



# Investigating Convergence of Linear SVM Implemented in PermonSVM Employing MPRGP Algorithm

Jakub Kružík<sup>1,2(✉)</sup>, Marek Pecha<sup>1,2,3</sup>, Václav Hapla<sup>3,4</sup>, David Horák<sup>1,2,3</sup>,  
and Martin Čermák<sup>1,2,3,5</sup>

<sup>1</sup> IT4Innovations National Supercomputing Center,  
VŠB - Technical University of Ostrava, Ostrava, Czech Republic  
jakub.kruzik@vsb.cz

<sup>2</sup> Institute of Geonics CAS, Ostrava, Czech Republic

<sup>3</sup> Department of Applied Mathematics, VŠB - Technical University of Ostrava,  
Ostrava, Czech Republic

<sup>4</sup> Department of Earth Sciences, ETH Zurich, Zürich, Switzerland

<sup>5</sup> ENET Centre, VŠB - Technical University of Ostrava, Ostrava, Czech Republic

**Abstract.** This paper deals with the novel PermonSVM machine learning tool. PermonSVM is a part of our PERMON toolbox. It implements the linear two-class Support Vector Machines. PermonSVM is built on top of PermonQP (PERMON module for quadratic programming) which in turn uses PETSc. The main advantage of PermonSVM is that it is parallel. The parallelism comes from a distribution of matrices and vectors. The MPRGP algorithm, implemented in PermonQP, is used as a solver of the quadratic programming problem arising from the dual SVM formulation. The scalability of MPRGP was proven in problems of mechanics with more than billion of unknowns solved on tens of thousands of cores. Apart from the scalability of our approach, we also investigate the relations between training rate, hyperplane margin, the value of the dual functional, and the norm of the projected gradient.

**Keywords:** Support Vector Machines · SVM · PERMON  
PermonSVM · PermonQP · MPRGP · Quadratic programming · QP

## 1 Introduction

In the last two decades, the Support Vector Machines (SVMs) [7], due to their accuracy and obliviousness to dimensionality [18], have become a popular machine learning technique with applications including genetics [5], image processing [9], and weather forecasting [16]. In this paper, we are only interested in SVMs for classification. SVMs belong to supervised learning algorithms, i.e. algorithms developing a decision model from labelled training samples (training dataset). The SVM decision model is represented by the maximal-margin hyperplane, i.e. the hyperplane that separates the training dataset into two classes

with the maximal possible gap between the hyperplane and both classes. Development of the SVM decision model leads to solving a quadratic programming (QP) problem. A brief description of SVMs is given in Sect. 2.

In Sect. 3, PermonSVM [10] is introduced. PermonSVM represents one of a few open-source SVM implementations for distributed environment (it is parallelized using MPI). It focuses on solving large SVM problems on supercomputers. PermonSVM is build on top of PETSc [4] and PermonQP [11]. PermonQP is a PETSc based package for the solution of large scale QP problems. It includes implementations of several QP solvers and it can also use any of KSP [17] and TAO [14] solvers.

Section 4 describes the MPRGP [8] algorithm used for the solution of the QP arising from the SVM formulation. MPRGP is implemented in PermonQP.

Finally, numerical results are presented in Sect. 5. We investigate the convergence of SVM by looking, in each MPRGP iteration, at the training rate (percentage of correctly classified samples in the training dataset), the hyperplane margin, the value of the QP cost function, and the norm of the projected gradient (used in the stopping criterion of MPRGP). The scalability of our approach is also demonstrated.

## 2 Support Vector Machines for Classifications

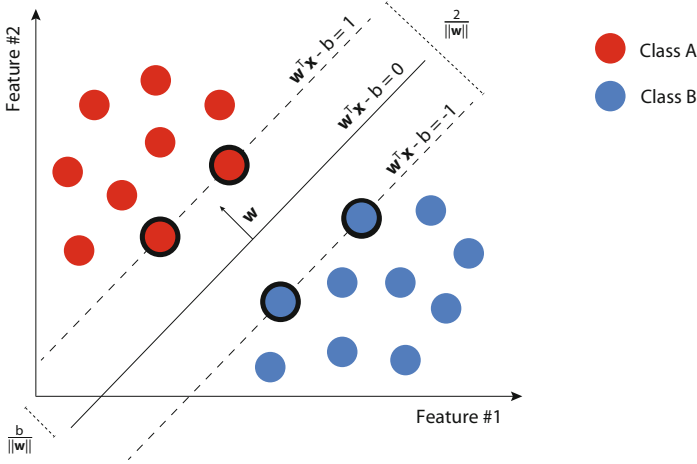
SVM is a supervised binary classifier, i.e. a classifier that decides whether a sample falls into either Class A (label 1) or Class B (label  $-1$ ) by means of a model. The model is determined from the already categorised training samples in the training phase of the classifier. Unless otherwise stated, let us assume that the training samples are linearly separable, i.e. it is possible to separate the Class A samples and the Class B samples using a hyperplane. The essential idea of the SVM classifier training is to find the *maximal-margin hyperplane* that divides the Class A from the Class B samples by the widest possible empty strip, which is called the functional margin. The samples contributing to the definition of such hyperplane are called the *support vectors* – see the circled samples lying on the dashed hyperplanes depicted in Fig. 1.

