Fast Hyperparameter Tuning for Support Vector Machines with Stochastic Gradient Descent

Marcin Orchel[1](✉) and Johan A. K. Suykens[2]

[1] Department of Computer Science, AGH University of Science and Technology, Kraków, Poland
morchel@agh.edu.pl

[2] ESAT-STADIUS, KU Leuven, 3001 Leuven (Heverlee), Belgium
johan.suykens@esat.kuleuven.be

Abstract. We propose a fast training procedure for the support vector machines (SVM) algorithm which returns a decision boundary with the same coefficients for any data set, that differs only in the number of support vectors and kernel function values. The modification is based on the recently proposed SVM without a regularization term based on stochastic gradient descent (SGD) with extreme early stopping in the first epoch. We realize two goals during the first epoch: we decrease the objective function value, and we tune the margin hyperparameter M. Experiments show that a training procedure with validation can be speed up substantially without affecting sparsity and generalization performance.

Keywords: Support vector machines · Stochastic gradient descent

We solve a classification problem by using SVM [14]. The SVM have been shown effective in many applications including computer vision, natural language, bioinformatics, and finance [12]. There are three main performance measures for SVM : the generalization performance, sparsity of a decision boundary and computational performance of learning. SVM are in the group of the most accurate classifiers and are generally the most efficient classifiers in terms of overall running time [16]. They may be preferable due to its simplicity compared to deep learning approach for image data, especially when training data are sparse. One of the problem in the domain of SVM is to efficiently tune two hyperparameters: the cost C which is a trade-off between the margin and the error term; and σ which is a parameter of a Gaussian kernel, also called the radial basis function (RBF) kernel [14]. The grid search is the most used in practice due to its simplicity and feasibility for SVM , where only two hyperparameters are tuned. The generalization performance of sophisticated meta-heuristic methods for hyperparameter optimization for SVM , like genetic algorithms, particle swarm optimization, estimation of distribution algorithms is similar to simpler random search and grid search [9]. The random search can have some advantages over grid search when more hyperparameters are considered like for neural networks [1]. The random search still requires considerable fraction of the grid size. The problem with a grid search method is high computational cost due to exhaustive search of a discretized hyperparameter space.

In this article, we tackle the problem of improving performance of hyperparameter search for the cost C in terms of computational time while preserving sparsity and generalization. In [4], authors use a general approach of checking fewer candidates. They first use a technique for finding optimal σ value, then they use a grid search exclusively for C with an elbow method. The potential limitation of this method is that it still requires a grid search for C, and there is an additional parameter, tolerance for an elbow point. In practice, the number of checked values has been reduced to 5 from 15. In [3], authors use an analytical formula for C in terms of a jackknife estimate of the perturbation in the eigenvalues of the kernel matrix. However, in [9] authors find that tuning hyperparameters generally results in substantial improvements over default parameter values. Usually, a cross validation is used for tuning hyperparameters which additionally increases computational time.

Recently, an algorithm for solving SVM using SGD has been proposed [10] with interesting properties. We call it Stochastic Gradient Descent for Support Vector Classification (SGD-SVC) for simplicity. Originally, it was called OLLAWV. It always stops in the first epoch, which we call *extreme early stopping* and has a related property of not using a regularization term. The SGD-SVC is based on iterative learning. Online learning has a long tradition in machine learning starting from a perceptron [12]. Online learning methods can be directly used for batch learning. However, the SGD-SVC is not a true online learning algorithm, because it uses knowledge from all examples in each iteration. The SGD-SVC due to its iterative nature is similar to many online methods having roots in a perceptron, like the Alma Forecaster [2] that maximizes margin. Many perceptron-like methods have been kernelized, some of them also related to SVM like kernel-adatron [14]. In this article, we reformulate slightly the SGDSVC by replacing a hyperparameter C with a margin hyperparameter M. This parameter is mentioned as a desired margin in [14], def. 4.16. The margin plays a central role in SVM and in a statistical learning theory, especially in generalization bounds for a soft margin SVM. The reformulation leads to simpler formulation of a decision boundary with the same coefficients for any data set that differs only in kernel function values and the number of support vectors which is related to the margin M. Such simple reformulation of weights is close in spirit to the empirical Bayes classifier, where all weights are the same. It has been inspired by fast heuristics used by animals and humans in decision-making [6]. The idea of replacing the C hyperparameter has been mentioned in [13] and proposed as ν support vector classification (ν-SVC). The problem is that it leads to a different optimization problem and is computationally less tractable. The ν-SVC has been also formulated as ν being a direct replacement of $C = 1/(n\nu)$ in [14], where n is the number of examples, with the same optimization problem as support vector classification (SVC). The margin classifier has been mentioned in [15], however, originally it has been artificially converted to the classifier with the regularization term. The statistical bounds for the margin classifier has been given in [5], but without proposing a solver based on these bounds. There is also a technique of solution/regularization path with a procedure of computing a solution for some values of C using a piecewise linearity property. However, the approach is complicated and requires solving a system of equations and several checks of $O(n)$ [7]. In the proposed method, we use one solution for a particular M for generating all solutions for remaining values of M.

