In our experiments we use a grid of size 400×400.

Bilinear interpolation is fast, and moreover it is easy to implement. When looking up $h(m, \kappa)$ this way, we obtain a plug-in replacement for golden section search in BSGD. However, we can equally well look up $WD(m, \kappa)$ instead to save additional computation steps. Another benefit of WD over h is regularity, see Figs. 2a and b and the following lemma.



(a) (b)

Fig. 2. Graphs of the functions $h(m, \kappa)$ (a) and $WD(m, \kappa)$ (b). The latter uses a log scale on the value axis.

Lemma 1. *The functions h and WD are smooth for $\kappa > e^{-2}$. The function h is continuous outside the set $Z = \{1/2\} \times [0, e^{-2}] \subset [0,1]^2$ and discontinuous on Z. The function WD is everywhere continuous.*

Proof. The function $s_{m,\kappa}(h') = m\kappa^{(1-h')^2} + (1-m)\kappa^{h'^2}$ used in line 7 of Algorithm 1 inside the arg max expression is a weighted sum of two Gaussian kernels. Depending on the parameters m and κ, it can have one or two modes. It has two modes for parameters in Z, as can be seen from an elementary calculation yielding $s''_{1/2,\kappa}(1/2) > 0 \Leftrightarrow \kappa < e^{-2}$. Due to symmetry, the dominant mode switches at $m = 1/2$. The inverse function theorem applied to branches of $s_{m,\kappa}$ implies that $h(m,\kappa) = \arg\max_{h'}\{s_{m,\kappa}(h')\}$ and $WD(m,\kappa) = (\alpha_i + \alpha_j) \cdot \left(m^2 + (1-m)^2 - [s_{m,\kappa}(h(m,\kappa))]^2 + 2m(1-m)\kappa\right)$ vary smoothly with their parameters as long as the same mode is active. The maximum operation is continuous, and so is WD. For each m there is a critical value of $\kappa \leq e^{-2}$ where $s_{m,\kappa}$ switches from one to two modes. We collect these parameter configurations in the set N. On N (in contrast to Z), h is continuous. With the same argument as above, h and WD are smooth outside $N \cup Z$. □

Bilinear interpolation is well justified if the function is continuous, and differentiable within each grid cell. The above lemma ensures this property for $\kappa > e^{-2}$, and it furthermore indicates that for its continuity, interpolating WD is preferable over interpolating h. The regime $\kappa < e^{-2}$ corresponds to merging two points in a distance of more than two "standard deviations" of the Gaussian kernel. This is anyway undesirable, since it can result in a large weight degradation. In fact, if $s_{m,\kappa}$ has two modes, then the optimal merge is close to the removal of one of the points, which is known to give poor results [25].

4 Experimental Evaluation

In this section we evaluate our method empirically, with the aim to investigate its properties more closely, and to demonstrate its practical value. To this end, we'd like to answer the following questions:

1. Which speed-up is achievable?
2. Do we pay for speed-ups with reduced test accuracy?
3. How do results depend on the budget size?
4. How much do merging decision differ from the original method?

To answer these questions we compare our algorithm to "standard" BSGD with merging based on golden section search. We have implemented both algorithms in C++; the implementation is available from the first author's homepage.[1] We train SVM models on the binary classification problems SUSY, SKIN, IJCNN, ADULT, WEB, and PHISHING, covering a range of different sizes. The regularization parameter $C = \frac{1}{n \cdot \lambda}$ and the kernel parameter γ were tuned on a grid of the form $\log_2(C), \log_2(\gamma) \in \mathbb{Z}$ using 10-fold cross-validation. The data sets are summarized in Table 1. SVMs were trained with 20 passes through the data, except for the huge SUSY data, where we used a single pass.

[1] https://www.ini.rub.de/the_institute/people/tobias-glasmachers/#software .

Table 1. Data sets used in this study, hyperparameter settings, and test accuracy of the exact SVM model found by LIBSVM.

data set	size	features	C	γ	accuracy
SUSY	4,500,000	18	2^5	2^{-7}	79.79%
SKIN	183,793	3	2^5	2^{-7}	99.96%
IJCNN	49,990	22	2^5	2^1	98.77%
ADULT	32,561	123	2^5	2^{-7}	84.82%
WEB	17,188	300	2^3	2^{-5}	98.81%
PHISHING	8,315	68	2^3	2^3	97.55%

To answer the first question, we trained SVM models with BSGD, comparing golden section search (GSS) with our new algorithms looking up $h(m, \kappa)$ (Lookup-h) or $WD(m, \kappa)$ (Lookup-WD). For reference, we also ran golden section search with precision $\varepsilon = 10^{-10}$ (GSS-precise). We used two different budget sizes for each problem.