Let us denote the training samples as a set of ordered pairs such that

$$T := \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\},$$

where  $m$  is the number of samples,  $\mathbf{x}_i \in \mathbb{R}^n$  ( $n \in \mathbb{N}$  represents a number of attributes) is the  $i$ -th sample and  $y_i \in \{-1, 1\}$  denotes the label of the  $i$ -th sample,  $i \in \{1, 2, \dots, m\}$ . Let  $H$  be the *maximal-margin hyperplane*  $\mathbf{w}^T \mathbf{x} - b = 0$ , where  $\mathbf{w}$  is a normal vector;  $\frac{b}{\|\mathbf{w}\|}$  determines the offset of the hyperplane  $H$  from the origin along its normal vector  $\mathbf{w}$ . The problem of finding the hyperplane  $H$  can be formulated as a constrained optimization problem in the following hard-margin primal SVM formulation:

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w} \quad \text{s.t.} \quad y_i (\mathbf{w}^T \mathbf{x}_i - b) \geq 1. \quad (1)$$



**Fig. 1.** An example of a two-class classification problem solved by the linear hard-margin SVM.

For the case of the non-perfectly linearly separable training samples, the soft-margin SVM was designed. To handle the sensitivity of the SVM classifier to possible outliers, we introduce slack variables  $\xi_1, \xi_2, \dots, \xi_m$ , and modify the hard-margin primal SVM formulation (1) into the soft-margin primal SVM formulation

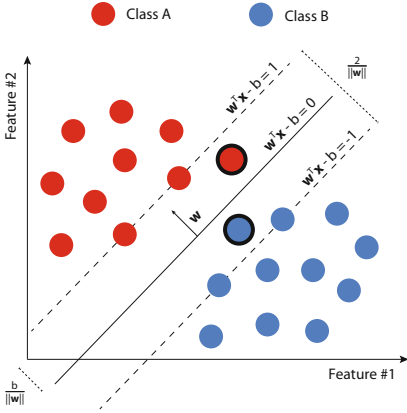
$$\min_{\mathbf{w}, b, \xi_i} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^m \xi_i \quad \text{s.t.} \quad \begin{cases} y_i (\mathbf{w}^T \mathbf{x}_i - b) \geq 1 - \xi_i, \\ \xi_i \geq 0, \end{cases} \quad (2)$$

where  $C$  is a user-specified penalty<sup>1</sup>. Higher value of  $C$  increases the importance of minimising  $\|\mathbf{w}\|$  (equivalent to maximising the margin) at the expense of satisfying the margin constraint for fewer samples. Let us refer here to several monographs mentioning the significance of  $C$ :

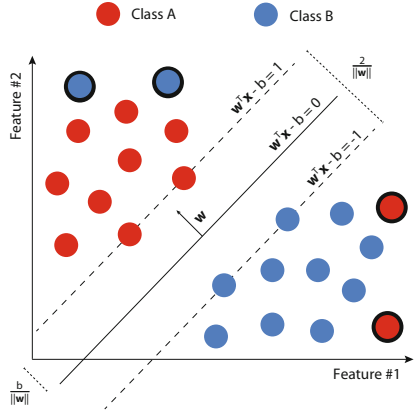
- “In the support-vector networks algorithm one can control the trade-off between complexity of decision rule and frequency of error by changing the parameter  $C, \dots$ ” [7]
- “The parameter  $C$  controls the trade off between errors of the SVM on training data and margin maximization ( $C = \infty$  leads to hard-margin SVM)” [15, p. 82].
- “... the coefficient  $C$  affects the trade-off between complexity and proportion of nonseparable samples and must be selected by the user” [6, p. 366].

We can observe that if  $0 \leq \xi_i \leq 1$ , then the  $i$ -th sample lies somewhere between the margin and their respective hyperplane (illustrated in Fig. 2); if  $\xi_i > 1$ , the  $i$ -th sample is misclassified (illustrated in Fig. 3).

<sup>1</sup> The penalty  $C$  is often called a regularization parameter in ML communities.



**Fig. 2.** Soft-margin SVM example: the encircled samples are correctly classified, but are on the wrong side of their respective hyperplane



**Fig. 3.** Soft-margin SVM example: the encircled samples are misclassified.

The primal formulation of the soft-margin SVM (2) can be simplified by exploiting the Lagrange duality with the Lagrange multipliers  $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_m]^T$ ,  $\beta = [\beta_1, \beta_2, \dots, \beta_m]^T$ . Evaluating the Karush-Kuhn-Tucker conditions, eliminating  $\beta$  and using other modifications, the problem results into the dual formulation with an inequality (box) constraint [7].

$$\min_{\alpha} \frac{1}{2} \alpha^T Y^T K Y \alpha - \alpha^T e \text{ s.t. } \mathbf{o} \leq \alpha \leq C e, \quad (3)$$

where  $e = [1, 1, \dots, 1]^T$ ,  $\mathbf{o} = [0, 0, \dots, 0]^T$ ,  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m]$ ,  $\mathbf{y} = [y_1, y_2, \dots, y_m]^T$ ,  $\mathbf{Y} = \text{diag}(\mathbf{y})$ , and  $\mathbf{K} \in \mathbb{R}^{n \times n}$  is symmetric positive semi-definite (SPS) matrix such that  $\mathbf{K} := \mathbf{X}^T \mathbf{X}$ . In the machine learning communities,  $\mathbf{K}$  is called the Gram matrix, the kernel matrix, or in the QP terminology, the Hessian.

Further, we introduce dual to primal reconstruction formulas for the normal vector

$$\mathbf{w} = \mathbf{X} \mathbf{Y} \alpha, \quad (4)$$

and the bias

$$b = \frac{1}{|I^{SV}|} \sum_{i \in I^{SV}} (\mathbf{x}_i^T \mathbf{w} - y_i), \quad (5)$$

where  $I^{SV}$  denotes the *support vector index set*, i.e.  $I^{SV} := \{i \mid \alpha_i > 0, i = 1, 2, \dots, m\}$ , and  $|I^{SV}|$  is the cardinality of  $I^{SV}$ . From the normal vector  $\mathbf{w}$  and bias  $b$ , we can easily set up the decision rule

$$\text{If } \mathbf{w}^T \mathbf{x} + b \geq 0, \text{ then } \mathbf{x} \text{ belongs to Class A, else } \mathbf{x} \text{ belongs to Class B.} \quad (6)$$

The decision rule (6) with concrete  $\mathbf{w}$  and  $b$  is also called the SVM model for the linearly separable problems.

