ISSN 1060-992X, Optical Memory and Neural Networks, 2016, Vol. 25, No. 4, pp. 203–218. © Allerton Press, Inc., 2016.

Fast Online Algorithm for Nonlinear Support Vector Machines and Other Alike Models

Vojislav Kecman

Computer Science Department, Virginia Commonwealth University, Richmond VA, USA

e-mail: vkecman@vcu.edu

Received July 4, 2016

Abstract—Paper presents a unique novel online learning algorithm for eight popular nonlinear (i.e., kernel), classifiers based on a classic stochastic gradient descent in primal domain. In particular, the online learning algorithm is derived for following classifiers: L1 and L2 support vector machines with both a quadratic regularizer $\mathbf{w}^t\mathbf{w}$ and the l_1 regularizer $|\mathbf{w}|_1$; regularized huberized hinge loss; regularized kernel logistic regression; regularized exponential loss with l_1 regularizer $|\mathbf{w}|_1$ and Least squares support vector machines. The online learning algorithm is aimed primarily for designing classifiers for large datasets. The novel learning model is accurate, fast and extremely simple (i.e., comprised of few coding lines only). Comparisons of performances of the proposed algorithm with the state of the art support vector machine algorithm on few real datasets are shown.

Keywords: online learning, OLLA, stochastic gradient descent, loss function, regularizer, nonlinear classifiers, kernel SVM, ultra large datasets

DOI: 10.3103/S1060992X16040123

INTRODUCTION

In the last few years the sizes of datasets used and analyzed in all areas of human activity have significantly grown and are still steadily growing. The increase in datasets size will continue at even greater rate in years to come. Hence, the data scientists are forced to develop novel learning algorithms that scale well with the number of training samples. That's what we are trying to achieve here. We are concerned with developing a unique fast learning algorithm for nonlinear classification (pattern recognition) problems which can handle huge number of high dimensional data. It is supposed to be implemented on a single processor meaning, we are not going into its parallelization issues and properties here. Sure, many parts of the algorithm presented here can be parallelized in future increasing the efficiency of the models presented in this manner. The approach taken and used for devising the online learning algorithm (OLLA) for nonlinear (kernelized) classifiers can be readily applied for regression problems too.

Support vector machines (SVMs) are known for creating sparse, generally nonlinear (i.e., kernel), classifiers and due to this characteristics they are one of the most suitable classifiers for handling large datasets while still keeping the accuracy at high level at the same time [1–5]. The paper is focused on devising and developing an OLLA tailored not only for SVMs but for a series of other popular classifiers which use different risk functions. We'll show that, in addition to the SVMs, the other models which are usually considered as the dense ones (e.g., the regularized kernel logistic regression (RKLR) model), can also be sparse if trained with OLLA proposed here.

We'll introduce many possible variants of models and obviously, due to the space limit, we can't show experimental results for all of them. Neither we can show all the comparisons of their performances. The experimental runs with the models are ongoing and very extensive. They will require a lot of computing time in order to show and point to their basic advantages and deficiencies. However, we will point to some basic properties of online (OL) learning approach. We are also sure that the presented approach will shed a new light on both online algorithms and properties of various nonlinear classifiers.

SETTING THE PROBLEM

SVMs are popular for their accuracy and sparseness both of which lead to a fast training even when dealing with large datasets. The learning algorithm of the classic, the so-called L1 SVM (which uses the sum of ξ in a risk function, where ξ measures a distance of data point to its margin) and L2 SVM (which uses sum of ξ^2), is based on solving a quadratic programming problem in a *dual domain* subject to the set

of block constraints and a single equality constraint (for L1 SVM). $\xi_i = 1 - y_i o_i$, where y_i is the known label for the data sample $\mathbf{x}_i$ and o_i is the model's output for the very same $\mathbf{x}_i$. These were the leading approaches for solving classification and regression tasks by SVMs. Recently, several authors have proposed the use of a standard stochastic gradient descent (SGD) approach for SVMs while solving machine learning problems [5–9]. The mentioned approaches are extended variants of a classic kernel perceptron algorithm [10]. This paper is all three a unification, a simplification and an expansion of otherwise close, approaches presented in [5–9] and [14, 15]. We present the single unique algorithm for several different, well known, classifiers and not only for SVMs. In addition, the presented OLLA is valid for a variety of classifiers disregarding whether they use an regularizer or not. (See in [11] the justification for not using the regularization term $0.5\|\mathbf{o}\|_2^2$ and $0.5\|\mathbf{w}\|_2^2$ in the risk functions (1) and (2) shown below).

Let's now assign the task: consider the problem of binary classification (i.e., dichotomization) given by the finite data set *D:* $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, . . ., $(\mathbf{x}_l, y_l)$, $\mathbf{x} \in \Re^n$, $y \in \{+1, -1\}$, where all l data points pairs $(\mathbf{x}_i, y_i)$ are assumed to be generated independently from the same distribution. We want to find a decision function (model) which output o represents a mapping function o: $\Re^n \to \Re$ that (approximately) minimizes the risk (composed as a sum of a loss term and an regularizer) given in (1) and (2). Note that $o(\mathbf{x}, \mathbf{w})$ is parameterized by the so-called weights denoted usually by $\mathbf{w}$. $o(\mathbf{x}, \mathbf{w})$ is called decision function in classification. In regression $o(\mathbf{x}, \mathbf{w})$ is called approximation, or regression, function. The risk is given as,

$$\underbrace{\inf_{o\in H} R_{rl}}_{} = \underbrace{C\sum_{i=1}^{l} L\left(y_i, o(\mathbf{x}_i)\right)}_{\text{LOSS}} + \underbrace{\frac{1}{2}\|\mathbf{o}\|_2^2}_{\text{REGULARIZER}}, \tag{1}$$

where H is a Reproducing Kernel Hilbert Space (RKHS), L denotes *loss* functions, $\|\,.\,\|_2$ represents the L_2 norm *regularizer* and the index *rl* stands for ***r**egularized **l**oss*. Note that this problem posing differs from a standard risk function used in a geometric approach of a classic SVMs' learning setting given in (2)

$$\underbrace{\min_{(\mathbf{w},b)\in H_o x\Re} R_{rl}}_{} = \underbrace{C\sum_{i=1}^{l} L\left(y_i, o_{(\mathbf{w},b)}(\mathbf{x}_i)\right)}_{\text{LOSS}} + \underbrace{\frac{1}{2}\|\mathbf{w}\|_2^2}_{\text{REGULARIZER}}, \tag{2}$$

where H_o is a general Hilbert space and a function $o(\mathbf{w}, b)$ is defined in terms of an affine hyperplane specified by $(\mathbf{w}, b)$ in this space. It is important to state that both approaches are equivalent for a fixed value of the bias term b [12]. This is not the case if one uses bias b. Then, the decision functions o obtained by optimizing (1) and (2) are different. Note, however, that if a positive definite kernel is used there is no need for a bias term b but b can, nevertheless, be used. This is why all the models presented here can be implemented with a bias b or, without it. If b is not a part of the model one should omit it in (11) and one takes $\beta = 0$ in the OLLA presented in the Lists 1 and 2. (Finally, let's just make a comment that in references coming from different disciplines the loss can be found under the names cost, merit, norm, fitness, objective, etc, ... , function).

Hence, in the rest of paper, we'll use (2) as the risk function to optimize. We'll devise OLLA for eight kernelized classifiers differing by loss functions and regularizers used. In both ML field and statistics community, the models originating from a minimization of different risks have different names as shown next. In particular, we'll derive a *unique* online algorithm for the following nonlinear classifiers:

L1 SVM with Regularizer $\mathbf{w}^t\mathbf{w}$

$$R_{rl} = C\sum_{i=1}^{l} \max\left(0, 1 - y_i o_i\right) + \frac{1}{2}\|\mathbf{w}\|_2^2 \tag{3}$$

L2 SVM with Regularizer $\mathbf{w}^t\mathbf{w}$

$$R_{rl} = C\sum_{i=1}^{l} \max\left(0, \frac{1}{2}\operatorname{sgn}\left(1 - y_i o_i\right)\left(1 - y_i o_i\right)^2\right) + \frac{1}{2}\|\mathbf{w}\|_2^2 \tag{4}$$

L1 SVM with Regularizer $|\mathbf{w}|_1$

$$R_{rl} = C\sum_{i=1}^{l} \max\left(0, 1 - y_i o_i\right) + |\mathbf{w}|_1 \tag{5}$$

L2 SVM with Regularizer $|\mathbf{w}|_1$

$$R_{rl} = C\sum_{i=1}^{l} \max\left(0, \frac{1}{2}\operatorname{sgn}(1 - y_i o_i)(1 - y_i o_i)^2\right) + |\mathbf{w}|_1 \tag{6}$$

