NEUROINFORMATICS-2016
International Conference on Artificial Neural Networks

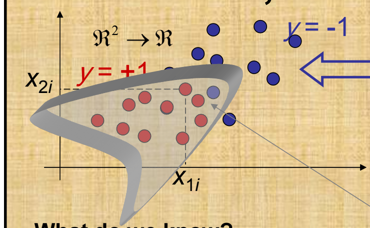
Fast, stochastic gradient descent (SGD) based, online learning algorithm (OLLA) for nonlinear (kernelized) support vector machines and other alike models for large datasets

Vojislav Kecman

Learning Algorithms and Applications Laboratory (LAAL)

VCU School of Engineering, COMPUTER SCIENCE

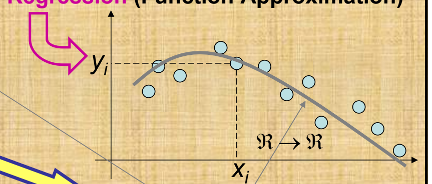
First, the basic notions



Supervised ML i.e., DM Tasks

Classification (Pattern Recognition)

Regression (Function Approximation)



What do we know?
Input vectors $\mathbf{x}_i$
Their labels y_i

Dataset

What do we want?
To find a dependency $o=f(\mathbf{x}, \mathbf{w})$ between input vectors $\mathbf{x}_i$ and labels y_i

Decision function in classification
Approximation i.e., Regression, function in regression

How will we do it?
By trying to be **close** to data but, **not too close**, and this will be achieved by **minimizing the RISK**

min $R(\mathbf{o}(\mathbf{x}, \mathbf{w})) = L + \lambda Reg = Loss + \lambda Regularizer$

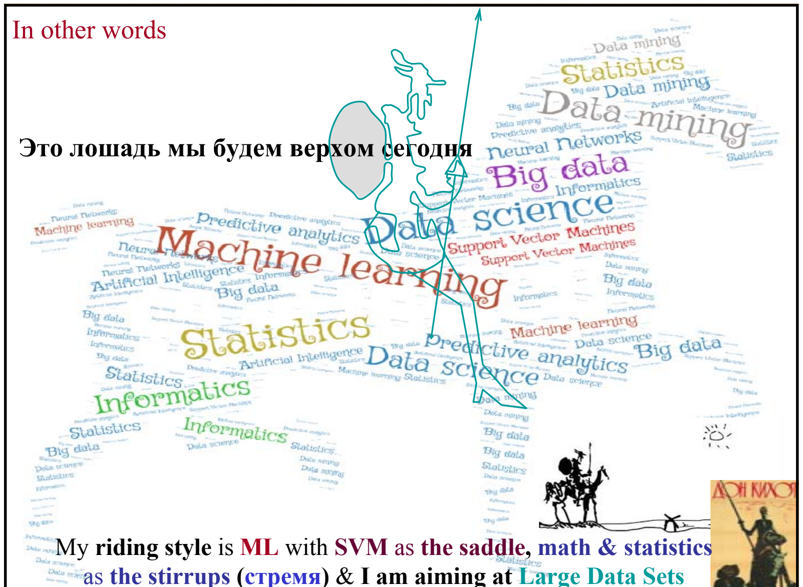
We design a function *o* from examples

Different Views i.e., Issues

- **Statistician**: How many examples I need to get good ***o***? What is the noise generator? What are the bounds? Are the data collected in a way that introduces bias? Are the data representative?
- **Mathematician**: How to represent ***o***? Is there a solution? Is it unique? Is it stable? Can the optimal solution be given in the closed form?
- **Operations Researcher**: What is the risk function? Is it convex? Are there any constraints? Are they linear?
- **Computer Scientist**: How to compute ***o*** efficiently? Are data in a sparse format or in a dense one? Is the solution sparse or dense? What to cash and when? Can it be done in memory? Parallelize it or not? Algorithmic complexity i.e., Scalability? Is the use of GP GPUs feasible? Should we go to clouds?