The outline of the article is as follows. First, we define a problem, then the methods and update rules. After that, we show experiments on real world data sets.

1 Problem

We consider a classification problem for a given sample data $\boldsymbol{x_i}$ mapped respectively to $y_i \in \{-1, 1\}$ for $i = 1, \ldots, n$ with the following decision boundary

$$f(\boldsymbol{x}) \equiv \boldsymbol{w} \cdot \varphi(\boldsymbol{x}) = 0, \tag{1}$$

where $\boldsymbol{w} \in \mathbb{R}^m$ with the feature map $\varphi(\cdot) \in \mathbb{R}^m$, $f(\cdot)$ is a decision function. We classify data according to the sign of the left side $f(\boldsymbol{x})$. This is the standard decision boundary formulation used in SVM with a feature map and without a free term b. The primal optimization problem for (C-SVC) is

Optimization problem (OP) 1.

$$\min_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \max \{0, 1 - y_i (\boldsymbol{w} \cdot \varphi(\boldsymbol{x_i}))\}, \tag{2}$$

where $C > 0$, $\varphi(\boldsymbol{x_j}) \in \mathbb{R}^m$.

The first term in (2) is known as a *regularization term* (regularizer), the second term is an *error term*. The $\boldsymbol{w}$ can be written in the form

$$\boldsymbol{w} \equiv \sum_{j=1}^{n} \beta_j \varphi(\boldsymbol{x_j}), \tag{3}$$

where $\boldsymbol{\beta} \in \mathbb{R}^n$. We usually substitute (3) to a decision boundary and we get

$$\sum_{j=1}^{n} \beta_j \varphi(\boldsymbol{x_j}) \cdot \varphi(\boldsymbol{x}) = 0. \tag{4}$$

The optimization problem OP 1 is reformulated to find β_j parameters.

The SGD procedure for finding a solution of SVM proposed in [10], called here SGD-SVC is to update parameters β_k iteratively using the following update rule for the first epoch

$$\beta_k \leftarrow -\eta_k \begin{cases} -C y_{w(k)}, & \text{if } 1 - y_{w(k)} \sum_{j=1}^{k-1} \beta_j \varphi\left(\boldsymbol{x_{w(j)}}\right) \cdot \varphi\left(\boldsymbol{x_{w(k)}}\right) \geq 0 \\ 0, & \text{otherwise} \end{cases}, \tag{5}$$

where η_k is a learning rate set to $\eta_k = 1/\sqrt{k}$ for $k = 1, \ldots, n$, all β_k are initialized with 0 before the first epoch. We set $w(1) = 1$. We always stop in the first epoch, either when the condition in (5) is violated, or when we updated all parameters β_k. The $w(k)$ is used for selection of an index using the worst violator technique. It means that we look for the example among all remaining examples, with the worst value of the condition in (5). We check the condition only for the examples not being used in the iteration process before. The worst violators are searched among all remaining examples, so when one wants to use this method for online learning, it is still required to train the model in a batch for optimal performance. We use a version of SVM without a free term b for simplicity, which does not impact any performance measures. We update each parameter maximally one time. Finally, only parameters β_k for the fulfilled condition during the iteration process have nonzero values. The remaining parameters β_k have zero values. In that way, we achieve sparsity of a solution. The number of iterations n_c for β_k parameters with the fulfilled condition is also the number of support vectors. The derivation of an update rule has been already given in [10]. We call the algorithm that stops always in the first epoch as *extreme early stopping*.

