The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLVIII-2/W9-2025
ISPRS Intl. Workshop “Photogrammetric and computer vision techniques for environmental and infraStructure monitoring, Biometrics and Biomedicine”
PSBB25 , 9–11 June 2025, Moscow, Russia

High-Performance Large-Scale SVM-based Multiclass Classification

Mikhail Yu. Kurbakov, Valentina V. Sulimova

Laboratory of Cognitive Technologies and Simulation Systems, Tula State University,
Lenine Ave. 92, Tula, Russia, 300012 - muwsik@mail.ru, vsulimova@yandex.ru

Keywords: High-performance computing, Multiclass Classification, SVM, Large training sets, Image classification.

Abstract

A typical characteristic of modern applied multiclass classification problems is the large scale. It significantly complicates or even makes it impossible an application of such popular, convenient and well-interpreted method as Support Vector Machines (SVM), which is well-proven for small-size classification problems. In this connection the actual problem is to increase SVM’s computational performance. The Double-Layer Smart Sampling SVM (DLSS-SVM) method allows to reduce the training time of multiclass SVM via double using the smart sampling technique. This paper proposes the high-performance version of DLSS-SVM (HP-DLSS-SVM). It is based on two-level parallel computing scheme, which exploits useful DLSS-SVM properties and computing system capabilities more fully. Experimental investigation of the proposed HPDLSS-SVM method was made on three large handwritten digit images data sets of different size. Experiments show that the proposed approach allows to essentially decrease training and testing times and at that to maintain the obtained recognition accuracy close to the best.

1. Introduction

Many applied problems from different socially significant areas, such as drug design, medical images analysis, video surveillance and information security systems and others are formulated as the multiclass classification problem. This problem in itself is a classic one, but the volume of data that should be processed is growing rapidly. As a result, the requirements for the computational performance of methods for solving them are becoming increasingly stringent. This leads to the need for a significant revision of existing methods.

In particular, the training time of such popular, convenient and well-interpreted method as Support Vector Machines (SVM) (Vapnik, 1995; Benfenati et al., 2023), which is well-proven for small-size classification problems nonlinearly depends on the number of objects. It significantly complicates or even makes impossible its application for large-scale tasks.

Many authors try to accelerate computations using parallel and distributed data processing technologies (Dekel et al., 2012; Niu et al., 2011; Agarwal et al., 2011; Zhao et al., 2011), including expression in MapReduce terms (Chu et al., 2006; Rizzi, 2016; Sleeman et al., 2021) and GPU applications (Wen et al., 2018). However, since the original sequential algorithms have an iterative nature and between-iteration data dependencies, the efficiency of parallelization is insufficient: existing parallel and distributed processing only mitigates, but does not solve, the problem of high computational complexity. At that attempts to avoid data dependencies lead to a noticeable decrease in the decision quality (You et al., 2015).

This paper is based on the new enough Dual-Layer Smart Sampling SVM (DLSS-SVM) method proposed by us (Kurbakov et al., 2024), which actually opens new possibilities for solving large scale multiclass SVM problems. The smart sampling conception consists in to form samples from the initial large training set by selecting objects that are support ones as a result of training in random samples. Such a way to form a sample in contrast to traditional random samples (Chauhan et al., 2018; Byrd et al., 2016; Makarova et al., 2019) allow easy to exclude from the consideration those objects that are far from an optimal discriminant hyperplane and not affect the resulting decision rule. The DLSS-SVM method consists in double applying the smart sampling technique. This allows, even with sequential implementation, to significantly reduce the computation time of the training stage compared to existing open access methods and at the same time maintain recognition accuracy close to the best (Kurbakov et al., 2024).

At the same time, it is easy to see that the DLSS-SVM method has internal parallelism which has not yet been used.

2. Contribution of the Paper

This paper proposes a high-performance version of DLSS-SVM, called High Performance Double-Layer Smart Sampling SVM (HP-DLSS-SVM). It is based on two-level parallel computing scheme, which exploits useful method properties and computing system capabilities more fully. Experiments with three large image data sets shows that the proposed method allows to essentially decrease training and testing times for large-scale multiclass SVM problem saving a near the best accuracy.