In other words

Это лошадь мы будем верхом сегодня



My riding style is ML with SVM as the saddle, math & statistics as the stirrups (стремя) & I am aiming at Large Data Sets

Well, without further ado here it is:

A **unique** and **single** **o**n **l**ine, SGD based, **l**earning **a**lgorithm (OLLA) for many NL classifiers

Define the number of epochs N. l is the 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 $\eta = \ln(t)/t$, or, …, something else

$i = i + 1$, $if\ i = l,\ i = 0,\ end$

Take $y_i o_i$ & calculate Λ & P

if $y_i o_i > 1$, $\Lambda = 0$, end ← Only for nondifferentiable risks (1)-(5)

$\mathbf{o} = \mathbf{o} + (\Lambda - \mathrm{P})\mathbf{k}(:, \mathbf{x}_i) + \Lambda\beta$

$\alpha_i = \alpha_i + \Lambda - \mathrm{P}$

$b = b + \Lambda\beta$ ← $\beta = 0$ if without bias b

end

Λ denotes a scalar originating from Loss, (slide 20)

P (rho) stands for a scalar coming from Regularizer (slide 20)

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

(see also [5] – [11])

L1 SVM with Regularizer w'w

(1) $R_{rl} = C\sum_{i=1}^{l} \underbrace{\max(0, 1 - y_i o_i)}_{L} + \boxed{\underbrace{\tfrac{1}{2}\|\mathbf{w}\|_2^2}_{R}}$

L2 SVM with Regularizer w'w

(2) $R_{rl} = C\sum_{i=1}^{l} \underbrace{\max\left(0, \tfrac{1}{2}\operatorname{sgn}(1 - y_i o_i)(1 - y_i o_i)^2\right)}_{L} + \boxed{\underbrace{\tfrac{1}{2}\|\mathbf{w}\|_2^2}_{R}}$

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

(3) $R_{rl} = C\sum_{i=1}^{l} \underbrace{\max(0, 1 - y_i o_i)}_{L} + \boxed{\underbrace{|\mathbf{w}|_1}_{R}}$

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

(4) $R_{rl} = C\sum_{i=1}^{l} \underbrace{\max\left(0, \tfrac{1}{2}\operatorname{sgn}(1 - y_i o_i)(1 - y_i o_i)^2\right)}_{L} + \boxed{\underbrace{|\mathbf{w}|_1}_{R}}$

Huberized hinge loss with Regularizer w'w

(5) $R_{rl} = C\sum_{i=1}^{l} \underbrace{L_{HHL}}_{L} + \boxed{\underbrace{\tfrac{1}{2}\|\mathbf{w}\|_2^2}_{R}}$

Kernel logistic regression with Regularizer w'w

(6) $R_{rl} = C\sum_{i=1}^{l} \underbrace{\ln\left(1 + e^{-y_i o_i}\right)}_{L} + \boxed{\underbrace{\tfrac{1}{2}\|\mathbf{w}\|_2^2}_{R}}$

Regularized exponential loss, Regularizer is $|\mathbf{w}|_1$ (Boosting)

(7) $R_{rl} = C\sum_{i=1}^{l} \underbrace{e^{-y_i o_i}}_{L} + \boxed{\underbrace{|\mathbf{w}|_1}_{R}}$

LS SVM, Quadratic Loss Function with Regularizer w'w

(8) $R_{rl} = C\sum_{i=1}^{l} \overbrace{\tfrac{1}{2}(1 - y_i o_i)^2}^{L} + \boxed{\overbrace{\tfrac{1}{2}\|\mathbf{w}\|_2^2}^{R}}$

Worth to Remember

$$\arg\min_{\mathbf{w}} Risk = Loss + \lambda Regularizer$$

Be CLOSE to data points **but, NOT TOO MUCH**

Loss is also known as **cost**, **merit**, **fitness**, **norm**, **objective**, **criterion function**, …, etc, etc, …

A 'CLOSE' i.e., a 'TOO MUCH', is controlled by the penalty i.e., tradeoff parameter $\boldsymbol{\lambda}$

What is the best (i.e., the right or, proper, tradeoff) parameter $\boldsymbol{\lambda}$?

Unfortunately, there is **hardly any way** to **know** the best $\boldsymbol{\lambda}$ ☹

CROSSVALIDATE!!!

However, let's first clarify the basic question which is:

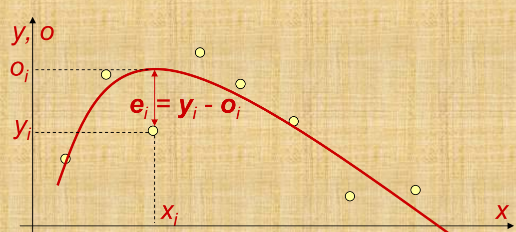
Why do we need both the Loss and the Regularizer?