Huberized hinge loss with Regularizer $\mathbf{w}^t\mathbf{w}$

$$R_{rl} = C\sum_{i=1}^{l} L_{HHL} + \frac{1}{2}\|\mathbf{w}\|_2^2, \tag{7a}$$

where L_{HHL} is defined as

$$L_{HHL} = \begin{cases} 0 & y_i o_i > 1 \\ \dfrac{(1 - y_i o_i)^2}{2\delta} & 1 \geq y_i o_i \geq 1 - \delta \\ 1 - y_i o_i - \dfrac{\delta}{2} & y_i o_i < 1 - \delta \end{cases} \tag{7b}$$

Kernel logistic regression with Regularizer $\mathbf{w}^t\mathbf{w}$

$$R_{rl} = C\sum_{i=1}^{l} \ln\left(1 + e^{-y_i o_i}\right) + \frac{1}{2}\|\mathbf{w}\|_2^2 \tag{8}$$

Exponential loss with Regularizer $|\mathbf{w}|_1$ (used in a boosting)

$$R_{rl} = C\sum_{i=1}^{l} e^{-y_i o_i} + |\mathbf{w}|_1 \tag{9}$$

LS SVM, Quadratic Loss Function with Regularizer $\mathbf{w}^t\mathbf{w}$

$$R_{rl} = C\sum_{i=1}^{l} \frac{1}{2}(1 - y_i o_i)^2 + \frac{1}{2}\|\mathbf{w}\|_2^2, \tag{10}$$

where C stands for the regularization penalty parameter. (It may be useful to comment that the risk $0.5\Sigma(y_i - o_i)^2 + \lambda|\mathbf{w}|_1$ is dubbed LASSO in statistics and a basis pursuit in signal processing). Related to the sum-of-error-squares, note that in the classification case when $y = \pm 1$, the loss in (10), namely $(1 - y_i o_i)^2$, equals $(y_i - o_i)^2$. Also, for a numerical convenience, we have used 0.5 in quadratic terms which doesn't change optimization result. Next, we have adopted the use of a penalty i.e., tradeoff parameter C as in a classic SVM problem posing and not λ penalty which typically multiplies the regularization term. The relation between the two is obvious. However, note that versions with C and λ show different behavior in OL learning. This phenomenon requires some investigation in future.

The proper value of C is not known in advance, and it should be obtained by cross-validation.

For a general nonlinear classifier (when the data points $\mathbf{x}_i$, $i = 1, \ldots, l$ are mapped from an original $\Re^n$ space to data points $\boldsymbol{\varphi}(\mathbf{x}_i)$ in a high (including, possibly, infinitely) dimensional feature space Φ), the model's output o for a given input vector $\mathbf{x}$ is defined as

$$o(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b, \tag{11}$$

where α_i are the coefficients (weights) of the expansion in a feature space, $k(\mathbf{x}, \mathbf{x}_i) = \boldsymbol{\varphi}(\mathbf{x})^T\boldsymbol{\varphi}(\mathbf{x}_i)$ is the value of i-th kernel for given vectors $\mathbf{x}$ and $\mathbf{x}_i$ while b stands for the bias term. As for the terminology used here let's mention that the classifiers are called *dense* if all the absolute values of the weights $\text{abs}(\alpha_i) > 0$. The classifiers are dubbed *sparse* if one, more or many $\text{abs}(\alpha_i) = 0$. The level of sparseness varies but the sparser the model, the faster the training, testing and applications, as well as the smaller a classifier's memory space required. SVMs are particularly well known for often producing very sparse models. Before presenting the OLLA and for better understanding the very nature of the algorithms below, we show the loss functions for all eight models presented earlier in Fig. 1 below. Note, that all the losses shown are *convex* functions. Note also that the losses for L1 and L2 SVM as well as the Huberized hinge loss are non-differen-

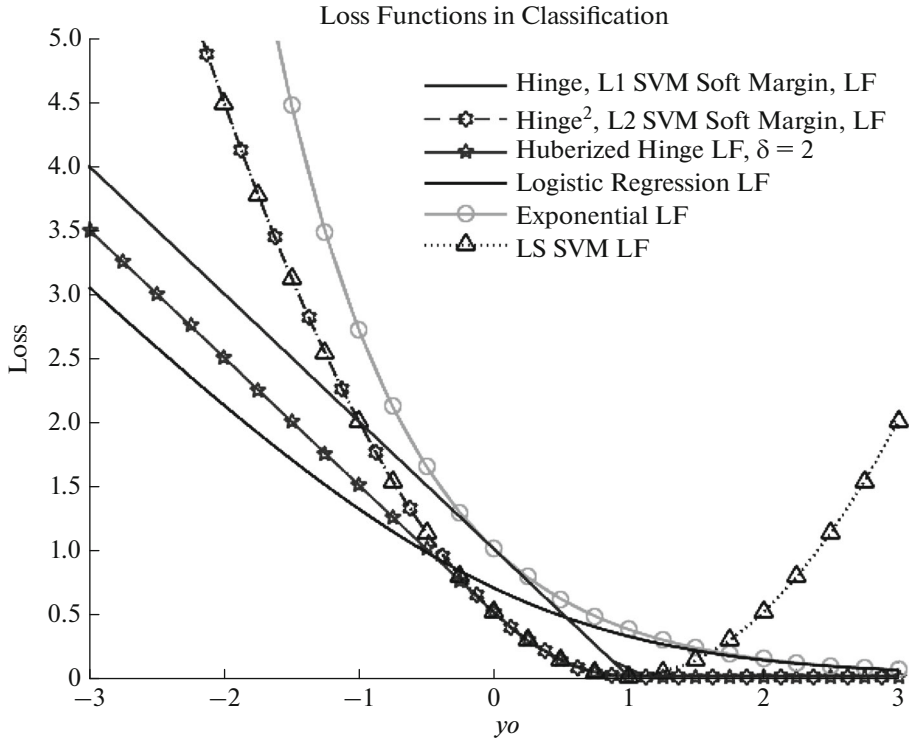

Fig. 1. Loss functions used for the models (3)–(10).

tiable functions but they do have subgradients and all of them have a knick-point at $yo = 1$. The gradient of all these losses after the knick-point equals zero, and this will lead to sparse models. Remind that y is a known label of data $\mathbf{x}$ which can take only +1 or −1 value, while o is the output of the model.

Hence, whenever the value $yo \geq 1$ the loss will equal zero and for $yo < 1$ the loss increases linearly (for L1 SVM) and quadratically (for L2 SVM). Note also that whenever $y_i o_i < 0$, the data $\mathbf{x}_i$ is misclassified. Huberized hinge loss function has three sections. Its loss equals zero for $yo > 1$, it is quadratic when $1 - \delta < yo \leq 1$ and it increases linearly for $yo < 1 - \delta$. Notice that for $\delta = 0$ Huberized loss actually equals the L1 SVM hinge loss.

Such a character of loss functions makes clear why their models are sparse. In the Lists below, the tiny loop *if* $yo > 1$, $\Lambda = 0$, *end* is the very origin of sparseness. However, looking at the differentiable logistic regression loss function (for which the tiny *if loop* doesn't apply) doesn't quite reveal why OLLA presented below produces sparse models for it too. Explanation will be given soon and now it's time to present the unique and single algorithm for different classifiers given by risks (3)–(10). The first version of a unique OLLA for all the models is given in the List 1 below.

List 1: Online Learning Algorithm for Minimizing Risks R_{rl} (3)–(10)

Define the number of epochs N, l *is number of data*
Shuffle the dataset & Initialize $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{o} = \mathbf{0}$, $b = 0$, $i = 0$
for $t = 1,\ldots, Nl$
$\eta = \sqrt{2/t}$, *or* $\eta = 1/t$, *or…something else*
$i = i + 1$. *Take* $y_i o_i$ & *calculate* Λ *and* P
if $y_i o_i > 1$, $\Lambda = 0$, **end** ← Only for nondifferentiable risks (3)–(7)
$\mathbf{o} = \mathbf{o} + (\Lambda - \mathrm{P})\mathbf{k}(:, \mathbf{x}_i) + \Lambda\beta$
$\alpha_i = \alpha_i + \Lambda - \mathrm{P}$
$b = b + \Lambda\beta$
if $i = l$, $i = 0$, **end**
end,

where the update parameters Λ (lambda) and P (the old Greek capital *Rho*) denote scalar values which are used in updating the output vector **o**, the weight vector in a kernel feature space **α** and the bias term *b* (if present). Mnemonics for the notation used is simple—Λ originates from the gradients of **L**oss while P (*Rho*) comes from the ones of the **R**egularizers. Due to the limited space assigned, we can't show all the details of algorithm's derivation for all eight models. Instead, we only point to some interesting and important characteristics of the OLLA.

As already mentioned, index *rl* denotes the risk function for regularized losses. Here, the OLLA introduced above will also cover classifiers *without* the regularization term. In such cases the risk will equal only the loss term and, consequently, P = 0 in the algorithm.

Based on the loss function characteristics, there are two groups of classifiers. The first five models (3)–(7) are defined by the fact that their losses *L* are not differentiable over the whole domain of *yo* values but, they do have defined subgradients with respect to model parameters **w** and *b*. Models given by risks (8)–(10) have differentiable losses over the whole *yo* domain. However, the OLLA in Lists 1 and 2 (which follows below) is same for both groups of models.