3. Sequential Smart Sampling SVM for Multiclass Classification

3.1 Multiclass and Binary Classification Problems

Let Ω^* be a set of all possible objects ω of some kind, each of which can be presented by n-length real-valued feature vector $\mathbf{x}(\omega) = [x_1, ..., x_n]$. We suppose that some finite subset of objects $\Omega = \{\omega_j, j = 1, ..., N\} \subset \Omega^*$ is oobservable through its representations $\mathbf{x}_j = \mathbf{x}(\omega_j), j = 1, ..., N$ jointly with class labels $y_j = y(\omega_j) \in \{1, ..., m\}, j = 1, ..., N$, $m \geq 2$ and

Publisher's note: Copernicus Publications has not received any payments from Russian or Belarusian institutions for this paper.

This contribution has been peer-reviewed.
https://doi.org/10.5194/isprs-archives-XLVIII-2-W9-2025-175-2025 | © Author(s) 2025. CC BY 4.0 License.

constitutes the training set $[\Omega, Y] = \{[\omega_j, y_j], j = 1,..., N\}$. The task is to make a decision function that for any new object $\omega \notin \Omega$ will estimate the unknown class-label $\hat{y}(X_\omega)$.

3.2 One-versus-the-Rest Multiclass Classification Strategy

Following (Kurbakov et al., 2024) to solve the initial multiclass classification problem we use the One-versus-the-Rest strategy (Bishop, C. M., 2006). It consists of fitting one binary classifier for each class. So, for m classes, m classifiers must be built. For each classifier, the main class is fitted against all other classes, and thus the full training set is used each time. The resulting class label we select by the maximum value of class probability.

3.3 Smart Sampling Conception

The Smart Sampling conception was originally proposed by us in (Makarova et al., 2020) to accelerate convergence of the Kernel-Based Mean Decision Rules method for binary SVM. The idea consists to form a smart sample only from support objects that are obtained as a result of training for small simple random samples. Binary SVM decision is built in form of the optimal discriminant hyperplane that is depends only on support objects. Excluding any of non-support object from the training set does not change the SVM decision. Objects selected in accordance with the smart sampling technique are good enough candidates to be support objects in the initial large problem for the full training set. So, the smart sampling allows to intellectually reduce large training set and so to decrease the training time. Smart sampling technique has shown its effectiveness for binary (Makarova et al., 2020), multiclass (Kurbakov et al., 2024) one-class SVM (Kurbakov et al., 2023). More detailed description of this conception can be found at (Kurbakov et al., 2024).

3.4 Double-Layer Smart Sampling SVM (DLSS-SVM) with Early Stopping

The main idea of the DLSS-SVM consists in double applying the smart sampling technique. Namely, it assumes constructing a number of first-layer (L1) smart samples, to train in them and to form the second-layer (L2) smart sample from support objects that are computed at the L1-training. The model, which is obtained as the result of training on the L2 smart sample is considered as a final decision rule.

It should be noted that a smart sample formation speed at both of levels and appropriate smart sample size are data-dependent. Thus, to avoid the situation, when the specified smart sample size value is too large and cannot be reached or the smart sample fills slowly (a small number of new support objects are added) the early stopping criterion is applied that allows to adapt to the data and to reduce the execution time.

The formal description of forming the double-layer smart sample with early stopping is given by Algorithm 1 and Algorithm 2.

Algorithm 1. First-layer Smart Sample with early stopping

Parameters:
SSize1 - desired size of first-layer smart sample
RSize – size of random samples
Δ_1 - the threshold for early stopping

1: set the 1-layer smart sample as empty $\Omega_{smart}^{L1} = \varnothing$ and its actual size as zero $SSize1_{act} = 0$
2: take small random sample $\Omega_{rnd} \subset \Omega$ of size *RSize*
3: train SVM with $[\Omega, Y]_{rnd}$ to obtain support objects $\Omega_{\text{sup}} \subseteq \Omega_{rnd}$
4: update the smart sample $\Omega_{smart}^{L1} = \Omega_{smart}^{L1} \cup \Omega_{\text{sup}}$
5: if the smart sample size is not enough $|\Omega_{smart}| < SSize1$ and early stopping criterion $|\Omega_{smart}^{L1}| - SSize1_{act} \geq \Delta_1$ does not hold true then upgrade $SSize1_{act} = |\Omega_{smart}^{L1}|$ and repeat steps 2-5.