The idea that we want to explore is to get rid of the C hyperparameter from the update rule and from the updated term for β_k (5).

2 Solution – Main Contribution

The decision boundary (4) for SGD-SVC can be written as

$$\sum_{k=1}^{n_c} C y_{w(k)} \eta_k \varphi\left(\boldsymbol{x}_{\boldsymbol{w}(\boldsymbol{k})}\right) \cdot \varphi\left(\boldsymbol{x}\right) = 0, \tag{6}$$

where $n_c \leq n$ is the number of support vectors. In the same way, we can write the margin boundaries

$$\sum_{k=1}^{n_c} C y_{w(k)} \eta_k \varphi\left(\boldsymbol{x}_{\boldsymbol{w}(\boldsymbol{k})}\right) \cdot \varphi\left(\boldsymbol{x}\right) = \pm 1. \tag{7}$$

When we divide by C, we get

$$\sum_{k=1}^{n_c} y_{w(k)} \eta_k \varphi\left(\boldsymbol{x}_{\boldsymbol{w}(\boldsymbol{k})}\right) \cdot \varphi\left(\boldsymbol{x}\right) = \pm 1/C. \tag{8}$$

The left side is independent of C, the right side is a new margin value. The new decision boundary can be written as

$$\sum_{k=1}^{n_c} y_{w(k)} \eta_k \varphi\left(\boldsymbol{x}_{\boldsymbol{w}(\boldsymbol{k})}\right) \cdot \varphi\left(\boldsymbol{x}\right) = 0. \tag{9}$$

We propose a classifier based on a margin solving the following optimization problem

OP 2.

$$\min_{\boldsymbol{w}} \frac{1}{2} \left\|\boldsymbol{w}\right\|^2 + \sum_{i=1}^{n} \max\left\{0, M - y_i \left(\boldsymbol{w} \cdot \varphi\left(\boldsymbol{x_i}\right)\right)\right\}, \tag{10}$$

where $M > 0$ is a desired *margin* – a hyperparameter that replaces the C hyperparameter. We call it M Support Vector Classification (M-SVC). The classifier with explicitly given margin has been investigated in [14]. In our approach, we tune a margin, unlike for standard SVM when the margin is optimized, see [14] page 220. We have the following proposition.

Proposition 1. *The OP 2 is equivalent to OP 1.*

Proof. We can write (10) as

$$\min_{\boldsymbol{w}} \frac{1}{2} \left\|\boldsymbol{w}\right\|^2 + M \sum_{i=1}^{n} \max\left\{0, 1 - y_i \left(\frac{\boldsymbol{w}}{M} \cdot \varphi\left(\boldsymbol{x_i}\right)\right)\right\}. \tag{11}$$

When we substitute $\boldsymbol{w}' \rightarrow \boldsymbol{w}/M$, we get

$$\min_{\boldsymbol{w}'} \frac{1}{2} \left\|\boldsymbol{w}' M\right\|^2 + M \sum_{i=1}^{n} \max\left\{0, 1 - y_i \left(\boldsymbol{w}' \cdot \varphi\left(\boldsymbol{x_i}\right)\right)\right\}, \tag{12}$$

So we get

$$\min_{\boldsymbol{w}'} \frac{1}{2} \left\|\boldsymbol{w}'\right\|^2 + \frac{1}{M} \sum_{i=1}^{n} \max\left\{0, 1 - y_i \left(\boldsymbol{w}' \cdot \varphi\left(\boldsymbol{x_i}\right)\right)\right\}. \tag{13}$$

The M is related to C by

$$M = 1/C. \tag{14}$$

It is a similar term as for ν-SVC classifier given in [14], where $C = 1/(n\nu)$ and $\nu \in (0, 1]$. Because the optimization problems are equivalent, generally all properties of SVM in the form OP 2 applies also for M-SVC. In [14], page 211, authors stated an SVM version, where the margin M is automatically optimized as an additional variable. However, they still have the constant C. From the statistical learning theory point of view, the original bounds [14], page 211 applies for a priori chosen M.