Note that OLLA has two different update sections for the risks (3)–(6). The first one is the calculation of Λ and P and the second one is redefining Λ for $y_i o_i > 1$ when their gradients and, consequently, Λ-s equal zero. Model with an regularized huberized hinge loss (RHHL) has three different sections hinted in (7b) and, depending upon the value of $y_i o_i$, the update coefficients Λ and P for RHHL classifier should be calculated as given in (7_{upd}) below.

The common knowledge in ML community is that the first group of non-differentiable risks creates sparse models, while the second group doesn't. Here, we'll show that the regularized kernel logistic regression model, generally considered to be dense [13], can within the OLLA environment conditions, create sparse models. This is an interesting and powerful result of the paper.

However, and again, the OLLA given in Lists 1 and 2 is valid for all the models. This means that it is also valid for the last three classifiers with risks *R* given by (8)–(10). As already mentioned, the loop *if* $y_i o_i > 1$ doesn't apply for them in the Lists.

There are plenty of varieties of possible classifiers one can design based on some preliminary assumptions one makes. First, we must decide whether to use the model *with* or *without* the bias term *b*. As already mentioned, if a positive definite kernel is used there is no need for bias term *b* but a bias *b* can, nevertheless, be used. Experiments show that, generally, the use of a bias term *b* makes the final model a little bit sparser. If the model had a bias term *b*, β = 1. Otherwise β = 0.

Next, one has to decide about the regularization. In other words, one can choose to minimize a loss function *with* an regularizer (either $\mathbf{w}'\mathbf{w}$ or $|\mathbf{w}|_1$ or, some other regularizer) or one can design the classifier *without* any regularizer. Arguments about not using regularizer term can be found in [11], [14], and [15]. The basic rationalization for not using the regularizer is based on the claim that early stopping may have similar effects on final results as the regularizer implementation. The model designer should decide for each particular classification problem whether the use of an regularizer is needed or not. We generally suggest to use it and let the experiments determine whether an regularizer is needed or not. In the cases *without* regularizer P = 0.

What are then the differences between that many dissimilar classifiers presented by their risk functions (3)–(10)? Obviously, all the varieties originate from the way how one calculates update parameters Λ and P as shown next:

L1 SVM with Regularizer $\mathbf{w}'\mathbf{w}$

$$\Lambda = \eta C y_i, \quad \mathrm{P} = \eta \alpha_i \qquad (3_{\text{upd}})$$

L2 SVM with Regularizer $\mathbf{w}'\mathbf{w}$

$$\Lambda = \eta C y_i (1 - y_i o_i), \quad \mathrm{P} = \eta \alpha_i \qquad (4_{\text{upd}})$$

L1 SVM with Regularizer $|\mathbf{w}|_1$

$$\Lambda = \eta C y_i, \quad \mathrm{P} = \eta \operatorname{sgn}(\alpha_i) \qquad (5_{\text{upd}})$$

L2 SVM with Regularizer $|\mathbf{w}|_1$

$$\Lambda = \eta C y_i (1 - y_i o_i), \quad \mathrm{P} = \eta \operatorname{sgn}(\alpha_i) \qquad (6_{\text{upd}})$$

Huberized hinge loss with Regularizer $\mathbf{w}^t\mathbf{w}$

$$\begin{aligned} y_i o_i \le 1-\delta \quad & \Lambda = \eta C y_i \\ 1-\delta < y_i o_i \le 1 \quad & \Lambda = \frac{\eta C y_i (1-y_i o_i)}{\delta}, \quad P = \eta \alpha_i \end{aligned} \tag{7_{upd}}$$

Kernel logistic regression with Regularizer $\mathbf{w}^t\mathbf{w}$

$$\Lambda = \frac{\eta C y_i}{\left(1+e^{y_i o_i}\right)}, \quad P = \eta \alpha_i \tag{8_{upd}}$$

Exponential loss with Regularizer $|\mathbf{w}|_1$ (used in a boosting)

$$\Lambda = \frac{\eta C y_i}{e^{y_i o_i}}, \quad P = \eta \operatorname{sgn}(\alpha_i) \tag{9_{upd}}$$

LS SVM, Quadratic Loss Function with Regularizer $\mathbf{w}^t\mathbf{w}$

$$\Lambda = \eta C y_i (1-y_i o_i), \quad P = \eta \alpha_i. \tag{10_{upd}}$$

One can readily see that each model has different update coefficients except the models for L2 SVM and LS SVM. Their Λ and P are same hinting that the final models may be same too. However, while the updates are same the algorithms, and consequently the final models, are not! What is the difference? The LS SVM's loss function is both continuous and differentiable and, the tiny loop $y_i o_i > 1$ in the pseudocode given in List 1 doesn't apply for it. At the same time, for L2 SVM one has to apply this loop. The consequences are straight: L2 SVM is a sparse classifier and LS SVM is a dense one! In other words, many α_i in L2 SVM model will be equal to zero, while all LS SVM's α_i will always be nonzero values. The denser the model, the longer training time will be. Because LS SVM is always producing dense models, it is not feasible for really big datasets.

Next, the number of models explodes if one plays with the choice of regularizer and whether to use bias b or not. Just replacing the regularizers used and shown in (3)–(7), by swapping $\mathbf{w}^t\mathbf{w}$ by $|\mathbf{w}|_1$, almost doubles the number of models one has to experiment with. Then, using or not using the bias term b doubles the number of models again. In practice, working with that many models will require a lot of time. For smaller datasets, however, experimenting with several models might be very helpful in finding out which model may be the best for a given dataset.

One of the claims made here, namely, that the unique and single learning algorithm presented in Lists 1 and 2 is very suitable for the large datasets, needs both some clarification and an explanation why this may be the case. The comments which follow are valid for all the models which can produce sparse solutions.

Out of eight models presented only the last two create dense models, meaning all their training data points are "support vectors". In other words, all α_i are nonzero i.e., $\alpha_i \neq 0$ for $i = 1, \ldots, l$. While there is no surprise why the first five models create sparse solutions (their losses are the non-differentiable "hinge" type functions with $L = 0$ for $y_i o_i > 1$), there is not quite a straightforward insight why this is the case with a continuous and differentiable logistic regression loss. However, one can readily see from (8_{upd}) that the update parameter Λ takes close to zero values for bigger values of $y_i o_i$. In other words, the nature of the logistic regression loss update parameter Λ is similar to the update parameters Λ for all the hinge type loss functions. This is a very good news and it is contrary to all the common knowledge and general understanding (as well as to a specific pointing at in [13]) about the kernelized logistic regression classifiers where one of the basic claims is that they always create dense models.

However, aside from, and in addition to, the loss functions characters, the OLLA given in List 1 is suitable for large datasets for the other two reasons. The first one is that the much needed value in each iteration step $y_i o_i$ is obtained after extracting the o_i value from the model output vector $\mathbf{o}$ which is in each iteration calculated as the product of a scalar $(\Lambda - P)$ with a kernel vector $\mathbf{k}(:, \mathbf{x}_i)$ instead as a dot product $o_i = \mathbf{k}(\mathbf{x}_i, :)\boldsymbol{\alpha} + b$. Note that calculation of $\mathbf{k}(:, \mathbf{x}_i)$ is the most time consuming computation in each step but, luckily, due to the sparseness character of all the models (except for the last two classifiers), this calculation is taking place for a fraction of samples (future "support vectors") only.

In particular, consider the calculations in the *first* epoch when we start from $\boldsymbol{\alpha} = \mathbf{0}$. Then, if one updates cyclically, for all the models and all data points the update coefficient $P = 0$ in each iteration step. With this, we have to do expensive calculations of $\mathbf{k}(:, \mathbf{x}_i)$ only for "support vectors" data, meaning only when

$y_i o_i < 1$, for all the hinge type of loss functions. That will also be the case for a logistic regression loss whenever $\Lambda \approx 0$ which will happen for almost "same" data points as for the hinge type losses. Once the basic set of "support vectors" has been found and cashed in the first epoch, we don't have to repeat their calculations in the next epochs. The kernel vectors $\mathbf{k}(:, \mathbf{x}_i)$ for possibly new violators will only be computed and cashed in all future iterations. All these features of OLLA speed up the learning in the case of large datasets significantly.

Finally, worth of mentioning is that the second source of algorithm's speed arises from the fact that the update scalars Λ for all the classifiers, except for L1 SVM model, are expressed in terms of already known $y_i o_i$ values. (L1 SVM classifier's Λ doesn't depend upon the $y_i o_i$ value).

However, we can redesign the simple OLLA as presented in the List 1 and change the way how one calculates the output o_i of the model. The reformulated algorithm given in List 2 calculates only the output value o_i by using the scalar product of kernel values and weight vector $\boldsymbol{\alpha}$ for support vectors only $\mathbf{k}(\mathbf{x}_i, \mathbf{x}_\mathbf{I})\boldsymbol{\alpha}_\mathbf{I}$, where $\mathbf{I}$ contains indices of support vectors. Hence, for sparse models OLLA in List 2 will be much faster and it should be used for really large datasets. The sparser the model, the bigger the speed-up of the algorithm.