RISK, or what does it mean '**be close, but not too much**'?

Task: Given the set of measurements ($\mathbf{x}_i$, y_i) .

Find: The dependency/output $o = f(\mathbf{x})$?

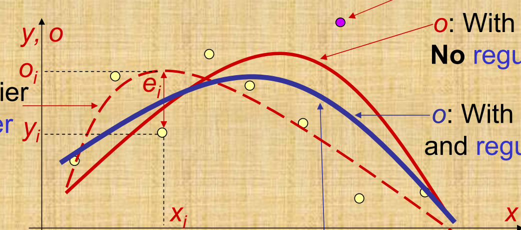
Assume the output is a quadratic function $o = w_1 x^2 + w_2 x + w_3$



o is parameterized by weights **w**, and our task is to **find weights**.
We'll do it by trying to minimize sum of error squares i.e.,
RISK is only the **loss**

$$min\ R = L + 0 = \tfrac{1}{2}\,\Sigma(y_i - o_i)^2 + 0$$

However, by an unlucky accident we've got a bad measurement point, outlier.
If previous risk $\Sigma(y_i - o_i)^2$ was used now, the outlier would distort previous nice approximation and the new one would become something as the solid red line

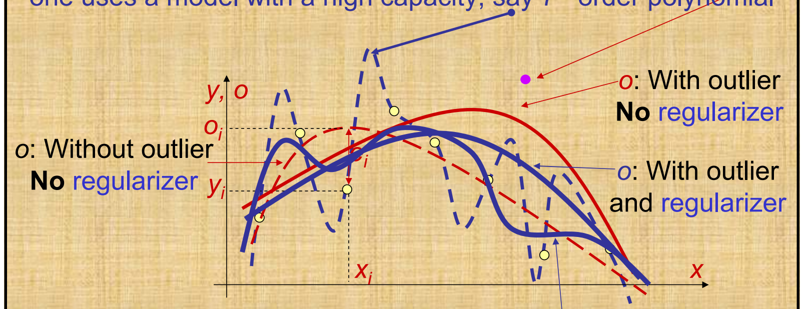


Obviously, you want to decrease the influence of the wrong measurements and you add, to your loss, the regularizer that doesn't allow your function to become wild, i.e., its weights to become too big. So, your new **risk** is

$$min\ R = L + Reg = \tfrac{1}{2}\,\Sigma(y_i - o_i)^2 + \tfrac{1}{2}\,\lambda\|\mathbf{w}\|^2$$

With an regularization, we'll get some better approximation despite the outliers.
$\boldsymbol{\lambda}$ is not known in advance and it should be found by a CROSSVALIDATION.

Don't be misled with an outlier! Even, without it we can miss it if, for example, one uses a model with a high capacity, say 7[th] order polynomial



Obviously, we want to decrease the influence of an **overly capacitive i.e., too 'good' a model.** We'll, AGAIN, need the regularizer that doesn't allow our function to become wild i.e., its weights to become too big. The **risk** is AGAIN

$$R = L + Reg = ½\,\Sigma(y_i - o_i)^2 + ½\,\lambda||\mathbf{w}||^2$$

With such an regularization, we'll get some better approximation AGAIN.

λ is to be found by a CROSSVALIDATION.

Before going any further let's take one important historical perspective which should shed more light on the origins of a Loss.

- Mid 18[th] century, my countryman☺, Serbian-Italian i.e. Ragusan, i.e., Dubrovnik, scientist **Ruđer Bošković** introduced **L1 norm** ⟹ **min $\Sigma_i|y_i - o_i|$ => min $\Sigma_i|e_i|$**

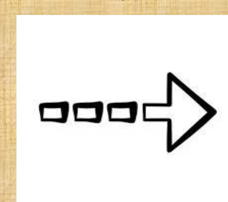
He solved it geometrically!