All methods found SVM models with comparable accuracy as shown in Table 2; in fact, in most cases the systematic differences are below one standard deviation of the variability between different runs.[2] In contrast, the time spent on budget maintenance differs significantly between the methods. In Fig. 3 we provide a detailed breakdown of the merging time, obtained with a profiler.

Lookup-WD and Lookup-h are faster than GSS, which is (unsurprisingly) faster than GSS-precise. The results are very systematic, see Table 3 and Fig. 3. The greatest savings of about 44% of the total training time are observed for the rather large SUSY data set. Although the speed-up can also be insignificant, like for the WEB data, lookup is never slower than GSS. The actual saving depends on the cost of kernel computations and on the fraction of SGD iterations in which merging occurs. The latter quantity, which we refer to as the merging frequency, is provided in Table 3. We observe that the savings shown in Fig. 3 nicely correlate with the merging frequency.

The profiler results provide a more detailed understanding of the differences: replacing GSS with Lookup-h significantly reduces the time for computing $h(m, \kappa)$. Replacing Lookup-h with Lookup-WD removes further steps in the calculation of $WD(m, \kappa)$, but practically speaking the difference is hardly noticeable.

Overall, our method offers a systematic speed-up. The speed-up does not come at any cost in terms of solution precision. This answers the first two questions.

If the budget size is chosen so large that merging is never needed then all tested methods coincide, however, this defeats the purpose of using a budget in the first place. We find that the merging frequency is nearly independent of the budget size as long as the budget is significantly smaller than the number of support vectors of the full kernel SVM model, and hence the fraction of runtime saved is independent of the budget size. The results in Fig. 3 are in line with this expectation, answering the third question.

[2] Note that with increasing number of passes (or epochs) the standard deviation does not tend to zero since the training problem is non-convex due to the budget constraint.

Table 2. Test accuracy achieved by the different methods, averaged over 5 runs at different budget sizes.

Data set	Budget size	Test accuracy GSS-precise	Test accuracy GSS-standard	Test accuracy Lookup-h	Test accuracy Lookup-WD
SUSY	100	76.975 ± 1.372	76.628 ± 2.030	76.934 ± 1.426	76.884 ± 1.261
	500	76.989 ± 3.109	75.583 ± 3.0558	75.581 ± 2.558	75.570 ± 3.925
SKIN	100	99.621 ± 0.711	99.629 ± 0.852	99.621 ± 0.201	99.617 ± 0.877
	200	99.868 ± 0.033	99.877 ± 0.053	99.855 ± 0.054	99.754 ± 0.089
IJCNN	100	97.141 ± 0.317	96.807 ± 0.344	97.132 ± 0.371	97.130 ± 0.363
	500	98.138 ± 0.158	98.055 ± 0.334	98.113 ± 0.448	98.070 ± 0.372
ADULT	100	$84.234 \pm 0,883$	84.166 ± 0.701	84.164 ± 0.988	84.200 ± 0.798
	500	84.280 ± 0.800	83.739 ± 1.303	83.836 ± 1.157	83.949 ± 1.001
WEB	100	$98.805 \pm 0,026$	98.793 ± 0.027	98.783 ± 0.045	98.793 ± 0.039
	500	$98.809 \pm 0,023$	98.781 ± 0.047	98.799 ± 0.029	98.807 ± 0.016
PHISHING	100	96.554 ± 0.158	96.254 ± 0.301	96.539 ± 0.242	96.389 ± 0.371
	500	97.555 ± 0.187	97.517 ± 0.292	97.518 ± 0.280	97.525 ± 0.201