List 2: Online Learning Algorithm for Minimizing Risks R_{rl} (3)–(10) More Suitable for Ultra Large Datasets

Define the number of epochs N, l *is number of data*
Shuffle the dataset
Initialize $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{o} = \mathbf{0}$, $b = 0$, $i = 0$, $\mathbf{I} = [\,]$, $\tau = 1\mathrm{e}-5$
for $t = 1, \ldots, Nl$
  $\eta = \sqrt{2/t}$, *or* $\eta = 1/t$, *or…something else*
  $i = i + 1$.
  $o_i = \mathbf{k}(\mathbf{x}_i, \mathbf{x}_\mathbf{I})\boldsymbol{\alpha}_\mathbf{I} + b \leftarrow$ Only SVecs used here. Time saving.
  Calculate Λ *and* P
  if $y_i o_i > 1$, $\Lambda = 0$, **end** $\leftarrow$ Only for nondifferentiable risks (3)–(7)
  $\alpha_i = \alpha_i + \Lambda - \mathrm{P}$
  $b = b + \Lambda\beta$
  if $|\alpha_i| > \tau$, **if** $i \notin \mathbf{I}$, $\mathbf{I} = \begin{bmatrix} \mathbf{I} \\ i \end{bmatrix}$, **end**, **end**
  if $i = l$, $i = 0$, **end**
end

where the tolerance parameter τ defines the smallest value of parameters α_i and in this way determines which data are support vectors. Default value for τ is 1e-5.

RELATIONS WITH OTHER KNOWN MODELS AND LEARNING ALGORITHMS

Stochastic gradient descent was used and known in many disciplines for ages. For example, the most popular learning algorithm in neural networks (NNs) known as the Error Back Propagation (EBP) is an SGD algorithm. EBP has been reinvented many times but it has been being used massively from 1980-ties and it's still very popular learning algorithm. The newest machine learning and data mining tool dubbed "*deep learning*" still uses the EBP. Hence, there is a natural question—what is then, if anything, new, or different, or exclusive in using the SGD for the eight models presented here? Well, a lot and, on the different levels of both understandings and meanings of the iterative learning algorithm and of all the risk functions involved.

First, the risk of a classic NN was usually *loss* only which, typically, was sum-of-errors squares i.e., $R = 0.5\Sigma(y_i - o_i)^2$. The $\Sigma|\, y_i - o_i\,|$ was rarely used as a loss. Sure, the designers of NNs had realized very early about possible advantages of, and needs for, performing the regularization. Hence, they were also regularizing. This action was dubbed *ridge regression* i.e., *weight decay*. (However, they, similar to the statistics community, were not quite aware about the geometric meaning of using the regularizer $\mathbf{w}^t\mathbf{w}$ i.e., $\|\mathbf{w}\|^2$, despite the fact that this was introduced by Vapnik and Chervonenkis in the late 1960s and early 1970s).

In addition, classic NNs' learning algorithms optimize both the hidden layer weights (which define the positions of basis functions and their shapes) and the output layer ones. This then creates non-convex risk function and one can end up with suboptimal model at one of many possible local minima.

Finally, the EBP didn't resolve the issue of the number of neurons in a hidden layer. The question about how many neurons to use in hidden layer was typically answered by repeating the experiments with various numbers of neurons.

Here, in OLLA, the risk is composed of both basically *novel* losses and, essentially, old regularizers being either $\mathbf{w}^t\mathbf{w}$ or $|\mathbf{w}|$. However, after an introduction of an SVM theory and corresponding learning algorithm, the regularizer $\mathbf{w}^t\mathbf{w}$ was given a new, geometric, meaning of a margin size between the two classes. Modeling with a clear understanding of a regularizer's novel meaning definitely enriches the model designer with both stronger insights and better comprehension of the results.

Next, only the output layer weights are optimized here (while the hidden layer ones are kept predefined i.e., fixed) which makes the risk a convex function.

Furthermore, and this is a very important aspect of the novel SGD application, the *kernels* are applied here.

The use of kernels brought a lot of novel appreciations into the whole ML field. In addition to that, on a model design level, the use of kernels resolved the issues of a placement (i.e., positioning) of hidden layer basis functions as well as of the cardinality of the hidden layer. In other words, the important and basic design issues in NNs, where to place the neurons and how many neurons to use in the hidden layer is resolved by the use of kernels and the learning algorithm here. All the training data points $\mathbf{x}_i$ are offered as the basis functions initially. Which ones of them will be used for building the model output will be known at the end of learning/training. Whenever the final model is sparse (and, it happens very often that the model uses only 2, 5, 10, 25 or 50% of data), the number of neurons is (significantly) smaller than the cardinality of the training dataset. Such a sparseness of the model is defined by several factors but, primarily, by the nature of the data. The smaller the classes' overlapping, the sparser the model will be. The other design hyperparameters such as the shape of kernels (i.e., the width σ of Gaussians or the order of polynomial kernel) and the choice of the penalty parameter C also influence the sparseness level of a model obtained a lot.

While the placement (i.e., positions) of hidden layer neurons in all the models presented here is defined by the training data, the shape of their basis functions is not and it must be found out by the (usually lengthy) crossvalidation.

DATA AND SOME EXPERIMENTAL RESULTS

Note that there is the huge number of possible nonlinear classifiers one can design with OLLA shown. In fact, depending whether one uses bias term b and whether one runs models *with* or *without* regularization one can design 30 different classifiers. Next, one can experiment by updating weights α_i in a cyclic fashion (as presented in Lists 1 and 2) or based on the maximal violator sample (may be recommended). This then doubles again the variety of classifiers design. In conclusion, the experimental variety is huge indeed and comparing that many models is far beyond one paper. A lot of experimental runs must be performed before making some definite statements about the 8 introduced classifiers' performances. However, there are consoling words to such a quest—similar to the real world there is no perpetuum mobile in machine learning either. In other words, don't even try to find the model which will be best for all possible classification problems and tasks. This is why, in our experiments, we focus only on a few important characteristics of the OLLA given in the List 1.

First, note that the *for loop* iterates maximally up to the, by user defined, number of iterations. If we choose two epochs, number of iterations will be $2l$. Obviously, the bigger the cardinality of datasets the less epochs may be needed. Our first experiments show that satisfying results can be achieved by much less than l iterations for bigger datasets. Sure, with so many different datasets in terms of samples numbers, dimensionality of the features and, most importantly, in terms of the classes' overlapping one can hardly make some general statements about how many iterations are needed. Next, as for all the other stochastic gradient based algorithms, the selection of a learning rate η is very important too. We have suggested 2 possibilities for η in List 1, but the choice of an η change strategy is an open research issue. Also, a proper stopping criterion for the OL learning model shown would be welcomed.

In addition to these classic issues of SGD procedures, all the other issues related to the kernelized classifiers still remain—what kernel to use, what should be the value of penalty parameter C, what is the value

Comparisons of performances of SMO and OL L1 SVM Algorithm

Dataset,	**#of data**/dim	**Error Rate SMO, %**	**Error Rate OL L1SVM, %**	**OL L1SVM is that time faster**
Cancer,	**198**/32	**24.00**	25.62	4.1
Sonar,	**208**/60	34.70	**24.35**	3.68
Vote,	**232**/16	22.68	**8.09**	1.89
Eukaryotic 4 *vs. rest*	**2427**/20	36.05	**14.55**	5.67
Prototask,	**8192**/21	33.57	**11.28**	5.61
Reuters	**11069**/8315	5.65	**4.48**	4.10
Chess board,	**8000**/2	13.62	**4.27**	7.65
Two normally distributed classes,	**15000**/2	28.25	**7.82**	41.50

of the shape i.e., width parameter of Gaussian kernel σ, what is the best order of the polynomial kernel, and so on. The second set of questions is usually resolved with the crossvalidation.

Finally, our first experiences with OLLA suggest that one may expect serious convergence issues for exponential loss with regularizer $|\mathbf{w}|_1$, because for big negative values of $y_i o_i$ the SGD updates led to a divergence of the algorithm. It's similar for L2 SVMs. Some constraints are needed for big negative values of $y_i o_i$. However, constraining may lead to the non-convex risk function. These characteristics and the feasibility of the OLLA for the last two models will be investigated on real datasets in future.

Putting aside discussions about all these issues, we can state that the first experimental results (done by us and others as well) point to very fine properties of the OLLA. Below, we'll first show only few results on the real small to medium datasets.

The comparisons in Table 1 are between the most popular SVM learning algorithm for SVMs the Sequential Minimal Optimization (SMO) [16] implemented within, also the most popular SVM software LIBSVM [17] and an SGD based OLLA for L1 SVM with the bias term *b* but *without* the regularization term. Results shown are obtained by using 5-fold crossvalidation. Table shows many merits of the OL approach and notably how its speed advantage increases with an increase in samples number.