### 3 PermonSVM: SVM Implementation Based on PETSc

PermonSVM is a new SVM tool designed to run mainly in parallel even on large supercomputers. It is written on top of PETSc [4] and PermonQP [11]. Distribution of matrices using MPI through the PETSc framework provides the parallelism.

PermonSVM provides an implementation of the two-class classification via soft-margin SVM. It implements a scalable training procedure based on a linear kernel. In the training procedure, PermonSVM takes advantage of the scalable matrix-vector product of PETSc matrices and vectors and an implicit representation of the Gram matrix (i.e. the matrix product  $\mathbf{X}^T \mathbf{X}$  is not formed), which saves memory and CPU time.

The resulting QP problem with an implicit Hessian matrix is solved by the scalable QP solvers implemented in the PermonQP package.

Additional features include fast, load-balanced cross-validation and grid search for parameter tuning, L1 and L2 loss-functions, and LIBSVM data parser. PermonSVM provides an executable for SVM classification as well as C API designed to be PETSc-like. Its typical usage is presented in Code 1.

```

MPI_comm comm = PETSC_COMM_WORLD;

PermonSVM svm;

Mat Xt, Xt_test; Vec y, y_test;

PetscInt n_examples = PETSC_DEFAULT, n_attributes = PETSC_DECIDE;
PetscInt n_test_examples = PETSC_DEFAULT, n_test_attributes = PETSC_DECIDE;
PetscInt numbering_base = 1;

/*PETSC_DEFAULT means find best C by means grid-search and cross-validation*/
PetscReal C = PETSC_DEFAULT;

char filename[PETSC_MAX_PATH_LEN] = "examples/heart_scale";
char filename_test[PETSC_MAX_PATH_LEN] = "examples/heart_scale.t";

PetscInt N_all, N_eq;

svm_file_load(filename, n_examples, n_attributes, numbering_base, Xt, y);
svm_file_load(filename_test, n_test_examples, n_test_attributes, numbering_base, Xt_test,
              y_test);

PermonSVMCreate(comm, &svm);

PermonSVMSetC(svm, C);
PermonSVMSetTrainingSamples(svm, Xt, y);

PermonSVMTrain(svm);
PermonSVMTest(svm, Xt_test, y_test, &N_all, &N_eq);

PermonSVMGetC(svm, &C);

```

Code 1: Calling PermonSVM API.

### 4 MPRGP Algorithm

MPRGP (Modified Proportioning and Reduced Gradient Projection) [8] represents an efficient algorithm for the solution of convex QP with box constraints, i.e. for

$$\min \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} - \mathbf{x}^T \mathbf{b} \quad \text{s.t.} \quad \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \quad (7)$$

where  $\mathbf{A} \in \mathbb{R}^{n \times n}$  is SPS,  $\mathbf{x}$  is the solution,  $\mathbf{b}$  is the right hand side,  $\mathbf{l}$  and  $\mathbf{u}$  is the lower respectively upper bound. The basic version can be considered as a modification of the Polyak algorithm. MPRGP combines the proportioning algorithm with the gradient projections.

Let  $\mathbf{g} = \mathbf{Ax} - \mathbf{b}$  be the gradient. Than we can define component-wise (for  $j \in \{1, 2, \dots, n\}$ ) gradient splitting which is computed after each gradient evaluation. The free gradient is defined as

$$g_j^f = \begin{cases} 0 & \text{if } x_j = l_j \quad \text{or} \quad x_j = u_j, \\ g_j & \text{otherwise.} \end{cases}$$

The reduced free gradient is

$$g_j^r = \begin{cases} 0 & \text{if } x_j = l_j \quad \text{or} \quad x_j = u_j, \\ \min\left(\frac{x_j - l_j}{\bar{\alpha}}, g_j\right) & \text{if } l_j < x_j < u_j \quad \text{and} \quad g_j > 0, \\ \max\left(\frac{x_j - u_j}{\bar{\alpha}}, g_j\right) & \text{if } l_j < x_j < u_j \quad \text{and} \quad g_j \leq 0, \end{cases}$$

where  $\bar{\alpha} \in (0, 2\|\mathbf{A}\|^{-1}]$  is used as a step length in the expansion step. The definition of the chopped gradient is

$$g_j^c = \begin{cases} 0 & \text{if } l_j < x_j < u_j, \\ \min(g_j, 0) & \text{if } x_j = l_j, \\ \max(g_j, 0) & \text{if } x_j = u_j. \end{cases}$$

Finally, the projected gradient is defined as  $\mathbf{g}^P = \mathbf{g}^f + \mathbf{g}^c$ . Its norm decrease is the natural stopping criterion of the algorithm.

Let the projection onto the feasible set  $\Omega = \{\mathbf{x} : \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$  be defined as

$$P_\Omega(\mathbf{x})_j = \min(u_j, \max(l_j, x_j)).$$

Now we have all the necessary ingredients to summarise MPRGP in Algorithm 1.

## 5 Numerical Experiments

In this section, we show the scalability of our approach as well as what the relations between the training rate (percentage of correctly classified samples), the hyperplane margin, the value of the dual functional, and the norm of the projected gradient are. The hyperplane (given by  $\mathbf{w}$  and  $b$ ) is computed in each iteration of MPRGP. Using the computed hyperplane, we can evaluate the training rate and the margin ( $2/\|\mathbf{w}\|$ ). The value of the dual functional is trivially computed from the gradient which is available in every MPRGP iteration. The computation of these metrics is relatively expensive. Therefore, it is by default disabled, but it can be toggled by a command line switch. The decrease of the

**Algorithm 1.** MPRGP

---

**Input:**  $\mathbf{A}$ ,  $\mathbf{x}^0 \in \Omega$ ,  $\mathbf{b}$ ,  $\Gamma > 0$ ,  $\bar{\alpha} \in (0, 2\|\mathbf{A}\|^{-1}]$