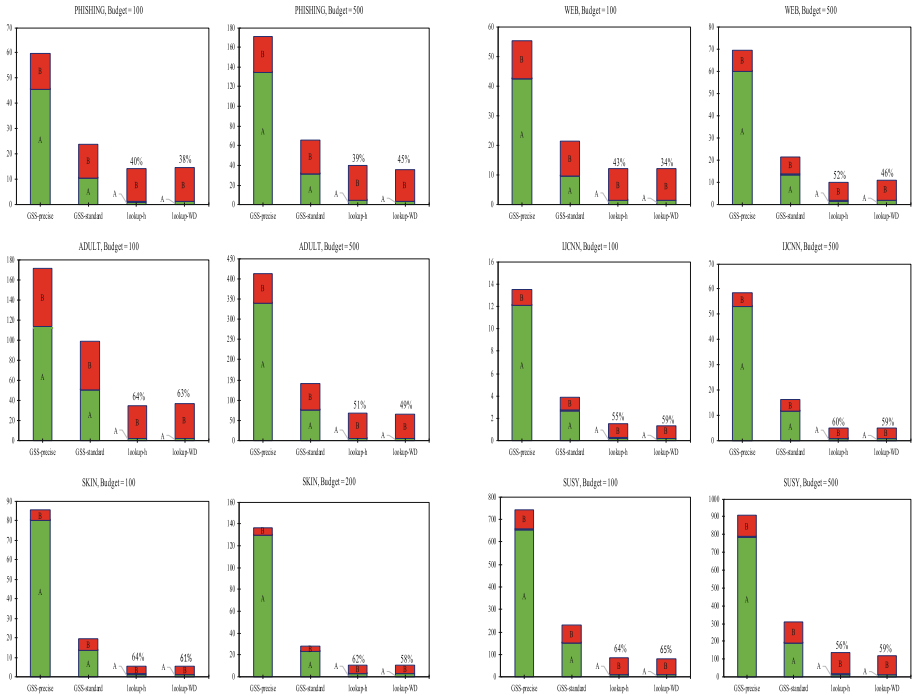


Fig. 3. Breakdown of the merging time in seconds for GSS-precise, GSS, Lookup-h and Lookup-WD. Section A represents the time invested to compute h using either golden section search or lookup. For the Lookup-WD method the same bar represents the look-up of $WD(m, \kappa)$. Section B summarizes all other operations like loop overheads, the computation of α_z, and the construction of the final merge vector z. The numbers on top of the columns for the lookup methods indicate the saving over GSS.

Table 3. Relative improvement of the total training time with respect to golden section search averaged over 5 runs (Lookup-h vs. GSS-standard and lookup-WD vs. GSS-standard), and fraction of merging events for budget size 100 and statistics on the quality of merging decisions (refer to the text for details).

Data set	Budget size	Lookup-h vs. GSS-standard	Lookup-WD vs. GSS-standard	Merging frequency	Equal merging decisions	Factor GSS	Factor lookup-WD
SUSY	100	43.911%	43.396%	43%	93.64%	1.01795	1.00733
	500	39.201%	39.199%				
SKIN	100	20.515%	17.788%	16%	74.31%	1.00047	1.00005
	200	14.173%	14.900%				
IJCNN	100	28.091%	30.372%	17%	91.79%	1.02429	1.00149
	500	30.569%	29.861%				
ADULT	100	21.627%	18.452%	32%	92.54%	1.05064	1.00402
	500	22.334%	22.339%				
WEB	100	3.053%	5.649%	6%	93.77%	1.00255	1.00039
	500	7.483%	0.508%				
PHISHING	100	15.385%	13.946%	21%	96.96%	1.00055	1.00008
	500	7.563%	10.924%				

In the next experiment we have a closer look at the impact of lookup-based merging decisions by investigating the behavior in single iterations, as follows. During a run of BSGD we execute GSS and Lookup-WD in parallel. We count the number of iterations in which the merging decisions differ, and if so, we also record the difference between the weight degradation values. The results are presented in Table 3. They show that the decisions of the two methods agree most of the time, for some problems in more than 99% of all budget maintenance events.

Finally, we investigate the precision with which the weight degradation is estimated by the different methods. While GSS can solve the problem to arbitrary precision, the reference implementation determines $h(m, \kappa)$ only to a rather loose precision of $\varepsilon = 0.01$ in order to save computation time. In contrast, we ran GSS to high precision $\varepsilon = 10^{-10}$ when precomputing the lookup table, however, we may lose some precision due to bilinear interpolation. This loss shrinks as the grid size grows, which comes at added storage cost, but without any runtime cost. We investigate the precision of GSS and Lookup-WD by comparing them to GSS-precise, which is considered a reasonable approximation of the exact minimum of $\|\Delta\|^2$. For both methods we record the factor by which their squared weight degradations exceed the minimum, see Table 3. All factors are very close to one, hence none of the algorithms is wasteful in terms of weight degradation, and indeed Lookup-WD with a grid size of 400×400 is more precise on all 6 data sets. This answers our last question.

5 Conclusion

We have proposed a fast lookup as a plug-in replacement for the iterative golden section search procedure required when merging support vectors in large-scale kernel SVM training. The new method compares favorably to the iterative baseline in terms of training time: it offers a systematic speed-up, resulting in computational savings of up to 65% of the merging time and up to 44% of the total training time, while the training time is never increased. With our method, nearly the full computation time is spent on actual SGD steps, while the fraction of efforts spent on budget maintenance can be reduced significantly. We have demonstrated that our approach results in virtually indistinguishable and even slightly more precise merging decisions. It is for this reason that the speed-up comes at absolutely no cost in terms of predictive accuracy.

Acknowledgments. We acknowledge support by the Deutsche Forschungsgemeinschaft (DFG) through grant GL 839/3-1.