It is very interesting that the SGD algorithm has beaten SMO in terms of both accuracy and speed.

Next, let's show another intriguing characteristics of the OLLA which is to be able to produce the sparse regularized kernel logistic regression classifiers.

The loss functions of an SVM and RKLR are shown in Fig. 2 again, but with an expanded negative part of *yo* values just to point to their two important characteristics. First, for smaller *yo* values the losses are parallel, meaning they have the same increase rates. Second, RKLR is a differentiable function while the SVM's hinge loss is not. And yet, OLLA will often create "similarly" sparse models for SVM and RKLR.

As commented previously the basic difference between two models is that the SVM's loss function (a.k.a., hinge loss or soft margin loss) is non-differentiable with a well defined knick-point at $yo = 1$, while the logistic regression loss is a continuously decreasing function over the whole domain of *yo* values.

Interestingly enough, despite such a character of the RKLR loss, the OLLA is able to create sparse models for RKLR model as well.

This part of the paper is a kind of response to, and discussion about, the common knowledge and understanding concerning the Kernel Logistic Regression and Import Vector Machine as introduced in [13]. The kernel logistic regression model with regularizer $\mathbf{w}^t\mathbf{w}$ is introduced earlier and repeated here for readers convenience.

$$R_{rl} = C\sum_{i=1}^{l} \ln\left(1 + e^{-y_i o_i}\right) + \frac{1}{2}\|\mathbf{w}\|_2^2. \tag{8}$$

The update coefficients for the classifier using R_{rl} above are given as $\Lambda = \eta C y_i \Big/ \left(1 + e^{y_i o_i}\right)$ and $\mathrm{P} = \eta\alpha_i$.

Let's now remind about the basic claims, or 'common knowledge', about RKLR as presented in [13]. They are known to be as follows:

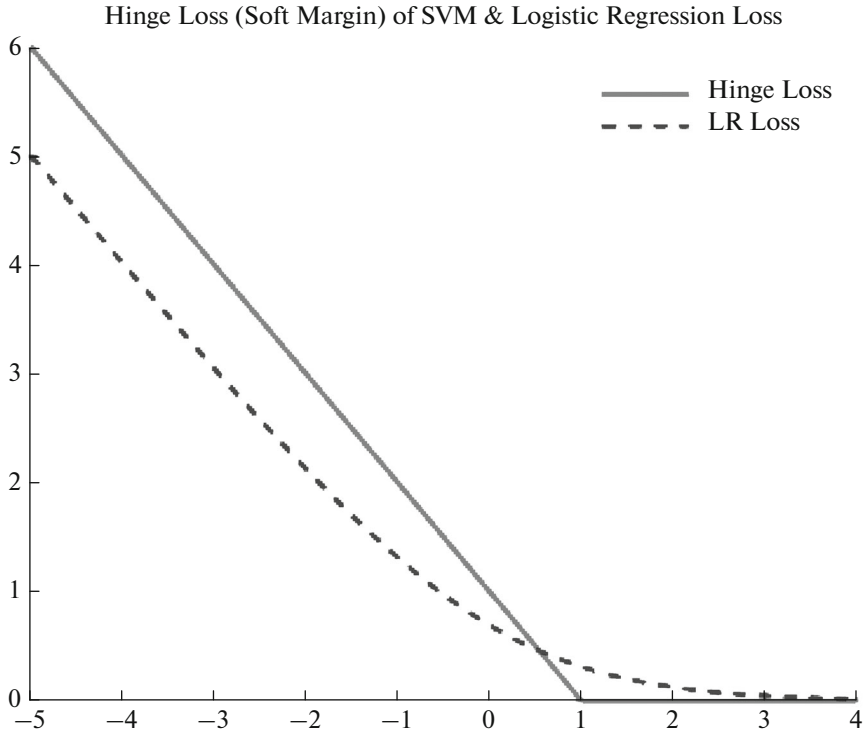

Fig. 2. SVM's hinge, and logistic regression's, loss functions.

Advantages of RKLR classifier

- it offers a natural estimate of the class probability $p(\mathbf{x}) = e^o \Big/ \left(1 + e^o\right)$
- it can be naturally generalized to the *multi*-class case through kernel multi-logit regression.

Disadvantages of RKLR classifier

- its output $o(\mathbf{x}) = \Sigma\alpha_i\, k(\mathbf{x}, \mathbf{x}_i) + b$, $i = 1, \ldots, l$ where *all* the weights α_i are nonzero values meaning that the RKLR classifier is dense.

The sparseness property is important for both the duration of a training phase (which for large datasets could be big indeed or, it can even last forever) and the final size of the model. This is why the authors in [13] have proposed a new, the so-called, Import Vector Machine (IVM) algorithm. The IVM is quite a sophisticated (read intricate) algorithm. This is why we will in next experiments show results obtained by the unique OLLA for RKLR. In other words, we aim to show that the RKLR model, under the OLLA environment proposed here, can be sparse and often sparser than SVM classifier while still keeping a descent accuracy.

The dataset in all experiments is a noisy two spirals data shown in Fig. 3. Gaussian kernel was used and we are focused here on comparisons in respect of accuracies and a size of final classifier (sparseness).

The results in Fig. 4 show that OLLA for RKLR creates much sparser classifier than SMO. For experiment number 12 (C = 1e5, and $\sigma = 0.03$) difference in accuracy is only 0.65% but OL RKLR model has three times less neurons in hidden layers, or expressed in terms of SVM, it has 3 times less support vectors.

In the next experiments we have slightly increased both penalty parameter C and bias b while all the other experimental conditions have remained same.

Figure 5 shows that SMO algorithm can easily achieve 100% accuracy but with much bigger model. The smallest difference in accuracy is in experiment 4 when OLLA for RKLR has smaller accuracy for 0.65% again but with 3 times smaller model. Note that in all the experiments here SVM by using SMO is always obtained by using both an regularizer $R = 0.5\mathbf{w}^t\mathbf{w}$ and bias term b.

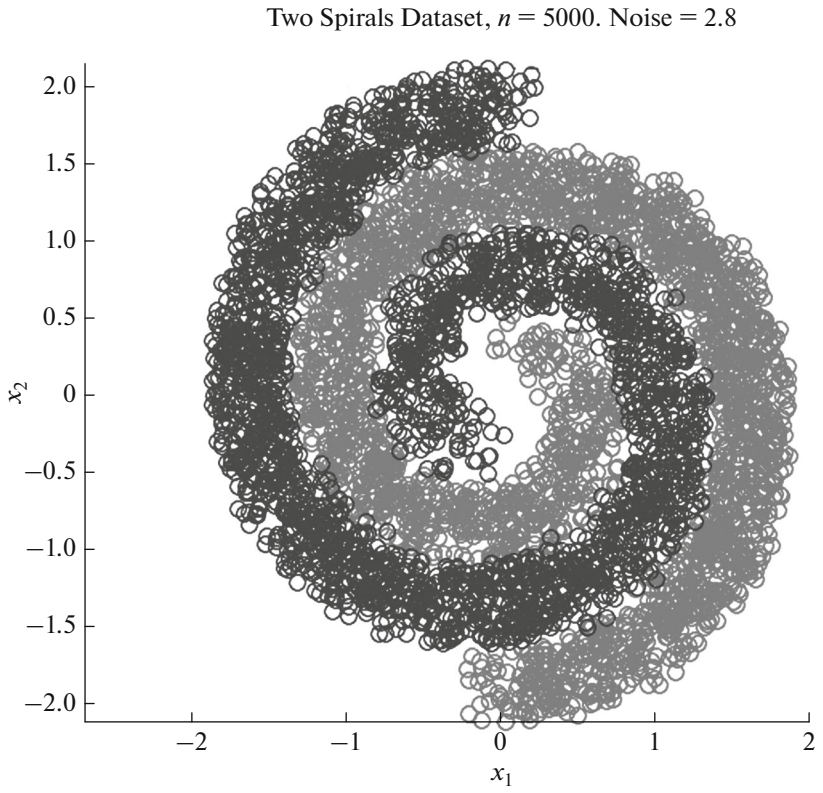

Fig. 3. A noisy two spirals data set in 2-dimensional space.