We can derive the update rules for M-SVC similar as for SGD-SVC. The new update rules called (SGD-M-SVC) are

$$\beta_k \leftarrow -\eta_k \begin{cases} -y_{w(k)}, & \text{if } M - y_{w(k)} \sum_{j=1}^{k-1} \beta_j \varphi\left(\boldsymbol{x_{w(j)}}\right) \cdot \varphi\left(\boldsymbol{x_{w(k)}}\right) \geq 0 \\ 0, & \text{otherwise} \end{cases}. \tag{15}$$

In the proposed update rules, there is no hyperparameter in the updated value, only in the condition, in opposite to (5). It means that for different values of a margin M, we get solutions that differ only in the number of terms. The corresponding values of parameters β_k are the same for each M value, so the ordering of corresponding parameters is the same. It means that we do not need to tune values of parameters β_k, only the stopping criterion and thus the number of terms in a solution. When we have a set of M values, and we have a model for the $M_{\max}$, we can generate solutions for all remaining M values just by removing the last terms in the solution for $M_{\max}$. We have a correspondence between M value and the number of support vectors n_c stated as follows.

Proposition 2. *After running the SGD-M-SVC for any two values M_1 and M_2, such as $M_1 > M_2$, the number of support vectors n_c is bigger or equals for M_1.*

Proof. The n_c is the number of support vectors and also the number of terms. The stopping criterion is the opposite for the update condition (15) for the k-th iteration. Due to the form $M < \cdot$, it is fulfilled earlier for M_2. There is a special case when stopping criterion would not be triggered for both values, then we get the same model with n terms. Another special case is when only one condition is triggered, then we get model for M_2 and for M_1 with all n terms.

3 Theoretical Analysis

The interesting property of the new update rules is that we realize two goals with update rules: we decrease the objective function value (10) and simultaneously, we generate solutions for a set of given different values of a hyperparameter M, and all is done in the first epoch. We can say, that we solve a discrete non-convex optimization problem OP 2 where we can treat M as a discrete variable to optimize. The main question that we want to address is *how is it possible, that we can effectively generate solutions for different values of M in the first epoch.* First, note due to convergence analysis of a stochastic method, we expect that we improve the objective function value of (10) during the iteration process. We provide an argument that we are able to generate solutions for different values of M. The SVM can be reformulated as solving a multiobjective optimization problem [11] with two goals, a regularization term, and the error term (2). The SVM is a weighted (linear) scalarization with the C being a scalarization parameter. For the corresponding multiobjective optimization problem for OP 2, we have the M scalarization parameter instead. Due to convexity of the two goals, the set all solutions of SVM for different values of C is a Pareto frontier for the multiobjective optimization problem. We show that during the iteration process, we generate *approximated Pareto optimal solutions.* The error term for the t-th iteration of SGD-M-SVC for the example to be added $\boldsymbol{x}_{\boldsymbol{w}(t+1)}$ can be written as

$$\sum_{\substack{i=1\\ i\neq t+1}}^{n} \max\left\{0, M - y_{w(i)} f_t\left(\boldsymbol{x}_{\boldsymbol{w}(i)}\right)\right\} + \max\left\{0, M - y_{w(t+1)} f_t\left(\boldsymbol{x}_{\boldsymbol{w}(t+1)}\right)\right\}, \quad (16)$$

where $f_t(\cdot)$ is a decision function of SGD-M-SVC after t-th iteration. After adding $t+1$-th parameter, we get an error term

$$\sum_{\substack{i=1\\ i\neq t+1}}^{n} \max\left\{0, M - y_{w(i)} f_t\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{i})}\right) - y_{w(i)} y_{w(t+1)} \frac{1}{\sqrt{t+1}} \varphi\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{t+1})}\right) \cdot \varphi\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{i})}\right)\right\}$$
$$+ \max\left\{0, M - y_{w(t+1)} f_t\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{t+1})}\right) - \frac{1}{\sqrt{t+1}}\right\} \tag{17}$$

assuming that we replace a scalar product with an RBF kernel function. The update for the regularization term from (10) is

$$\left\|\boldsymbol{w}_{\boldsymbol{t+1}}\right\|^2 = \sum_{i=1}^{t+1} \sum_{j=1}^{t+1} y_{w(i)} y_{w(j)} \frac{1}{\sqrt{i}} \frac{1}{\sqrt{j}} \varphi\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{i})}\right) \cdot \varphi\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{j})}\right). \tag{18}$$