References

1. Bottou, L., Lin, C.J.: Support Vector Machine Solvers, pp. 1–28. MIT Press, Cambridge (2007)
2. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Lechevallier, Y., Saporta, G. (eds.) COMPSTAT 2010, pp. 177–186. Physica-Verlag, Heidelberg (2010). https://doi.org/10.1007/978-3-7908-2604-3_16
3. Burges, C.J.: Simplified support vector decision rules, pp. 71–77. Morgan Kaufmann (1996)
4. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**, 27 (2011)
5. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
6. Cui, J., Li, Z., Lv, R., Xu, X., Gao, J.: The application of support vector machine in pattern recognition. IEEE Trans. Control Autom. (2007)
7. Fine, S., Scheinberg, K.: Efficient SVM training using low-rank kernel representations. J. Mach. Learn. Res. **2**(Dec), 243–264 (2001)
8. Graf, H.P., Cosatto, E., Bottou, L., Dourdanovic, I., Vapnik, V.: Parallel support vector machines: the cascade SVM. In: NIPS (2005)
9. Hare, S., et al.: Struck: structured output tracking with kernels. IEEE Trans. Pattern Anal. Mach. Intell. **38**(10), 2096–2109 (2016)
10. Hsieh, C.J., Chang, K.W., Lin, C.J., Keerthi, S.S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: ICML (2008)
11. Hsieh, C.J., Si, S., Dhillon, I.: A divide-and-conquer solver for kernel support vector machines. In: International Conference on Machine Learning (ICML), pp. 566–574 (2014)
12. Joachims, T.: Text categorization with support vector machines: learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 137–142. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0026683

13. Joachims, T.: Making large-scale SVM learning practical. In: Advances in Kernel Methods - Support Vector Learning. MIT Press, Cambridge (1999)
14. Ladicky, L., Torr, P.: Locally linear support vector machines. In: International Conference on Machine Learning (ICML), pp. 985–992 (2011)
15. Lewis, D.P., Jebara, T., Noble, W.S.: Support vector machine learning from heterogeneous data: an empirical analysis using protein sequence and structure. Bioinformatics **22**(22), 2753–2760 (2006)
16. Lin, G., Shen, C., Shi, Q., van den Hengel, A., Suter, D.: Fast supervised hashing with decision trees for high-dimensional data. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014)
17. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. MIT Press, Cambridge (2012)
18. Nguyen, D., Ho, T.: An efficient method for simplifying support vector machines. In: Proceedings of the 22nd ICML, pp. 617–624 (2005)
19. Noble, W.S.: Support vector machine applications in computational biology. In: Schölkopf, B., Tsuda, K., Vert, J.P. (eds.) Kernel Methods in Computational Biology. MIT Press, Cambridge (2004)
20. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. NIPS **3**(4) (2007)
21. Schölkopf, B., et al.: Input space versus feature space in kernel-based methods. IEEE Trans. Neural Netw. **10**(5), 1000–1017 (1999)
22. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: primal estimated sub-GrAdient SOlver for SVM. Math. Program. **127**(1), 3–30 (2011)
23. Steinwart, I.: Sparseness of support vector machines. J. Mach. Learn. Res. **4**(Nov), 1071–1105 (2003)
24. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995). https://doi.org/10.1007/978-1-4757-2440-0
25. Wang, Z., Crammer, K., Vucetic, S.: Breaking the curse of kernelization: budgeted stochastic gradient descent for large-scale SVM training. J. Mach. Learn. Res. **13**, 3103–3131 (2012)
26. Wen, Z., Shi, J., He, B., Li, Q., Chen, J.: Thunder-SVM (2017). https://github.com/zeyiwen/thundersvm
27. Mu, Y., Hua, G., Fan, W., Chang, S.F.: Hash-SVM: scalable kernel machines for large-scale visual classification. In: IEEE Conference on Computer Vision and Pattern Recognition (2014)
28. Yu, J., Xue, A., Redei, E., Bagheri, N.: A support vector machine model provides an accurate transcript-level-based diagnostic for major depressive disorder. Transl. Psychiatry **6**(10), e931 (2016). https://doi.org/10.1038/tp.2016.198
29. Zanni, L., Serafini, T., Zanghirati, G.: Parallel software for training large scale support vector machines on multiprocessor systems. J. Mach. Learn. Res. **7**, 1467–1492 (2006)
30. Zhang, H., Berg, A.C., Maire, M., Malik, J.: SVM-KNN: discriminative nearest neighbor classification for visual category recognition. In: Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 2126–2136. IEEE (2006)
31. Zhang, K., Lan, L., Wang, Z., Moerchen, F.: Scaling up kernel SVM on limited resources: a low-rank linearization approach. In: AISTATS (2012)
32. Zhang, T.: Solving large scale linear prediction problems using stochastic gradient descent. In: International Conference on Machine Learning (2004)
33. Zhu, Z.A., Chen, W., Wang, G., Zhu, C., Chen, Z.: P-packSVM: parallel primal gradient descent kernel SVM. In: IEEE International Conference on Data Mining (2009)