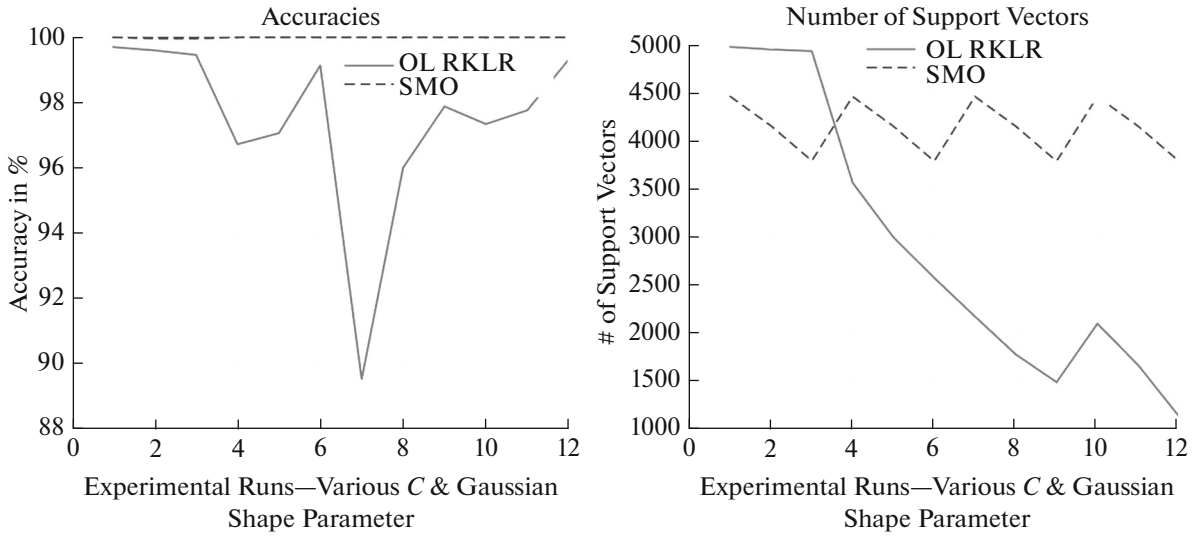

Fig. 4. Comparisons of performances: SVM trained by SMO vs. OLLA for RKLR on the two noisy spirals datasets. Gaussian kernel: $C \in$ [1e2 1e3 1e4 1e5], $\sigma \in$ [0.02 0.025 0.03], 2 epochs, OLLA results for RKLR are obtained *with* an regularizer $0.5\mathbf{w}'\mathbf{w}$ and *with* a bias term b. Iterations are performed cyclically.

Results in Fig. 6 are obtained without using a bias term b in RKLR. The other experimental conditions in Fig. 6 are same as in Fig. 4. One can readily see that differences in accuracy are very small, while the OLLA for RKLR creates sparser models.

The results in Fig. 7 are obtained without using an regularizer, but with a use of bias term b in RKLR model.

The regularization effect here is obtained with an early stopping. Namely, results are obtained by doing only 2 epochs. The effect of using an regularizer or doing some early stopping is commented and discussed in [11].

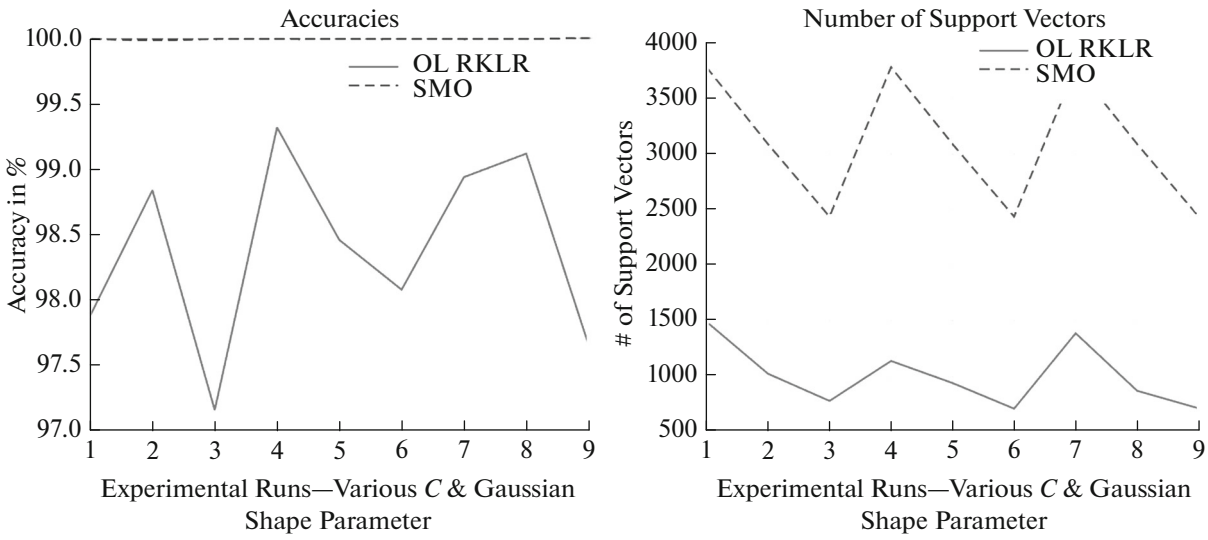

Fig. 5. Comparisons of performances: SVM trained by SMO vs. OLLA for RKLR on the two noisy spirals datasets. Gaussian kernel: $C \in$ [1e4 1e5 1e6], $\sigma \in$ [0.03 0.04 0.05], 2 epochs, OLLA results for RKLR are obtained *with* an regularizer $0.5\mathbf{w}'\mathbf{w}$ and *with* a bias term b. Iterations are performed cyclically.

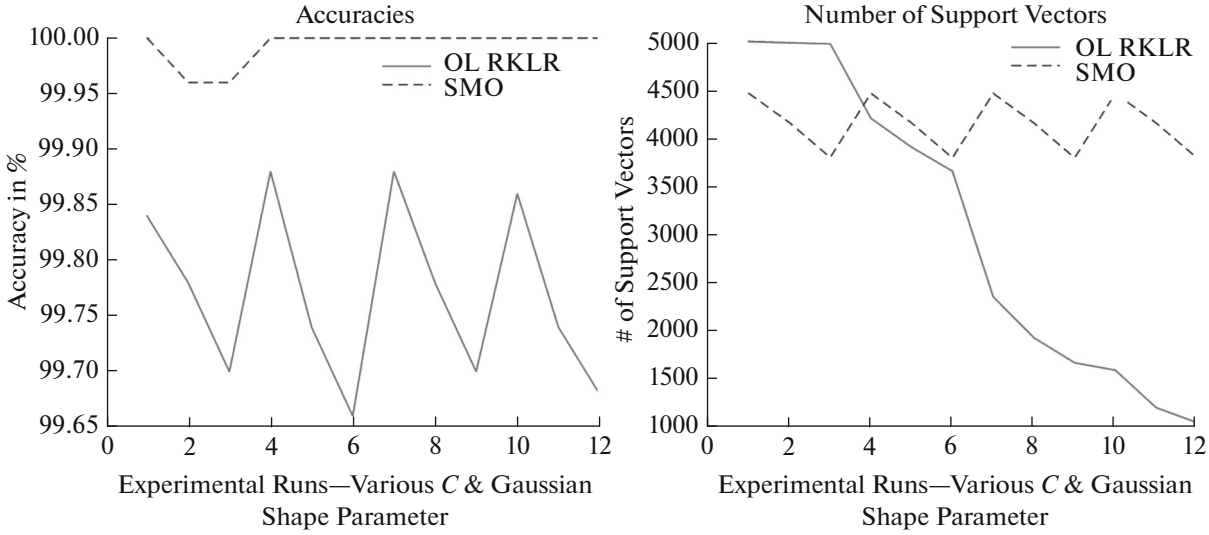

Fig. 6. Comparisons of performances: SVM trained by SMO vs. OLLA for RKLR on the two noisy spirals datasets. Gaussian kernel: $C \in$ [1e2 1e3 1e4 1e5], $\sigma \in$ [0.02 0.025 0.03], 2 epochs, OLLA results for RKLR are obtained *with* an regularizer $0.5\mathbf{w}'\mathbf{w}$ and *without* a bias term b. Iterations are performed cyclically.

Figure 8 shows the results of RKLR classifier without using both an regularizer and the bias term b. Results are both interesting and competitive for RKLR classifier—for a loss of 0.12% in accuracy, RKLR model is 3 times smaller.

The next plots in Fig. 9. are obtained under different experimental conditions for OL RKLR. Now, the update was not proceeded cyclically, meaning going in predefined order. These results are obtained by performing an update for the max violator in each iteration. This step brings a little CPU time burden because one has to find the worst violator data point (most negative $y_i o_i$) but, in this particular run, it increases the accuracy of RKLR classifier slightly while still keeping it much sparser than SVM trained by SMO.

The final comparisons are shown in Fig. 10.