So we get

$$\left\|\boldsymbol{w}_{\boldsymbol{t+1}}\right\|^2 = \left\|\boldsymbol{w}_{\boldsymbol{t}}\right\|^2 + 2 y_{w(t+1)} \frac{1}{\sqrt{t+1}} f_t\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{t+1})}\right) + \frac{1}{\sqrt{t+1}} \frac{1}{\sqrt{t+1}}. \tag{19}$$

The goal of analysis is to show that during the iteration process, we expect decreasing value of an error term and increasing value of a regularization term. It is the constraint for generating Pareto optimal solutions. Due to Proposition 2, we are increasing value of M, which corresponds to decreased value of C due to (14). For SVM, oppositely, we are increasing value of a regularization term, when C is increased. We call this property a *reversed scalarization for extreme early stopping*. First, we consider the error term. We compare the error term after adding an example (17) to the error term before adding the example (16). The second term in (17) stays the same or it has smaller value due to the update condition for the $t+1$-th iteration

$$M - y_{w(t+1)} f_t\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{t+1})}\right) \geq 0 \tag{20}$$

and due to the positive $1/\sqrt{t+1}$. Moreover, the worst violator selection technique maximizes the left side of (20) among all remaining examples, so it increases the chance of getting smaller value. Now regarding the first term in (16). After update (17), we decrease a value of this term for examples already processed with same class so for which $y_{w(i)} = y_{w(t+1)}$ for $i \leq t$. However, we increase particular terms for remaining examples with the opposite class. The worst violators will likely be surrounded by examples for an opposite class. So we expect bigger similarities to the examples with the opposite class, thus we expect $\varphi\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{t+1})}\right) \cdot \varphi\left(\boldsymbol{x}_{\boldsymbol{w}\,(\boldsymbol{i})}\right)$ to be bigger.

Regarding showing increasing values of (19) during the iteration process. The third term in (19) is positive. The second term in (19) can be positive or negative. It is closely related to the update condition (20). During the iteration process, we expect the update condition to be improved, because, we have an improved model. During the iteration process, the update condition starts to improving and there is a point for which

$$y_{w(t+1)} f_t \left(\boldsymbol{x}_{\boldsymbol{w}(\boldsymbol{t+1})} \right) > -1/\sqrt{t+1}. \tag{21}$$

Then the update for (19) becomes positive. We call this point a *Pareto starter*. So we first optimize the objective function value by minimizing the regularization term and minimizing the error term, then after Pareto starter we generate approximated Pareto optimal solutions, while still improving the objective function value by minimizing only the error term.

3.1 Bounds for M

We bound M by finding bounds for the decision function $f(\cdot)$. Given σ, we can compute the lower and upper bound for $f(\cdot)$ for the RBF kernel for a given number of examples as follows

$$l = (-1) \exp\left(0/\left(-2\sigma^2\right)\right) \sum_{i=1}^{n} \frac{1}{\sqrt{i}}, u = \exp\left(0/\left(-2\sigma^2\right)\right) \sum_{i=1}^{n} \frac{1}{\sqrt{i}} = \sum_{i=1}^{n} \frac{1}{\sqrt{i}}. \tag{22}$$

It holds that $l \leq f(\cdot) \leq u$. In the lower bound, we assume all examples with a class -1. The upper bound is a harmonic number $H_{n,0.5}$. The bounds capture cases when margin functions have all examples on the same side. For random classes for examples (with the Rademacher distribution), the expected value of l (with replaced -1 with classes for particular examples) is 0. We also consider the case with one support vector with class 1 for capturing the error term close to 0. We have the error term $1 - \exp\left(1/\left(-2\sigma^2\right)\right)$. Given $\sigma_{\max}$ arbitrarily, we can compute $\sigma_{\min}$ assuming one support vector according to the numerical precision. For simplicity, we can use one value, $\sigma_{\max}$ for computing l_2. Overall, we can compute bounds for $f(\cdot)$ as the lower bound based on l_2 and the upper bound based on u. For example, for $n = 100000$ and $\sigma_{\max} = 2^9$, we get after rounding powers to integers $\sigma_{\min} = 2^{-4}$, $M_{\min} = 2^{-19}$, $M_{\max} = 2^{10}$.

4 Method

The SGD-M-SVC returns the equivalent solutions as SGD-SVC. However, it is faster for validating different values of M. First, we run a prototype solver SGD-M-SVC with $M = M_{\max} = 2^{10}$ with provided a list of sorted M_i values as a parameter and with particular σ value. During the iteration process in the prototype solver, we store margin values defined as $m_k \equiv y_{w(k)} f_{k-1}\left(\boldsymbol{x}_{\boldsymbol{w}(\boldsymbol{k})}\right)$.