- 1  $\mathbf{g} = \mathbf{A}\mathbf{x}_0 - \mathbf{b}$ ,  $\mathbf{p} = \mathbf{g}^f(\mathbf{x}^0)$ ,  $k = 0$
- 2 **while**  $\|\mathbf{g}^P(\mathbf{x}^k)\|$  is not small:
- 3   **if**  $\|\mathbf{g}^c(\mathbf{x}^k)\|^2 \leq \Gamma^2 \mathbf{g}^r(\mathbf{x}^k)^T \mathbf{g}^f(\mathbf{x}^k)$ :
- 4      $\alpha_f = \max\{\alpha_{cg} : \mathbf{x}^k - \alpha_{cg}\mathbf{p}\}$
- 5      $\alpha_{cg} = \mathbf{g}^T \mathbf{p} / \mathbf{p}^T \mathbf{A} \mathbf{p}$
- 6     **if**  $\alpha_{cg} < \alpha_f$ :
- 7       // CG step
- 8        $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{cg}\mathbf{p}$
- 9        $\mathbf{g} = \mathbf{g} - \alpha_{cg}\mathbf{A}\mathbf{p}$
- 10        $\beta = \mathbf{g}^f(\mathbf{x}^{k+1})^T \mathbf{A}\mathbf{p} / \mathbf{p}^T \mathbf{A} \mathbf{p}$
- 11        $\mathbf{p} = \mathbf{g}^f(\mathbf{x}^{k+1}) - \beta\mathbf{p}$
- 12     **else:**
- 13       // Expansion step
- 14        $\mathbf{x}^{k+\frac{1}{2}} = \mathbf{x}^{k+1} - \alpha_f\mathbf{p}$
- 15        $\mathbf{g} = \mathbf{g} - \alpha_f\mathbf{p}$
- 16        $\mathbf{x}^{k+1} = P_\Omega(\mathbf{x}^{k+\frac{1}{2}} - \bar{\alpha}\mathbf{g}^f(\mathbf{x}^{k+\frac{1}{2}}))$
- 17        $\mathbf{g} = \mathbf{A}\mathbf{x}^{k+1} - \mathbf{b}$
- 18        $\mathbf{p} = \mathbf{g}^f(\mathbf{x}^{k+1})$
- 19     **else:**
- 20       // Proportioning step
- 21        $\alpha_{cg} = \mathbf{g}^T \mathbf{g}^c(\mathbf{x}^k) / \mathbf{g}^c(\mathbf{x}^k)^T \mathbf{A} \mathbf{g}^c(\mathbf{x}^k)$
- 22        $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_{cg}\mathbf{g}^c(\mathbf{x}^k)$
- 23        $\mathbf{g} = \mathbf{g} - \alpha_{cg}\mathbf{A}\mathbf{g}^c(\mathbf{x}^k)$
- 24        $\mathbf{p} = \mathbf{g}^f(\mathbf{x}^{k+1})$
- 25      $k = k + 1$

**Output:**  $\mathbf{x}^k$

---

projected gradient norm is natural as well as the default stopping criterion of MPRGP.

Relations mentioned above are demonstrated on a dataset from the ExCAPE project [1] and also on the URL [13] dataset. The ExCAPE project aim is to predict compound bioactivity for the pharmaceutical industry. The tested dataset is related to Pfam protein database. It contains 226.1 thousand samples with 2048 attributes. The URL dataset relates to the detecting of malicious websites involved in criminal scams. It contains 2.4 million samples with 3.23 million attributes. The dataset is publicly available on LIBSVM datasets websites [2] in the LIBSVM format.

The experiments were run on the Salomon supercomputer [3] at IT4Innovations. Salomon consists of 1008 compute nodes. Each compute node contains two 2.5 GHz, 12-core Intel Xeon E5-2680v3 (Haswell) processors and 128 GB of memory. Compute nodes are interconnected by InfiniBand FDR56. Salomon has the peak performance of 2 petaFLOPS.

The initial guess was set to the zero vector. The relative norm of projected gradient (i.e. the ratio of the projected gradient norm and the right-hand side norm) being smaller than  $1e-1$  was used as the stopping criterion in all numerical experiments. From our experience, while the tolerance is exceptionally high, it is more than adequate to find a good solution. This is illustrated by the following results.

In Tables 1 and 2 and in accompanying Figs. 4 and 5 the impact of the parameter  $C$  is shown. We report the maximal achieved training rate (Max rate) and the training rate upon the solver convergence (Converged rate) as well as the number of iterations needed to reach these rates.

Looking at the results of the ExCAPE dataset (Table 1 and Fig. 4), except for  $C = 1e-5$ , the difference between the maximal rate and converged rate ranges from 0.4 to 0.63%. Also note, that the best rate is achieved after relatively few iterations. To actually satisfy the convergence criterion it is necessary to do between 2.6 and 8 times as many iterations needed to get the maximum rate.

**Table 1.** ExCAPE dataset: comparison of the maximal achieved training rate (and iteration it occurred) with training rate obtained after solver converged (and again iteration this occurred).