Algorithm 2. Second-layer Smart Sample with early stopping

Parameters:
SSize1, *SSize2* - desired sizes of first-layer and second-layer smart samples, respectively
RSize – size of random samples
Δ_1, Δ_2 - thresholds for early stopping at the 1-st and 2-nd levels, respectively

1: set the 2-layer smart sample as empty $\Omega_{smart}^{L2} = \varnothing$ and its actual size as zero $SSize2_{act} = 0$.
2: form the 1-layer smart sample $\Omega_{smart}^{L1}(SSize1, Rsize, \Delta_1) \subset \Omega$ (Algorithm 1)
3: train SVM with $[\Omega, Y]_{smart}^{L1}$ to obtain support objects $\Omega_{\text{sup}}^{L1} \subseteq \Omega_{smart}^{L1}$
4: update the smart sample $\Omega_{smart}^{L2} = \Omega_{smart}^{L2} \cup \Omega_{\text{sup}}^{L1}$
5: if the smart sample size is not enough $|\Omega_{smart}^{L2}| < SSize2$ and early stopping criterion $|\Omega_{smart}^{L2}| - SSize2_{act} \geq \Delta_2$ does not hold true then upgrade $SSize2_{act} = |\Omega_{smart}^{L2}|$ and repeat steps 2-5.

It should be noted that in contrast to equal-sized random samples, the resulted smart samples can be of different size. The desired values *SSize1* and *SSize2* actually are landmarks, but each of them can be slightly exceeded in the process of the smart sample updating (step 4 of the Algorithm 2). Besides, actual smart sample size can be essentially less in contrast to the specified value due to the early stopping (step 5 of the Algorithm 2).

4. High-Performance DLSS-SVM (HP-DLSS-SVM)

4.1 Upper Level Parallel Computing Scheme

In accordance with the description in Section 3, DLSS-SVM solves the N-class SVM recognition problem by reducing it to N binary one-versus-the-rest SVM problems. It is evident that all N binary recognition problems are independent and can be solved in parallel. The respective parallel computing scheme is extremely simple and easily realized.

But the degree of parallelism is limited by the number of classes in the original multiclass problem, which can be significantly less than the number of available processors. In this connection we consider additional level of parallelism.

4.2 Lower Level Parallel Computing Scheme

Additional resources of inner parallelism of DLSS-SVM is in the stages of forming smart samples of both layers. By definition, each smart sample consists of support objects that are obtained as a results of training for some subsamples (random samples to form L1-smart samples or L1-smart samples to form L2-smart samples). So, all smart samples of the same layer can be formed in parallel manner. Since larger parallel parts are more preferable, this work exploits the parallelism of L2-layer smart samples.

However, the respective approach has the problem, which consists in unknown number of L1-smart samples that are required to form a L2-smart sample. It varies for tasks (even across different binary SVM tasks within single multiclass ones) and cannot be determined a priory.

To overcome this obstacle, we introduce an additional parameter P_L, which defines the number of L1-smart samples for parallel computing. If the total number of unique support objects obtained from P_L L1-smart samples is greater than or equal to the desired L2-smart sample size, the computing of the respective smart sample is stopped else a new portion of P_L samples is generated and used to obtain additional objects. The process continues till the desired L2-smart sample size will be reached. It should be noted the optimal value of the parameter P_L is data-dependent and can be different for different tasks.

The respective lower-level parallel computing scheme is presented at the figure 1.

4.3 Software Implementation of HPDLSS-SVM

It should be noted the proposed parallel scheme can be implemented in different ways depending on the computing system architecture and programming tools.

In this paper we deal with a shared memory system and the Python programming language, since the sequential version (DLSS-SVM) is implemented in it. These peculiarities have own advantages and disadvantages.

First of all, it is not practical to use multithreading at the lower level of a parallel scheme, since only one thread can process Python at a time due to the GIL(Global Interpreter Lock). As a result, multiprocessing is used at both of levels of the presented parallel scheme via the python *multiprocessing* package.

Processes, unlike threads, operate in different address spaces, so when they are used to organize parallel computing, the question of organizing interprocess communication arises, as well as the question of data duplication, which is especially relevant in connection with the large scale of the tasks being solved.

Both of these problems are solved in this work through the shared memory mechanism (using *multiprocessing. shared_memory* module) in the case if all data of one task fits into RAM and through the memory mapping mechanism (using class *mmap* of the respective python module) if RAM is not enough. The memory mapping mechanism put data on disk and maps it into process memory by parts. Thus, it removes the limitation on the amount of RAM, but is slower compared to the shared memory mechanism.