Algorithm 1. SGD-M-SVC for M_i

```
Input: $M_i$ value to check, $\sigma$, $m_k$ values, validation errors $v$, a map $(M_i, k)$ of margin
    values to indices
Output: v
1: index = $(M_i, k)$.get($M_i$) //get the index $k$ from a map $(M_i, k)$ for $M_i$
2: v = $v_k(k)$ // get a validation error for the found $k$ index from a list of $v_k$ values
```

Because sometimes it may happen that $m_k < m_{k-1}$, then we copy a value of m_{k-1} to m_k, so we have always a sorted sequence of m_k values. The size of m_k is $n - 1$ at most. We also store validation errors v_k that are updated in each iteration. The size of this list is the size of a validation set. We also update the map of M_i values to k indices during the iteration process. Then, we use a solver returning a validation error for particular M_i as specified in Algorithm 1. Given validation error, we can compare solutions for different hyperparameter values.

4.1 Computational Performance

The computational complexity of SGD-SVC based on update rules (5) is $O(n_c n)$, when n_c is the number of iterations. It is also the number of support vectors. So sparsity influences directly computational performance of training. The requirement for computing the update rule for each parameter is a linear time. The update rules (5) are computed in each iteration in a constant time. However, a linear time is needed for updating values of a decision function for all remaining examples. The procedure of finding two hyperparameter values σ and C using the cross validation, a grid search method and SGD-SVC has the complexity $O(n_c(n - n/v)v|C||\sigma| + n_c n|C||\sigma|)$ for v-fold cross validation, where $|C|$ is the number of C values to check, $|\sigma|$ is the number of σ values to check, n_c is the average number of support vectors. For each fold, we train a separate model. The first term is related to training a model. The second term is related to computing a validation error. The complexity of SGD-M-SVC is

$$O(n_{c,p}(n - n/v)v|\sigma| + n_{c,p}n|\sigma|), \tag{23}$$

where $n_{c,p}$ is the number of support vectors for a prototype solver. We removed a multiplier $|C|$ from the first term, that is related to the training complexity, and from the second term, that is related to the computation of a validation error.

5 Experiments

The M-SVC returns equivalent solutions to C-SVC. However, it is faster for validating different values of M. We validate equally distributed powers of 2 as M values from 2^{-19} to 2^{10} for integer powers, based on the analysis in the Sect. 3.1. We use our own implementation of both SGD-SVC and SGD-MSVC. We compare performance of both methods for real world data sets for binary classification. More details about data sets are on the LibSVM site ([8]). We selected all data sets from this site for binary classification. For all data sets, we scaled every feature linearly to $[0, 1]$. We use the RBF kernel in a form $K(\boldsymbol{x}, \boldsymbol{z}) = \exp(-\|\boldsymbol{x} - \boldsymbol{z}\|^2 / (2\sigma^2))$. The number of hyperparameters to tune is 2, σ and M for SGD-M-SVC, and σ and C for SGD-SVC. For all hyperparameters, we use a grid search method for finding the best values. The σ values are integer powers of 2 from 2^{-4} to 2^9. We use the procedure similar to repeated double cross validation for performance comparison. For the outer loop, we run a modified k-fold cross validation for $k = 15$, with the optimal training set size set to 80% of all examples with maximal training set size equal to 1000 examples. We limit a test data set to 1000 examples. We limit all read data to 35000. When it is not possible to create the next fold, we shuffle data and start from the beginning. We use the 5-fold cross validation for the inner loop for finding optimal values of the hyperparameters. After that, we run the method on training data, and we report results on a test set.

The observations based on experimental results are as follows. The proposed method SGD-M-SVC is about 7.6 times faster SGD-SVC (see Table 1) for binary classification, with the same generalization performance and the number of support vectors. We have 30 values of M to tune. Some authors tune value of C with fewer values. Then the effect of this speed improvement may be smaller. We generally expect the accuracy performance to degrade slowly for smaller number of values of M. We also implemented the method for multiclass classification with similar results, however we do not report it here due to space constraints. We validated also theoretical results. We check Pareto frontier every 10 iterations. The results is that the approximated Pareto frontier is generated from almost the beginning of a data set after processing 0.05% examples on average (column pS in Table 1). Approximated Pareto frontier is generated perfectly for some data sets (1.0 in a column pU), on average in 75% updates. While we check Pareto updates every 10 iterations, it may be worth to check them only for selected solutions for given M, which are distributed differently. From the practical point of view, we recommend to use SGD-M-SVC instead of SGD-SVC due to speed performance benefits.