C	1e-5	1e-4	1e-3	1e-2	1e-1
Max rate	76.8712	82.4114	84.3426	84.5243	84.6941
Converged rate	73.1472	81.8946	83.9301	84.1291	84.0641
Max rate iter	4	58	756	8415	9976
Converged rate iter	24	277	2868	22240	80306

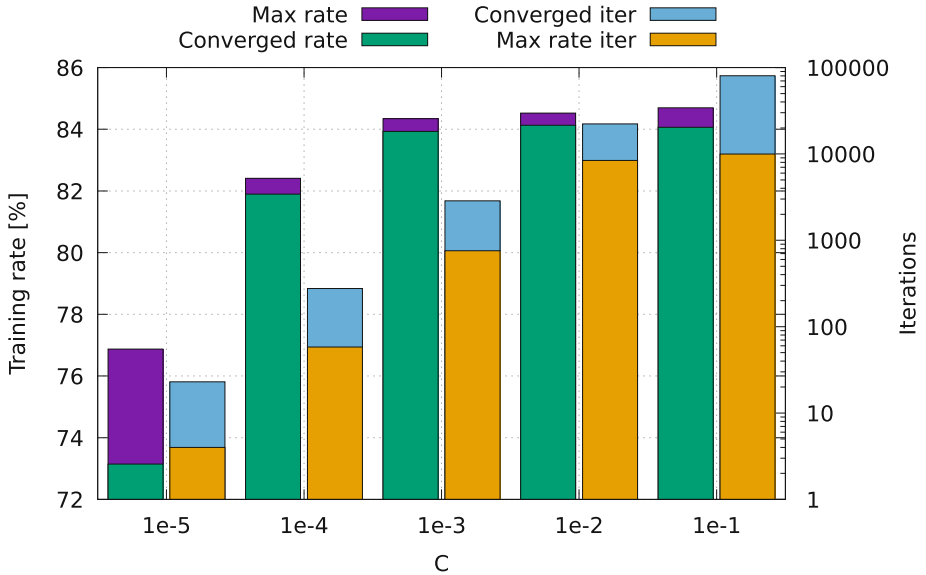
The differences are much smaller for the URL dataset (Table 2 and Fig. 5). Again, we ignore in the following discussion the results for the smallest parameter  $C$ , because the solution is not good enough. The rate attained after the convergence is between 0.04 and 0.13% lower than the maximum rate. However, to reach the best rate it is necessary to do only from 50 to 70% of the number iterations needed to achieve convergence.

Further, we analyse the training rate, margin, value of dual functional, and norm of the projected gradient on the per iteration basis. The results are shown for the ExCAPE dataset in Figs. 6 and 7 for  $C = 1e-3$ , and for the URL dataset in Figs. 8 and 9 for  $C = 1e-5$ .

The MPRGP algorithm guarantees the decrease of the functional value in every iteration. In these examples, the norm of the projected gradient decreases monotonously as well. However, this is not guaranteed, and in fact, we observed high fluctuations for the ExCAPE dataset with larger values of the  $C$  parameter.

More interestingly, the training rate peaks after a relatively small number of iterations. The training rate also oscillates. It is barely noticeable in these examples. However, we observed very severe oscillation for the ExCAPE dataset





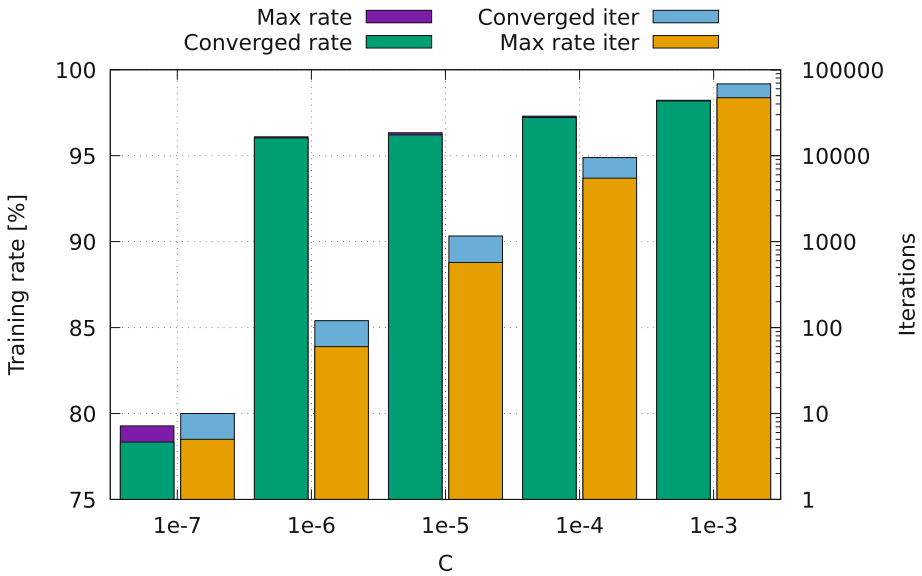
**Fig. 4.** ExCAPE dataset: comparison of the maximal achieved training rate (and iteration it occurred) with training rate obtained after solver converged (and again iteration this occurred).

**Table 2.** URL dataset: comparison of the maximal achieved training rate (and iteration it occurred) with training rate obtained after solver converged (and again iteration this occurred).

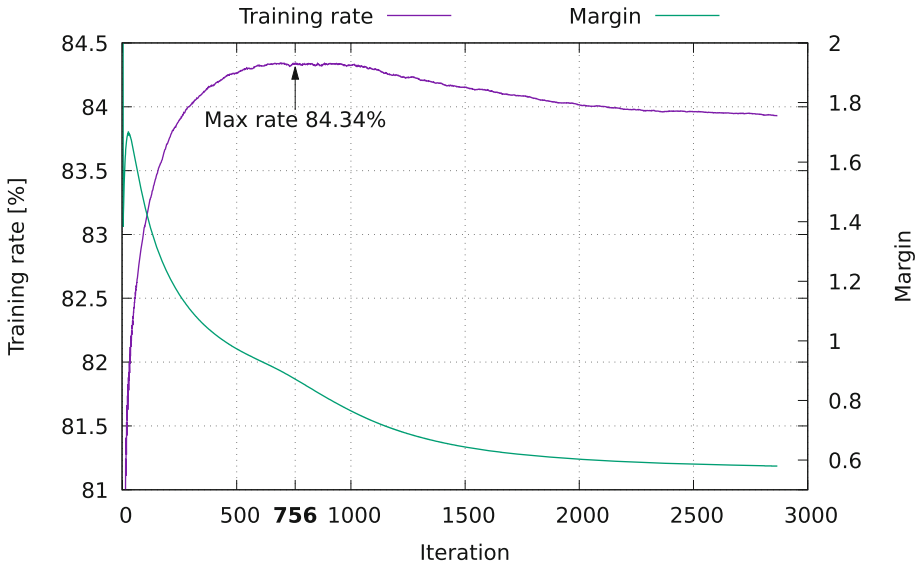
C	1e-7	1e-6	1e-5	1e-4	1e-3
Max rate	79.2815	96.0976	96.3368	97.2961	98.2226
Converged rate	78.3386	96.0374	96.2101	97.2186	98.1828
Max rate iter	5	60	571	5469	47329
Converged rate iter	10	120	1161	9525	68548

with larger values of the parameter  $C$ . The rate difference between consecutive iterations was sometimes over 17%. Also, notice that the hyperplane margin starts to decrease after relatively few iterations.