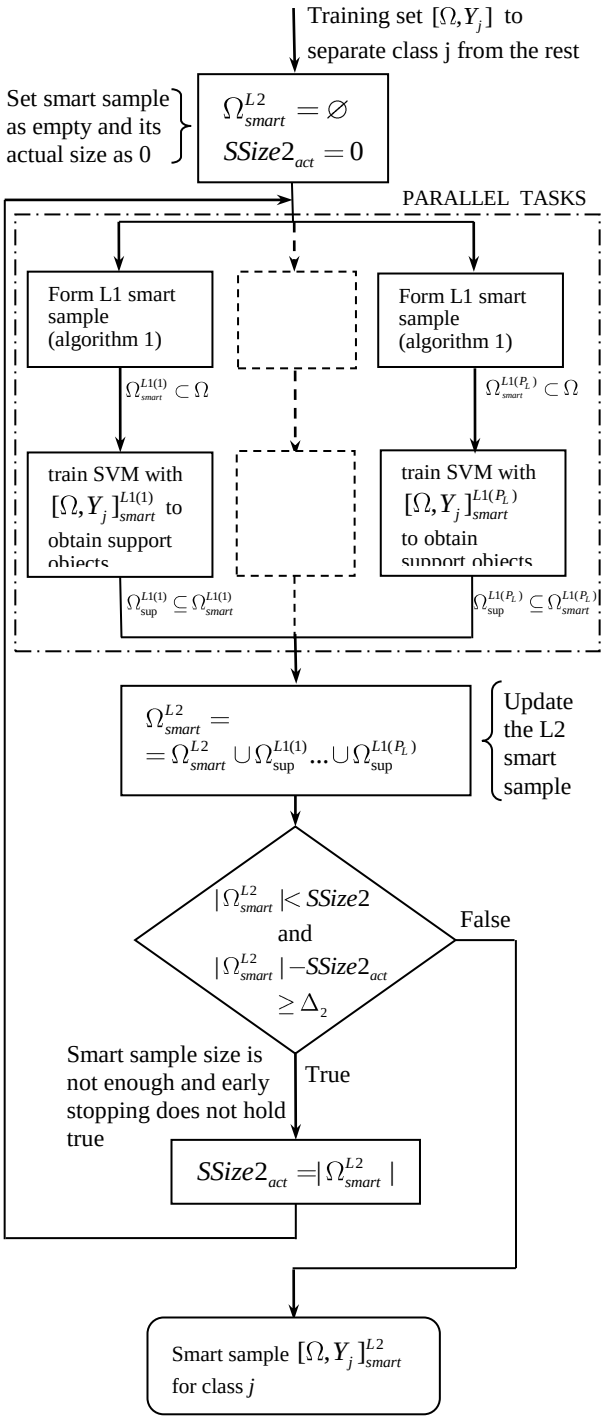

Figure 1. Lower level parallel scheme to form L2 smart sample to train model for class *j*.

5. Parallel testing

The recognition process consists of substituting each of test objects $\omega_j, j = 1,..., N_{tst}$ into the decision rule, which results in the determination of the respective class labels estimates $\hat{y}_j, j = 1,..., N_{tst}$. Test of one object is made fast enough. To speed up testing of a set of objects, simple natural data parallelism is used by dividing the entire set into groups and testing each subgroup in parallel.

6. Multiclass Recognition Quality Estimation

To estimate the multiclass recognition quality, such standard quality measure as accuracy is used (Opitz, J., 2024):

$$Accuracy = \frac{correct\ \ classifications}{all\ classifications} \cdot 100\% \ .$$

7. Experiments

7.1 Data description

In experiments of this paper 3 large MNIST data sets and the KDDcup data set are used.

Originally MNIST is the data set of 60000 handwritten digit images for the training and 10000 images for the testing. In these experiments the basic data set was extended by addition synthetic images obtained as pseudo-random deformations and translations of original MNIST images. Synthetic images were generated using infiMNIST program by Leon Bottou that is available at https://leon.bottou.org/projects/infimnist.

Main characteristics of obtained data sets are presented at the Table 1.

Training sets	Objects (original / synthetic)	features
MNIST60k	60 000 (60 000 / 0)	784
MNIST200k	200 000 (60 000 / 140 000)	784
MNIST500k	500 000 (60 000 / 440 000)	784
Test set		
MNIST10k	10 000 (10 000 / 0)	784

Table 1. MNIST data sets characteristics

Feature values of all MNIST data sets have the meaning of image pixel brightness and are in the range of 0…255. Before training and testing all MNIST data are normalized by dividing by 255.