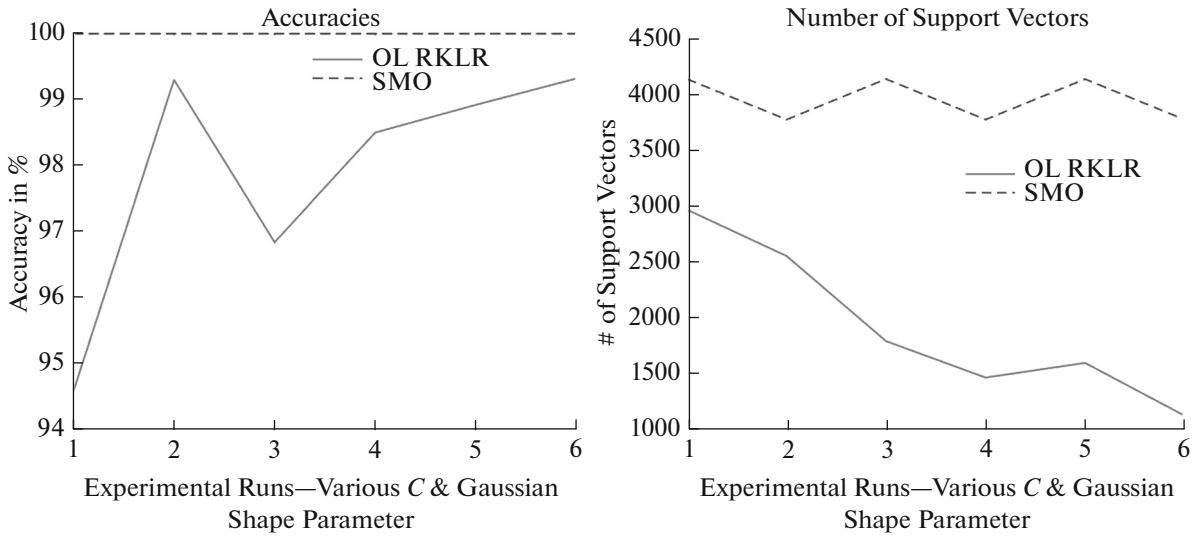

Fig. 7. Comparisons of performances: SVM trained by SMO vs. OLLA for RKLR on the two noisy spirals datasets. Gaussian kernel: $C \in$ [1e3 1e4 1e5], $\sigma \in$ [0.025 0.03], 2 epochs, OLLA results for RKLR are obtained *without* an regularizer and *with* a bias term *b*. Iterations are performed cyclically.

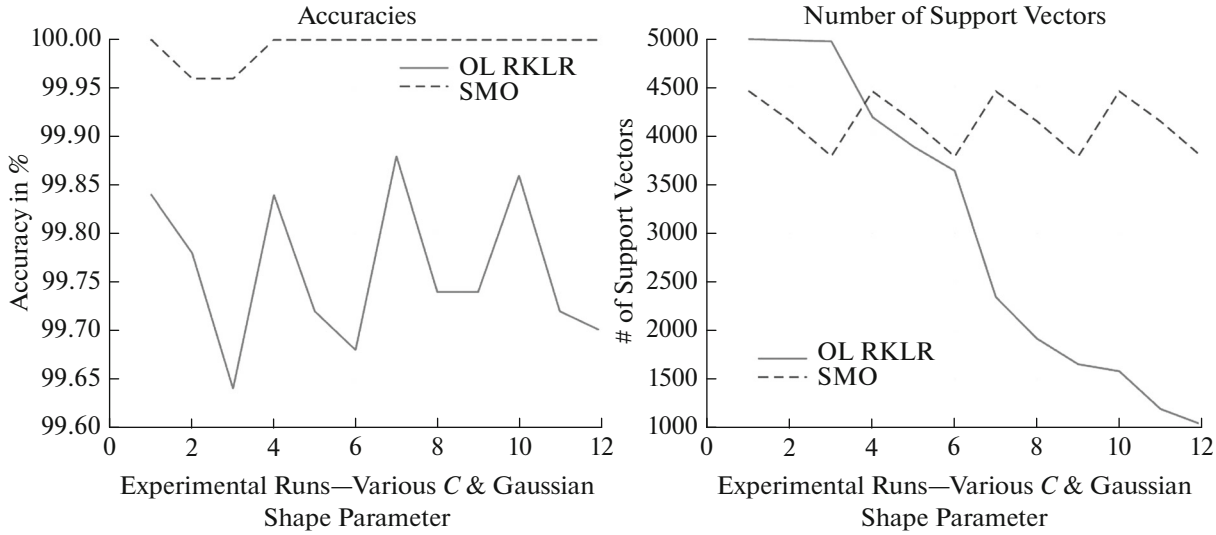

Fig. 8. Comparisons of performances: SVM trained by SMO vs. OLLA for RKLR on the two noisy spirals datasets. Gaussian kernel: $C \in$ [1e2 1e3 1e4 1e5], $\sigma \in$ [0.02 0.025 0.03], 2 epochs, OLLA results for RKLR are obtained *without* an regularizer and *without* a bias term *b*. Iterations are performed cyclically.

In Fig. 10, instead of a stochastic gradient, we implemented the Newton–Raphson method for updating the RKLR model. which uses a well known second order derivative in the iterative recalculation of the weight vector $\mathbf{w}$ as shown here $\mathbf{w} = \mathbf{w} - \partial R/\partial \mathbf{w}/\partial^2 R/\partial \mathbf{w}^2$.

The Newton–Raphson procedure is also in the core of our ISDA algorithm which is basically the Gauss–Seidel method for solving the SVM problems [18–19]. (It is worth of mentioning that the numeric software company Mathworks has implemented ISDA algorithm within its MATLAB classification code fitcsvm [20]).

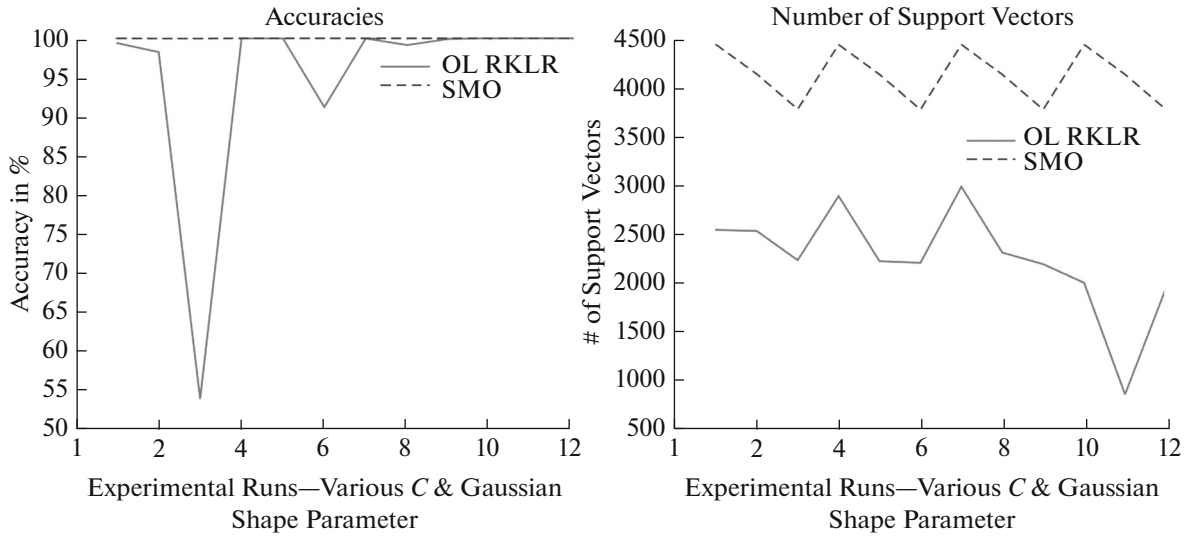

Fig. 9. Comparisons of performances: SVM trained by SMO vs. OLLA for RKLR on the two noisy spirals datasets. Gaussian kernel: $C \in$ [1e2 1e3 1e4 1e5], $\sigma \in$ [0.02 0.025 0.03], 2 epochs, OLLA results for RKLR are obtained *with* an regularizer $0.5\mathbf{w}^t\mathbf{w}$ and *with* a bias term b. Iterations are *not* performed cyclically, but by updating α_i of the *worst violator*.

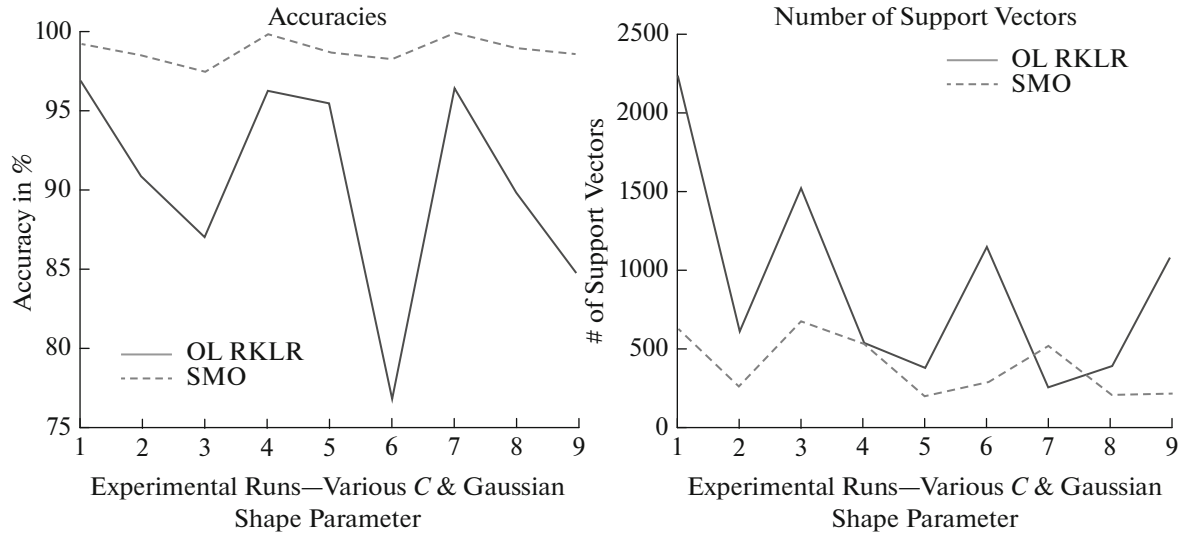

Fig. 10. Comparisons of performances: SVM trained by SMO vs. OLLA for RKLR on the two noisy spirals datasets. Gaussian kernel: $C \in$ [1e2 1e4 1e6], $\sigma \in$ [0.1 0.5 1], 5 epochs, OLLA results for RKLR are obtained *with* an regularizer $0.5\mathbf{w}^t\mathbf{w}$ and *without* a bias term b. The updates are based on Newton–Raphson method. Iterations are performed cyclically.