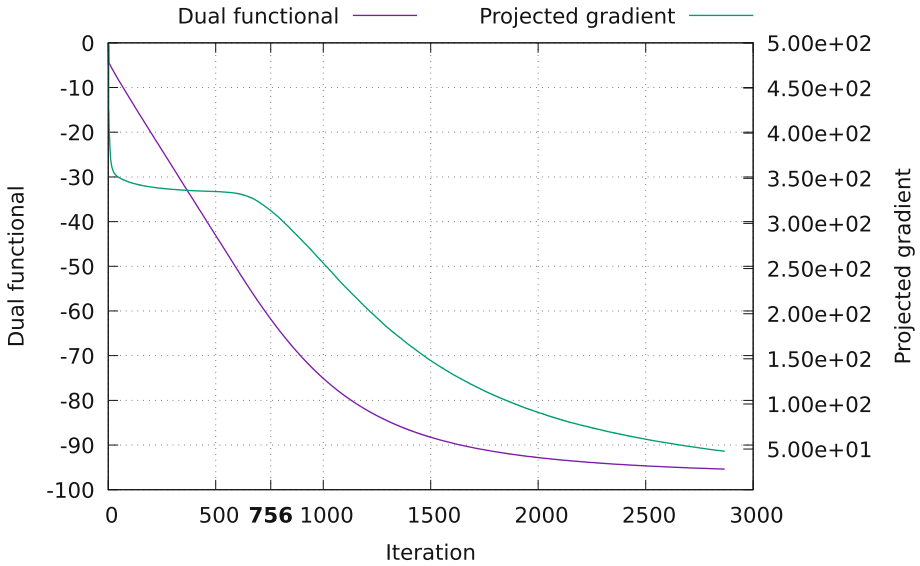
The decreasing value of the dual functional and that it is negative means, thanks to positive semi-definiteness of the Hessian and the positiveness of  $\alpha$ , that the dual solution  $\alpha$ , on the whole, increases. Meaning, that the satisfaction of the first constraint in (2) improves. The margin generally has a decreasing tendency, i.e. the norm of  $\mathbf{w}$  increases, suggesting (from (2)) that the sum of distances of samples from their respective hyperplanes decreases as well. Note, that this does not tell us anything about the training rate. In fact, we can see that improving this sum can lead to decrease in the training rate.



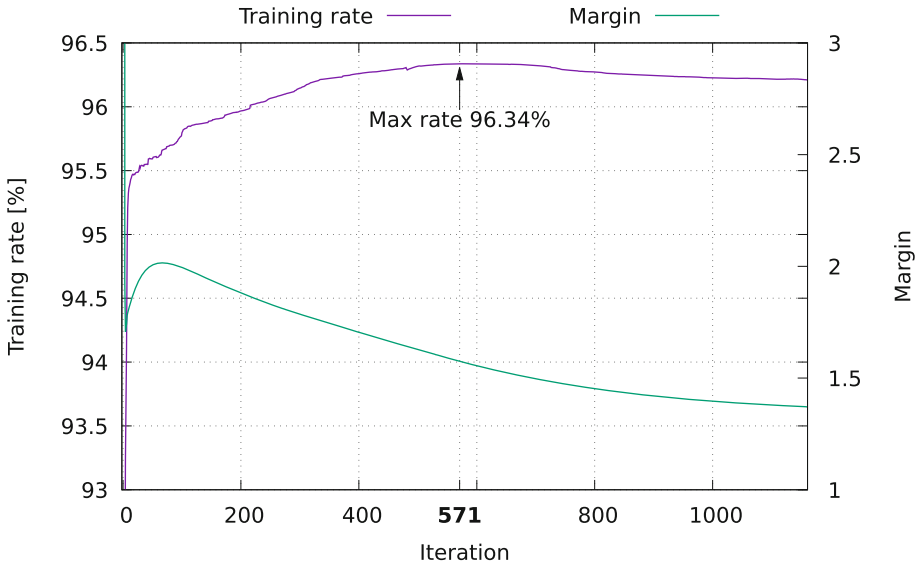
**Fig. 5.** URL dataset: comparison of the maximal achieved training rate (and iteration it occurred) with training rate obtained after solver converged (and again iteration this occurred).