KDDcup data set contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment. It contains 4 898 431 objects for the training and 311 029 objects for test. The original KDDcup data set with data description can be downloaded at http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

Each object of KDDcup data set is initially represented by 38 numerical and 3 categorical features. Each categorical feature was encoded through the One-Hot-Encoding procedure. As the result of encoding 122 numerical features were obtained.

Originally KDDcup data set contains 5 class labels: normal (no attack) and 4 types of attacks: dos, u2r, r2l and probe. But since 3 last of them contain a very small number of samples, in this experiment we combined them into one class, thus obtaining three classes: "no attack", "dos attack" and "other types of attacks".

7.2 Computational System Characteristics

Experiments described in the paper were carried out on a computational system with the following characteristics: Процессор: Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50GHz (22 cores / 44 threads), RAM 196 Gb.

7.3 Related Approaches

This section describes popular approaches and open-access tools for solving the binary SVM classification problem. Each of them is considered below as the basis for construction the multiclass SVM classifier.

7.3.1 SVC and LinearSVC. First of all, we consider implementations that allow to obtain an accurate decision of the SVM problem. In this study we use implementations that are based on the traditional state-of-the-art LibSVM **Ошибка! Источник ссылки не найден.** library and included in python scikit-learn package: sklearn.svm.LinearSVC (optimized for linear problems) and universal sklearn.svm.SVC that is used for construct nonlinear decisions.

7.3.2 RBFSampler + LinearSVC. This is one of popular heuristic approaches aimed to accelerate SVM training in the nonlinear case. It combines generation of nonlinear features whose inner product is approximately equal to the kernel values (Rahimi and Recht, 2008).

We use python scikit-learn implementation here sklearn. kernel_approximation.RBFSampler with consequent well-scalable linear training by sklearn.svm.LinearSVC.

7.4 Experimental Setup

The main goal of this work is to speed up the solution of the multi-class SVM recognition problem, i.e. to obtain a result equal or near in quality to the libsvm result, which is recognized as the standard for the quality of the solution of this problem. Therefore, we compare only SVM-based approaches and do not consider other methods for solving the multi-class recognition problem.

Each of methods for solving binary SVM problem described in Section 7.3.1 and 7.3.2 was coupled with both One-versus-Rest (OvR) and One-versus-One (OvO) strategies (Bishop, C. M., 2006) for extending binary SVM to multiclass one and applied to each of training sets from the Table 1.

In all experiments we set the SVM parameter C=10 and for all nonlinear methods (including RBFSampler) the RBF kernel with $\gamma = 0.01$ was used.

For LinearSVC (as a separate classifier and after RBFSampler transformation) default parameters were set in (except of C=10).

The main parameter of RBFSampler, named *nc* (number of components that corresponds to the dimensionality of the computed feature space) was taken equal to 1000 and 3000.

For sequential version of the Double-Layer Smart Sample SVM (DLSS-SVM) method next parameters were varied: the random sample size (*rs*), the 1-st and 2-nd layer smart sample size (*ss1* and *ss2*, respectively). The early stopping thresholds for the 1-st and 2-nd layers were set in as 100 in all experiments.

The proposed HPDLSS-SVM compared to DLSS-SVM has three additional parameters: P_U , P_L and P_{tst} . The first two of them set the number of processes at the upper and the lower levels of the proposed parallel computing scheme of training, respectively. And last set the number of processes for testing.

7.5 Experimental Results

7.5.1 Comparing to other methods. Tables 2-5 contain training and testing times and accuracy averaged through 3 runs. Standard deviations of accuracy for MNIST60k and MNIST200k does not exceed 0.001, for MNIST500k does not exceed 0.002, and for KDDcup does not exceed 0.005.

Method		Time (s)		Accuracy (%)
		train	test	
SVC OvR		2738	101.90	97.35
SVC OvO		203.50	232.50	96.94
LinearSVC OvR		79.75	0.44	87.44
LinearSVC OvO		21.99	1.28	88.39
RBF Sampler	*nc* = 1000 OvO	84.65	0.9284	92.84
	nc = 1000 *OvR*	78.79	14.67	93.99
	nc = 3000 OvO	249.44	2.79	95.47
	nc = 3000 *OvR*	180.89	49.50	95.90
DLSS-SVM	*rs*=3000 *ss1*=*ss2*=1000	51.20	77.02	96.07
	rs=5000 *ss1*=*ss2*=3000	476.82	146.03	97.26
	rs=6000 *ss1*=1000 *ss2*=10000	874.07	288.25	97.40
HP-DLSS-SVM *rs*=6000 *ss1*=1000 *ss2*=10000 $P_U=10$ $P_L=4$ $P_{tst}=20$		199.19	33.46	97.40