Table 1. Experiment 1. The numbers in descriptions of the columns mean the methods: 1 - SGD-SVC, 2- SGD-M-SVC. Column descriptions: *dn* – data set, *size* – the number of all examples, *dim* – dimension of the data set, *err* – misclassification error, *sv* – the number of support vectors, *t* – average training time per outer fold in seconds, the best time is in bold (last row is a sum), *pU* – Pareto optimal solutions ratio (last row is an average), *pS* – Pareto starter ratio (last row is an average).

dn	size	dim	err1	err2	sv1	sv2	t1	t2	pU	pS
aa	34858	123	0.149	0.149	333	333	98	**9**	0.59	0.02
australian	690	14	0.146	0.146	219	219	35	**3**	0.55	0.08
avazu-app	35000	25619	0.133	0.133	673	673	90	**10**	0.57	0.01
avazu-site	35000	27344	0.211	0.211	444	444	100	**10**	0.44	0.02
cod-rna	35000	8	0.063	0.063	356	356	104	**8**	0.72	0.04
colon-cancer	62	2000	0.19	0.19	27	27	**0**	**0**	1.0	0.21
covtype	35000	54	0.292	0.292	672	672	121	**9**	0.77	0.01
criteo.kaggle2014	35000	662923	0.222	0.222	581	581	108	**11**	0.65	0.01
diabetes	768	8	0.236	0.236	334	334	40	**3**	0.42	0.06
duke	44	7129	0.222	0.222	21	21	**0**	**0**	0.87	0.31
epsilon_normalized	35000	2000	0.269	0.269	716	716	133	**11**	0.65	0.02
fourclass	862	2	0.001	0.001	556	556	46	**4**	1.0	0.01
german.numer	1000	24	0.257	0.257	411	411	67	**6**	0.55	0.03
gisette_scale	7000	4971	0.039	0.039	414	414	130	**17**	1.0	0.01
heart	270	13	0.17	0.17	100	100	6	**0**	0.76	0.12
HIGGS	35000	28	0.448	0.448	890	890	129	**10**	0.54	0.01
ijcnn1	35000	22	0.09	0.09	235	235	64	**9**	0.72	0.05
ionosphere_scale	350	33	0.081	0.082	92	89	8	**0**	0.8	0.13
kdd12	35000	54686452	0.04	0.04	819	819	74	**10**	0.39	0.01
kdda	35000	20216664	0.145	0.145	639	639	100	**12**	0.77	0.01
kddb	35000	29890095	0.144	0.144	849	849	97	**12**	0.97	0.0
kddb-raw-libsvm	35000	1163024	0.144	0.144	789	789	89	**10**	0.6	0.01
leu	72	7129	0.062	0.062	22	22	**0**	**0**	1.0	0.24
liver-disorders	341	5	0.394	0.394	202	202	10	**0**	0.34	0.07
madelon	2600	500	0.332	0.332	963	963	132	**10**	1.0	0.0
mushrooms	8124	112	0.001	0.001	842	842	103	**8**	1.0	0.01
news20.binary	19273	1354343	0.151	0.151	760	760	196	**52**	0.83	0.01
phishing	5772	68	0.059	0.059	346	346	111	**9**	0.96	0.03
rcv1.binary	35000	46672	0.064	0.064	608	608	129	**13**	0.87	0.01
real-sim	35000	20958	0.103	0.103	435	435	112	**12**	0.68	0.02
skin_nonskin	35000	3	0.011	0.011	132	132	89	**8**	0.99	0.05
sonar_scale	208	60	0.122	0.122	97	97	3	**0**	0.98	0.03
splice	2989	60	0.12	0.12	674	674	120	**9**	0.99	0.0
SUSY	35000	18	0.281	0.281	630	630	123	**9**	0.42	0.03
svmguide1	6910	4	0.04	0.04	173	173	97	**8**	0.87	0.08
svmguide3	1243	21	0.186	0.186	383	383	90	**9**	0.51	0.04
url_combined	35000	3230439	0.044	0.044	302	302	112	**10**	0.8	0.04
wa	34686	300	0.02	0.02	348	348	72	**9**	0.88	0.05
websam_trigram	35000	680715	0.044	0.044	201	201	502	**155**	0.75	0.05
websam_unigram	35000	138	0.07	0.07	260	260	101	**8**	0.71	0.03
All							3767	**495**	0.75	0.05