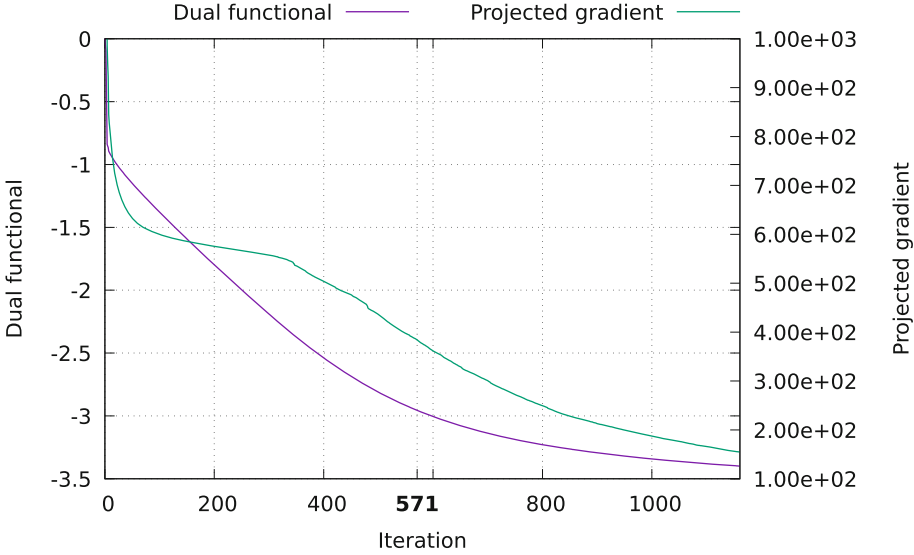
**Fig. 6.** ExCAPE dataset,  $C = 1e-3$ : the relation of the training rate and margin on the iteration number. The iteration number given in bold is where the maximum training rate was reached.



**Fig. 7.** ExCAPE dataset,  $C = 1e-3$ : the relation of the value of dual functional and the norm of the projected gradient on the iteration number. The iteration number given in bold is where the maximum training rate was reached.



**Fig. 8.** URL dataset,  $C = 1e-5$ : the relation of the training rate and margin on the iteration number. The iteration number given in bold is where the maximum training rate was reached



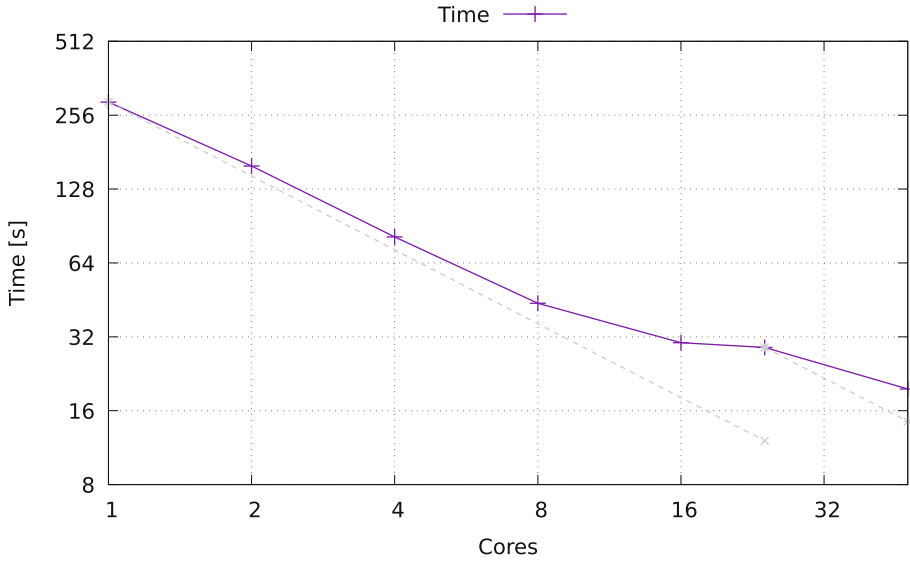
**Fig. 9.** URL dataset,  $C = 1e-5$ : the relation of the value of dual functional and the norm of the projected gradient on the iteration number. The iteration number given in bold is where the maximum training rate was reached.

**Table 3.** ExCAPE dataset,  $C = 1e-3$ : MPRGP strong parallel scalability

Number of cores	1	2	4	8	16	24	48
Time	289.36	158.87	81.75	43.86	30.29	28.95	19.63

The default stopping criterion of MPRGP based on the norm of the projected gradient seems ill-suited for SVM. It appears, despite the large tolerance, that the problems are solved unnecessary accurately. However, it is relatively easy to implement and use stopping criteria commonly used in SVM solvers. Looking only at the training rate, it seems that MPRGP can obtain a reasonable solution very quickly in few iterations.

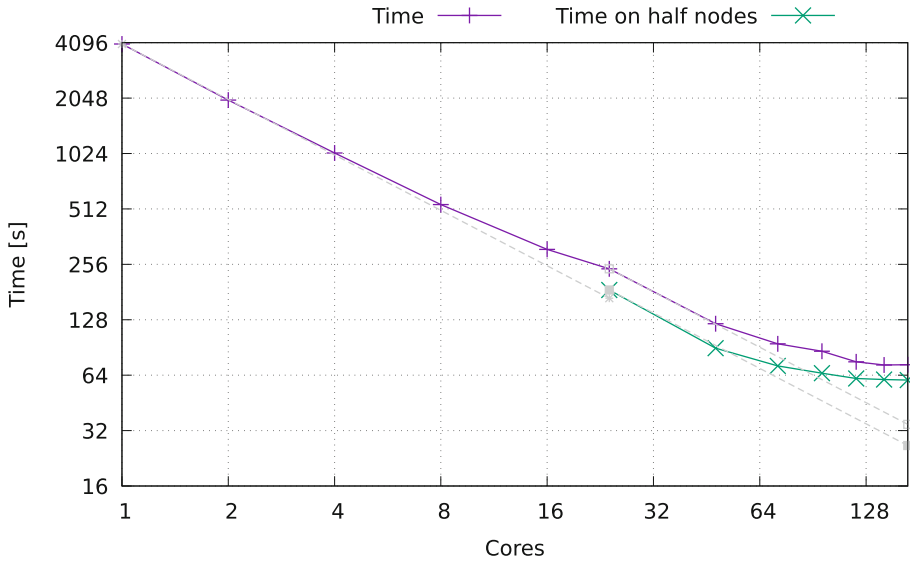
Finally, we demonstrate the strong scalability of our solver. The big advantage of PermonSVM is that it can run in a distributed environment. Moreover, the MPRGP algorithm was proven to be scalable; for example, it can solve problems of mechanics with more than billion of unknowns on tens of thousands of cores [12]. The scalability results for the ExCAPE dataset are summarized in Table 3 and Fig. 10. The results for the URL dataset are presented in Table 4 and Fig. 11. The scalability is essentially the same as the scalability of the sparse matrix-vector product. This operation is memory bounded as illustrated by “Time on half nodes” results on the URL dataset. In this case, only half of the cores on each node are used (6 cores on each socket). This MPI rank placement significantly increases memory throughput. Thanks to this, the scaling is almost perfect up to 48 cores, after which the size of the distributed dataset starts to be too small to utilise the cores fully.