Table 2. Experimental results for MNIST60k

Method		Time (s)		Accuracy (%)
		train	test	
SVC OvR		20688	302.2	98.7
SVC OvO		2686	504.9	98.54
LinearSVC OvR		1098	0.184	88.69
LinearSVC OvO		297.6	0.639	91.44
RBF Sampler	*nc* = 1000 OvO	320.6	1.73	94.87
	nc = 1000 *OvR*	282.7	10.81	95.45
	nc = 3000 OvO	787.28	2.68	96.94
	nc = 3000 *OvR*	688.08	45.87	97.43
DLSS-SVM	*rs*=3000 *ss1*=*ss2*=1000	80.00	87.59	96.73
	rs=5000 *ss1*=*ss2*=3000	304.71	194.08	98.11
	rs=6000 *ss1*=1000 *ss2*=10000	854.75	280.09	98.40
HP-DLSS-SVM *rs*=6000 *ss1*=1000 *ss2*=10000 $P_U=10$ $P_L=4$ $P_{tst}=20$		128.91	60.49	98.40

Table 3. Experimental results for MNIST200k

As we can see from the Tables 2-5, the accurate state-of-the-art LinearSVC method is scalable well enough. Its accuracy is higher for OvO strategy in contrast to OvR one, but absolute values are much less then SVC because binary SVM subproblems are as a rule linear inseparable and nonlinear decision functions are required.

The accurate state-of-the-art SVC method is nonlinear, but despite the use of a number of heuristics to speed up the work with kernels, its scalability is very low and for MNIST500k the training time for OvR strategy as well as for KDDcup for both OvR and OvO strategies exceeds 10 hours. Applying OvO strategy essentially reduces the training time due to decreasing size of binary problems, but the absolute values remain to be very large.

An attempt to replace work with kernels with a specially generated feature space using RBFSampler allows to reduce training time, but leads to loss of information and consequent a noticeable loss of accuracy.

Method		Time (s)		Accuracy (%)
		train	test	
SVC OvR		>36000	-	-
SVC OvO		17133	784.5	98.76
LinearSVC OvR		2457	0.174	88.39
LinearSVC OvO		729.5	0.641	91.60
RBF Sampler	*nc* = 1000 OvO	782.8	1.958	94.64
	nc = 1000 *OvR*	750.41	1.047	94.78
	nc = 3000 OvO	2548.19	2.243	96.93
	nc = 3000 *OvR*	2266.32	3.371	97.16
DLSS-SVM	*rs*=3000 *ss1*=*ss2*=1000	135.05	99.71	97.19
	rs=5000 *ss1*=*ss2*=3000	379.87	184.67	97.81
	rs=6000 *ss1*=2000 *ss2*=10000	861.32	308.2	98.60
HP-DLSS-SVM *rs*=6000 *ss1*=2000 *ss2*=10000 $P_U=10$ $P_L=4$ $P_{tst}=20$		131.20	62.93	98.60

Table 4. Experimental results for MNIST500k

Method		Time (s)		Accuracy (%)
		train	test	
SVC OvR		>36000	-	-
SVC OvO		>36000	-	-
LinearSVC OvR		3335.13	0.308	79.67
LinearSVC OvO		860.58	0.624	78.40
RBF Sampler	*nc* = 1000 OvO	822.26	3.399	86.20
	nc = 1000 *OvR*	447.54	1.686	85.06
	nc = 3000 OvO	Internal error		
	nc = 3000 *OvR*			
DLSS-SVM	*rs*=3000 *ss1*=*ss2*=1000	11.98	14.69	83.00
	rs=5000 *ss1*=*ss2*=3000	475.63	91.68	87.80
	rs=6000 *ss1*=2000 *ss2*=10000	1740.70	192.40	88.00
HP-DLSS-SVM *rs*=6000 *ss1*=2000 *ss2*=10000 $P_U=10$ $P_L=4$ $P_{tst}=20$		353.22	12.34	88.00

Table 5. Experimental results for Kddcup

DLSS-SVM allows achieving the best or near the best accuracy in significantly less time compared to SVC or achieving essentially better quality for the same time compared to other sequential methods.