Note that the Newton–Raphson method doesn't use learning rate η, and the updates for $\mathbf{o}$ and α_i used in obtaining results in Fig. 10 differ from the ones shown in List 1 which are valid for the SGD based algorithm. For the Newton-Raphson iterative scheme the updates for $\mathbf{o}$ and α_i are

$$\mathbf{o} = \mathbf{o} + \omega \frac{\dfrac{Cy}{1+e^{yo}} - \alpha_i}{\dfrac{Ck_{ii}}{4\cosh\left(yo/2\right)^2} + 1} \mathbf{k}\left(:, \mathbf{x}_i\right),$$

$$\alpha_i = \alpha_i + \omega \frac{\dfrac{Cy}{1+e^{yo}} - \alpha_i}{\dfrac{Ck_{ii}}{4\cosh\left(yo/2\right)^2} + 1}.$$

ω has a meaning of an over-relaxation parameter which potentially can speed up the training (but, also, if wrongly chosen, it can slow it down). Because the risk function of RKLR is not a quadratic function the theoretical analysis and/or experiments would, probably, give some hints about proper ω values range in the future. The results in Fig. 10 are obtained by using $\omega = 1$.

CONCLUSIONS

Paper derives and shows a unique stochastic gradient based online learning algorithm for eight popular nonlinear i.e., kernelized, classifiers—L1 and L2 SVMs with both a quadratic and l_1 regularizer, regularized kernel logistic regression, regularized huberized hinge loss, exponential loss with l_1 regularizer and LS SVM with L2 regularizer. Despite the great variety of classifiers the unique OLLA is same. Two variants of OLLA are presented in Lists 1 and 2.

The update model is optimized for use on huge datasets. An important part of speeding the learning up is the way how the output vector $\mathbf{o}$ is updated. Two possibilities are offered. In List 1, in each iteration step a scalar value $(\Lambda - P)$ multiplies the kernel vector $\mathbf{k}(:, \mathbf{x}_i)$. Traditionally, o_i was calculated as $o_i = \mathbf{k}(\mathbf{x}_i, :)\boldsymbol{\alpha} + b$ in each iteration step. Such a vector product is computationally more expensive than here proposed product $(\Lambda - P)\mathbf{k}(:, \mathbf{x}_i)$. The further speeding up (as shown in List 2 by $o_i = \mathbf{k}(\mathbf{x}_i, \mathbf{x}_\mathbf{I})\boldsymbol{\alpha}_\mathbf{I} + b$) is obtained by a scalar product of sparser vectors $\mathbf{k}$ and $\boldsymbol{\alpha}$ calculated for support vectors only. Taking into account that learning can lead to sparse models (only a fraction of samples are support vectors used to build a model) some clever calculation and cashing strategy of the output o_i produces a very efficient code for ultra large datasets.

Additional speeding up is possible if updating in each iteration step was based on worst violator and not performed in a cyclic manner as shown in Lists 1 and 2. We have run some experiments and we can say that the update based on the worst violator brings some relevant and important speed-up.

Finally, we start experiments with an OLLA by using the second derivative (whenever existing and available) and by replacing the updates Λ and P shown in (3_{upd}) to (10_{upd}) by the new ones obtained by using very well known second order update $\mathbf{w} = \mathbf{w} - \partial R/\partial \mathbf{w}/\partial^2 R/\partial \mathbf{w}^2$ where $\mathbf{w}$ is the weight vector in a kernel feature space. This is known as the Newton-Raphson iteration scheme.

The investigations of the two later approaches, namely the use of the worst violator and/or the use of Newton-Raphson method, are at an initial stage. The theoretical insights, as well as the experiments, will tell more about their feasibility.

All the models but the last two (exponential loss with l_1 regularizer and LS SVM with l_2 regularizer) create sparse classifiers. The exponential loss with l_1 regularizer model often diverges which is obvious from the expression for its update coefficient Λ. Namely, for bigger negative values of $y_i o_i$ Λ increases enormously. Neither of the last two models can deal with very large datasets because number of support vectors equals the cardinality of dataset and the learning proceed extremely slow. In short, as of today, the OLLA doesn't work quite properly for these two models. This is also a good results saying that OLLA is not the best training tool for all the classifiers. Actually, there is no single approach which works best for all possible risk functions, meaning for all the possible classifiers.

Very interesting results are obtained for the RKLR classifier pointing that, by using OLLA, it can be very sparse. This result is contrary to the common knowledge in the field which says that RKLR is a dense classifier i.e., that all the training data participate in creating the final RKLR classifier model.

The proposed online learning algorithm is at the first stages of its theoretical and experimental verification and we expect a lot of fine theoretical developments and improvements, as well as a plenty of experimental results on the variety of datasets soon.

REFERENCES

1. Vapnik, V.N., Chervonenkis, A.Y., On the uniform convergence of relative frequencies of events to their probabilities. *(In Russian), Doklady Akademii Nauk USSR*, 1968, (4), pp. 181–186.
2. Cortes, C., Vapnik, V.N., Support Vector Networks. *Machine Learning* 1995, 20, pp. 273–297.
3. Vapnik, V.N., *The Nature of Statistical Learning Theory*, Springer, 1995.
4. Kecman, V., *Learning and Soft Computing, Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, The MIT Press, 2001.
5. Schölkopf, B., Smola, A. J., *Learning with Kernels*, The MIT Press, 2002.
6. Kivinen J., Smola A.J., Williamson R. C., *Large Margin Classification for Drifting Targets*, *Neural Information Processing Systems*, 2002.

7. Shalev-Shwartz, S., Singer, Y., Srebro, N., Pegasos: Primal estimated subgradient solver for SVM, *Proc. 24th Intl. Conf. on ML (ICML'07), ACM*, 2007, pp. 807–814.
8. Shalev-Shwartz, S., Ben-David, S., *Understanding Machine Learning—From Theory to Algorithms,* New York: Cambridge University Press, 2014.
9. Bottou, L., Large-Scale Machine Learning with Stochastic Gradient Descent, *Proc. 19th Int. Conf. on Comp. Stats., COMPSTAT'2010, Springer,* 2010, pp. 177–187.
10. Herbrich, R., *Learning Kernel Classifiers, Theory and Algorithms,* The MIT Press, 2002.
11. Collobert, R., Bengio, S., Links between Perceptrons, MLPs and SVMs, *Proc. of the 21*st *Intl. Conf. on Machine Learning, Banff, Canada*, 2004.
12. Steinwart, I., Christmann, A., *Support Vector Machines*, Springer, 2008.
13. Zhu, J., Hastie, T., Kernel Logistic Regression and the Import Vector Machine, *Journal of Computational and Graphical Statistics*, 2005, V. 14, No: 1, pp. 185–20.
14. Melki, G., Kecman, V., Speeding-Up On-Line Training of L1 Support Vector Machines, *Proc. of IEEE SoutheastConf, Norfolk, VA,* 2016.
15. Kecman, V., *Fast online algorithm for nonlinear support vector machines and other alike models*, *Seminar at VCU*, 2016.
16. Platt, J.C., Sequential Minimal Optimization, A Fast Algorithm for Training Support Vector Machines, *Microsoft Research, Technical Report MSR-TR-98-14,* 1998.
17. Chang, C-C. and Lin, C-J., "LIBSVM: A library for support vector machines". *In: ACM Transactions on Intelligent Systems and Technology 2, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm,* 2011, 27:1-27:27.
18. Kecman, V., Huang, T.-M., Vogt, M., Iterative Single Data Algorithm for Training Kernel Machines from Huge Data Sets: Theory and Performance, *In Support Vector Machines: Theory and Applications. Ed. Lipo Wang,* Berlin: Springer-Verlag, 2005, pp. 255–274.
19. Huang, T.-M., Kecman, V., Kopriva, I.*, Kernel Based Algorithms for Mining Huge Data Sets, Supervised, Semi-supervised, and Unsupervised Learning,* Springer-Verlag, Berlin, Heidelberg, see http://www.learning-from-data.com 2005.
20. http://www.mathworks.com/help/stats/fitcsvm.html.