6 Conclusion

We proposed a novel method for SVC based on tuning margin M instead of C, with an algorithm SGD-M-SVC which improves substantially tuning time for the margin M hyperparameter compared to tuning the cost C in SGD-SVC. We provided theoretical analysis of an approximated Pareto frontier for this solver, which confirms the ability to generate solutions for different values of M during the first epoch.

Acknowledgments. The theoretical analysis of the method is supported by the National Science Centre in Poland, project id 289884, UMO-2015/17/D/ST6/04010, titled "Development of Models and Methods for Incorporating Knowledge to Support Vector Machines" and the data driven method is supported by the European Research Council under the European Union's Seventh Framework Programme. Johan Suykens acknowledges support by ERC Advanced Grant E-DUALITY (787960), KU Leuven C1, FWO G0A4917N. This paper reflects only the authors' views, the Union is not liable for any use that may be made of the contained information.

References

1. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
2. Cesa-Bianchi, N., Lugosi, G.: Prediction, learning, and games. Cambridge University Press (2006). https://doi.org/10.1017/CBO9780511546921
3. Chang, C., Chou, S.: Tuning of the hyperparameters for l2-loss svms with the RBF kernel by the maximum-margin principle and the jackknife technique. Pattern Recognition **48**(12), 3983–3992 (2015). https://doi.org/10.1016/j.patcog.2015.06.017
4. Chen, G., Florero-Salinas, W., Li, D.: Simple, fast and accurate hyper-parameter tuning in gaussian-kernel SVM. In: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14–19, 2017, pp. 348–355 (2017). https://doi.org/10.1109/IJCNN.2017.7965875
5. Cristianini, N., Shawe-Taylor, J.: An introduction to support vector machines : and other kernel-based learning methods. Cambridge University Press, 1 edn. (March 2000)
6. Gigerenzer, G., Todd, P., Group, A.R.: Simple Heuristics that Make Us Smart. Oxford University Press, Evolution and cognition (1999)
7. Hastie, T., Rosset, S., Tibshirani, R., Zhu, J.: The entire regularization path for the support vector machine. J. Mach. Learn. Res. **5**, 1391–1415 (2004)
8. Libsvm data sets (2011). www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
9. Mantovani, R.G., Rossi, A.L.D., Vanschoren, J., Bischl, B., de Carvalho, A.C.P.L.F.: Effectiveness of random search in SVM hyper-parameter tuning. In: 2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12–17, 2015. pp. 1–8 (2015). https://doi.org/10.1109/IJCNN.2015.7280664
10. Melki, G., Kecman, V., Ventura, S., Cano, A.: OLLAWV: online learning algorithm using worst-violators. Appl. Soft Comput. **66**, 384–393 (2018)

11. Orchel, M.: Knowledge-uncertainty axiomatized framework with support vector machines for sparse hyperparameter optimization. In: 2018 International Joint Conference on Neural Networks, IJCNN 2018, Rio de Janeiro, Brazil, July 8–13, 2018, pp. 1–8 (2018). https://doi.org/10.1109/IJCNN.2018.8489144
12. Sammut, C., Webb, G.I. (eds.) Encyclopedia of Machine Learning and Data Mining. Springer (2017). https://doi.org/10.1007/978-1-4899-7687-1
13. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
14. Shawe-Taylor, J., Cristianini, N.: Kernel methods for pattern analysis. Cambridge University Press, Cambridge (2004). https://doi.org/10.1017/CBO9780511809682
15. Vapnik, V.N.: Statistical Learning Theory. Wiley-Interscience, September 1998
16. Zhang, C., Liu, C., Zhang, X., Almpanidis, G.: An up-to-date comparison of state-of-the-art classification algorithms. Expert Syst. Appl. **82**, 128–150 (2017). https://doi.org/10.1016/j.eswa.2017.04.003