The proposed HP-DLSS-SVM method for optimal number of processes (that are stated at tables 2-5), uses the computing system possibilities more fully in contrast to other methods. As we can see, it allows to speed-up DLSS-SVM training and testing without losing accuracy. From the other hand it outperforms almost all other results both in time and quality at once. In particular, HPDLSS-SVM for MNIST500k (Table 4) is more than 100 times faster in contrast to the traditional libsvm

library (in this paper the SVC from the python scikit-learn tools) with losing the only 0.14% in the quality.

Since HP-DLSS-SVM following DLSS-SVM implements a nonlinear model, even with parallel realization its testing time is expectedly longer compared to linear models (LinearSVC and RBF Sampler+LinearSVC), but shorter compared to other nonlinear models.

7.5.2 Scalability investigation. This subsection describes experiments whose main goal is to show how change the performance of the proposed HP-DLSS-SVM for different number of processes. All experiments of this section are presented for MNIST500k data set for parameters *rs*=6000, *ss1*=1000, *ss2*=10000.

Table 6 presents HPDLSS-SVM training times in seconds for different number of parallel processes at the upper (P_U) and lower (P_L) levels of the proposed parallel computing scheme. In this table, only those cells are filled that correspond to the total number of processes $P_U \cdot P_L$ that is suitable for the computing system used (Section 7.2).

P_U	P_L							
	1	2	4	6	8	10	15	20
1	906	720	553	488	454	415	400	408
2	470	433	352	283	267	264	278	286
4	358	219	202	198	194	190	-	-
6	253	193	178	164	-	-	-	-
8	201	185	178	-	-	-	-	-
10	183	138	131	-	-	-	-	-

Table 6. HPDLSS-SVM training time (s) for different number of parallel processes at the upper (P_U) and lower (P_L) levels

Table 7 contains speed-up values, computed for the table 6 respectively to the sequential time, obtained for P_U =1 and P_L =1.

P_U	P_L							
	1	2	4	6	8	10	15	20
1	1.00	1.26	1.64	1.86	2.00	2.18	2.27	2.22
2	1.93	2.09	2.57	3.20	3.39	3.43	3.26	3.17
4	2.53	4.14	4.49	4.58	4.67	4.77	-	-
6	3.58	4.69	4.90	5.52	-	-	-	-
8	4.51	4.90	5.09	-	-	-	-	-
10	4.95	6.57	6.92	-	-	-	-	-

Table 7. HPDLSS-SVM speed-up for different number of parallel processes at the upper (P_U) and lower (P_L) levels

Figure 2 shows dependences of acceleration on the total number of processes $P_U \cdot P_L$.

As we can see from tables 6 - 7 and the figure 2, the parallelization at both of levels decreases the training time. Most speed-up is reached for $P_U = 10$, that corresponds to the number of MNIST classes. At that the best speed-up is obtained for parallelization at both of levels which confirms the feasibility of using the proposed two-level parallel computing scheme.
At that, the obtained speed-up is evidently less than the total number of processes. At the upper level this fact is explained by different computational complexity of binary SVM problems and forced wait for the slowest task. At the lower level there is some sequential fragment, which limits the maximal possible speed-up. And moreover, the speed-up gain naturally slows down significantly (or even becomes negative) when the total number of processes ($P_U \cdot P_L$) exceeds the number of computing system cores (24 for the current system), as we can clear see at the figure 2.

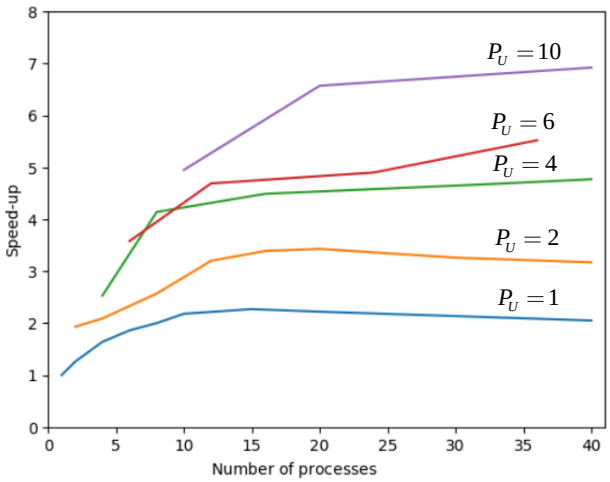

Figure 2. Dependences of speed-up on the total number of processes ($P_U \cdot P_L$).