**Fig. 10.** ExCAPE dataset,  $C = 1e-3$ : MPRGP strong parallel scalability

**Table 4.** URL dataset,  $C = 1e-5$ : MPRGP strong parallel scalability

Number of cores	1	2	4	8	16	24	48	72	96	120	144	168
Time	4025.91	1996.99	1030.58	540.32	308.59	242.10	121.80	94.74	86.36	75.67	72.70	72.98
Time on half nodes	-	-	-	-	-	185.74	89.74	71.87	65.60	61.39	60.55	60.22



**Fig. 11.** URL dataset,  $C = 1e-5$ : MPRGP strong parallel scalability

## 6 Conclusion

We have introduced a novel, open-source PermonSVM machine learning tool employing scalable quadratic programming algorithms implemented in the PermonQP module. PermonSVM provides an implementation of the two-class classification via soft-margin SVM. Currently, it supports only a linear kernel. As a default, it uses the MPRGP algorithm for the solution of QP obtained from the dual SVM formulation.

We demonstrated the behaviour of the MPRGP algorithm on a dataset from the ExCAPE project as well as on the URL dataset. We analysed the relations between the training rate, the hyperplane margin, the value of the dual functional and the norm of the projected gradient on the per iteration basis. We note that the algorithm achieves a good training rate after relatively few iterations. The scalability of our approach was also demonstrated.

Further work will include implementation of a better stopping criterion and nonlinear kernels.

**Acknowledgments.** This work was supported by the Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science (LQ1602), and from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center (LM2015070); by the internal student grant competition project SGS No. SP2018/165; by projects LO1404: Sustainable development of CENET, and CZ.1.05/2.1.00/19.0389: Research Infrastructure Development of the CENET; and by the Czech Science Foundation (GACR) projects no. 15-18274S and 17-22615S. We would also like to acknowledge partners in the ExCAPE project for providing us with training datasets related to the Pfam protein database.

## References

1. ExCAPE: exascale compound activity prediction. <http://www.excape-h2020.eu>
2. LIBSVM data: classification, regression, and multi-label. <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>
3. IT4Innovations: Salomon cluster documentation - hardware overview. National Supercomputing Center, VSB-Technical University of Ostrava (2017). <https://docs.it4i.cz/salomon-cluster-documentation/hardware-overview>
4. Balay, S., Abhyankar, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Rupp, K., Smith, B.F., Zhang, H.: PETSc - Portable, Extensible Toolkit for Scientific Computation. <http://www.mcs.anl.gov/petsc>
5. Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares Jr., M., Haussler, D.: Knowledge-based analysis of microarray gene expression data by using support vector machines. *Proc. Nat. Acad. Sci. U.S.A.* **97**(1), 262–267 (2000)
6. Cherkassky, V., Mulier, F.M.: *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press, Hoboken (2007)
7. Cortes, C., Vapnik, V.: Support-vector networks. *Mach. Learn.* **20**(3), 273–297 (1995)

8. Dostál, Z.: Optimal Quadratic Programming Algorithms, with Applications to Variational Inequalities. SOIA, vol. 23. Springer, New York (2009). <https://doi.org/10.1007/b138610>
9. Foody, G.M., Mathur, A.: The use of small training sets containing mixed pixels for accurate hard image classification: training on mixed spectral responses for classification by a SVM. *Remote Sens. Environ.* **103**(2), 179–189 (2006)
10. Hapla, V., Horák, D., Pecha, M.: PermonSVM (2017). <http://permon.it4i.cz/permonsvm.htm>
11. Hapla, V., Horák, D., Čermák, M., Kružík, J., Pospíšil, L., Sojka, R.: PermonQP (2015). <http://permon.it4i.cz/qp/>
12. Horak, D., Dostal, Z., Hapla, V., Kruzik, J., Sojka, R., Cermak, M.: Projector-less TFETI for contact problems: preliminary results. In: *Civil-Comp Proceedings*, vol. 111 (2017)
13. Ma, J., Saul, L., Savage, S., Voelker, G.: Identifying suspicious URLs: an application of large-scale online learning, pp. 681–688 (2009). Cited By 173
14. Munson, T., Sarich, J., Wild, S., Benson, S., McInnes, L.C.: TAO users manual. Technical report ANL/MCS-TM-322. Argonne National Laboratory (2015). <http://tinyurl.com/tao-man>
15. Rychetsky, M.: Algorithms and Architectures for Machine Learning Based on Regularized Neural Networks and Support Vector Approaches (*Berichte Aus Der Informatik*). Shaker Verlag GmbH, Herzogenrath (2001)
16. Shi, J., Lee, W.J., Liu, Y., Yang, Y., Wang, P.: Forecasting power output of photovoltaic systems based on weather classification and support vector machines. *IEEE Trans. Ind. Appl.* **48**(3), 1064–1069 (2012)
17. Smith, B.F., et al.: PETSc users manual. Technical report ANL-95/11 - Revision 3.5. Argonne National Laboratory (2016). <http://tinyurl.com/petsec-man>
18. Vishnu, A., Narasimhan, J., Holder, L., Kerbyson, D., Hoisie, A.: Fast and accurate support vector machines on large scale systems. In: *2015 IEEE International Conference on Cluster Computing*, pp. 110–119, September 2015