8. Conclusions

The paper proposes high performance version of the modern sequential method for large-scale multiclass SVM learning named HP-DLSS-SVM. It is based on the two-level hybrid parallel computing scheme and uses the shared memory mechanism for process communication and the memory mapping mechanism to avoid the lack of memory problem. Experiments show that the proposed approach allows to essentially decrease training and testing times and at that to maintain the obtained recognition accuracy close to the best.

Acknowledgements

This research is funded by the Ministry of Science and Higher Education of the Russian Federation within the framework of the state task FEWG-2024-0001.

References

Agarwal, A., Duchi, J.C., 2011. Distributed delayed stochastic optimization. *Advances in Neural Information Processing Systems*, 24, 873-881. doi.org/10.48550/arXiv.1104.5525.

Benfenati, A., Chouzenoux, E., Franchini, G., Latva-Aijo, S., Narnhofer, D., Pesquetm, J.-C., Scott, S.J., Yousefi, M., 2023. Majorization-Minimization for sparse SVMs. doi.org/10.48550/arXiv.2308.16858.

Bishop, C. M., 2006. Pattern Recognition and Machine Learning. Springer.

Byrd, R.H., Hansen, R.H., Nocedal, J., Singer, Y., 2016. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2), 1008-1031. doi.org/10.1137/140954362.

Chauhan, V.K., Sharma, A., Dahiya, K., 2018. Faster learning by reduction of data access time. *Applied Intelligence*, 48(12), 4715-4729. doi.org/10.1007/s10489-018-1235-x.

Chu, C.-T., Kim, S.K., Lin, Y.-A., Yu ,Y., Bradski, G., Ng, A.Y., Olukotun, K., 2006. Map reduce for machine learning on multicore. Advances in neural information processing systems, 19, 281-288.

Dekel, O., Gilad-Bachrach, R., Shamir, O., Xiao, L., 2012. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1), 165-202. doi.org/10.48550/arXiv.1012.1367.

Kurbakov, M. Yu. and Sulimova, V. V., 2024. Fast SVM-based Multiclass Classification in Large Training Sets, *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, X-2/W1-2024, 17–23, https://doi.org/10.5194/isprs-annals-X-2-W1-2024-17-2024.

Kurbakov, M. Yu. and Sulimova, V. V., 2023. Fast SVM-based One-Class Classification in Large Training Sets. *IX International Conference on Information Technology and Nanotechnology (ITNT)*, Samara, Russian Federation, 2023, pp. 1-6, doi: 10.1109/ITNT57377.2023.10139268.

Makarova, A.I., Sulimova V.V., 2019. Fast approximate two-class SVM learning for large training sets. *Procs. of Int. Conference Information Technology and Nanotechnology* (In Russian), 21-24.

Niu, F., Recht, B., Re, C., Wright, S.J., 2011. HOGWILD: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. *Advances in Neural Information Processing Systems*, 21(1). doi.org/10.48550/arXiv.1106.5730.

Opitz, J., 2024. A Closer Look at Classification Evaluation Metrics and a Critical Reflection of Common Evaluation Practice. *Transactions of the Association for Computational Linguistics*. **12**: 820–836. doi:10.1162/tacl_a_00675.

Rahimi, A., Recht, B., 2008. Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning. *Advances in neural information processing systems*, 21, 1313-1320.

Rizzi, A.M., 2016. Support vector regression model for BigData systems. doi.org/10.48550/arXiv.1612.01458.

Sleeman, W.C., Krawczyk, B., 2021. Multi-class imbalanced big data classification on Spark. *Knowledge-Based Systems*, 212. doi.org/10.1016/j.knosys.2020.106598.

Vapnik, V.N., 1995. The nature of statistical learning theory. Springer-Verlag New York, Inc.

Wen, Z., Shi, J., Li, Q., He, B., Chen, J., 2018. ThunderSVM: a fast SVM library on GPUs and CPUs. *The Journal of Machine Learning Research*, 19(21), 1-5.

Zhao, P., Zhang, T., 2014. Accelerating Minibatch Stochastic Gradient Descent using Stratified Sampling. doi.org/10.48550/arXiv.1405.3080.

You, Y., Demmel, J., Czechowski, K., Song, L., Vuduc, R., 2015. CA-SVM: Communication-Avoiding Parallel Support Vector Machines on Distributed Systems. *2015 IEEE International Parallel and Distributed Processing Symposium*, 847-859. doi.org/10.1109/IPDPS.2015.117.