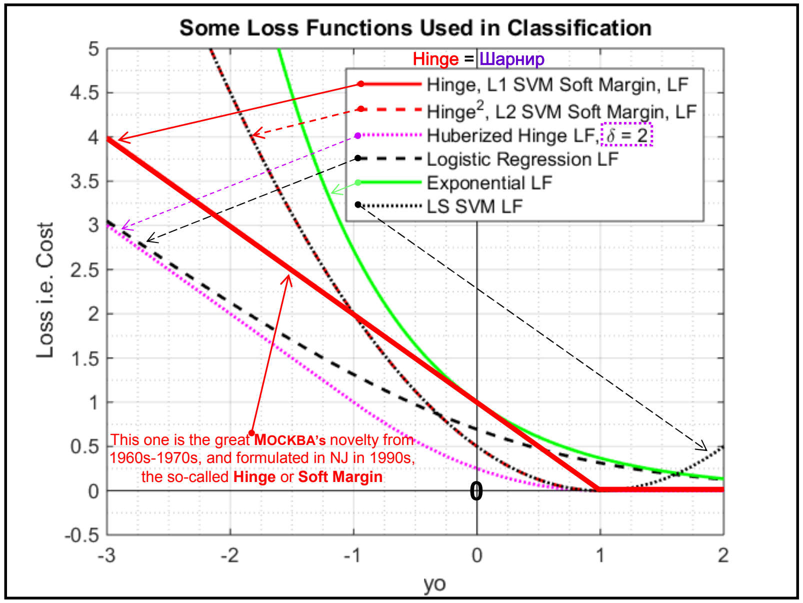
- End of 18[th] and beginning of 19[th] century there were French & German **Legendre** and **Gauss** who introduced **L2 norm**, **min ½ $\Sigma_i(y_i - o_i)^2$ => min ½ $\Sigma_i e_i^2$**
'Easy' to solve, and they solved it, analytically!

- And, so it was ⟹ sum-of-error-squares ruled the (data) science for more than 170 years. Still going strong!
- A lot has changed since 1960s-1970s, and all of it has started here in **МОСКВА**. That's when our story begins!

Well, except Bošković's L1, Legendre's & Gauss' L2 losses from 18[th]-19[th] century, what about the

novel, refreshed, more suited for modern high-dimensional and sparse data, **losses**?

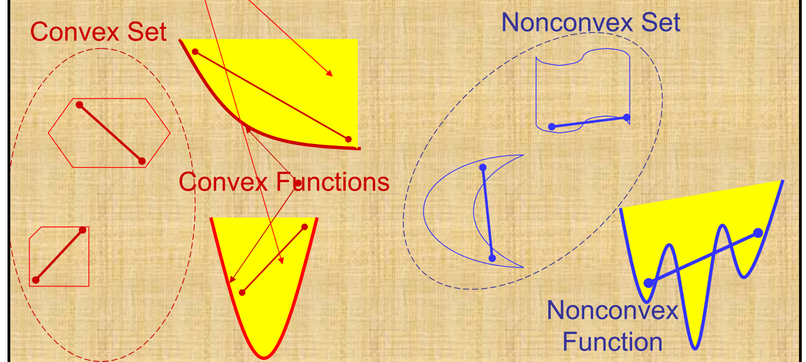




Note that all the losses are convex functions!

Why is convexity so important?

The epigraph (i.e. a set of points above the graph) of a convex function is a convex set.



Posing the problem 1st way

Consider the finite data set *D*: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l)$, $\mathbf{x} \in \Re^n$, $y \in \{+1, -1\}$, *l* data points pairs $(\mathbf{x}_i, y_i)$, assumed to be generated independently from the same distribution *P*.

Find a decision function (model) $o(\mathbf{x})$ which represents a mapping function *o*: $\Re^n \to \Re$ that (approximately) minimizes the risk (composed as a sum of a loss and an regularizer) given in (1).

$$\underbrace{\inf R_{rl}}_{o \in H} = C \underbrace{\sum_{i=1}^{l} L\left(y_i, o(\mathbf{x}_i)\right)}_{\text{Loss}} + \lambda \underbrace{\frac{1}{2}\|\mathbf{o}\|_2^2}_{\text{Regularizer}} \qquad (1)$$

Notice the slight change in using penalty parameter **C** instead of ***λ***

where ***H*** is a Reproducing Kernel Hilbert Space (RKHS), *L* denotes *loss* functionals, $\|\,.\,\|_2$ is the regularizer which is *L*2 norm and *rl* stands for *regularized loss*.

This problem posing differs from a standard risk used in SVMs.

Posing the problem 2nd way

A geometrical approach of a classic SVMs' ([1] – [5])

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

where H_o is a general Hilbert space and a function $o(\mathbf{w}, b)$ is defined in terms of an affine hyperplane specified, i.e., parameterized, 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])

Stochastic, i.e., Online, Learning

Well, let's minimize the Risk! In our stochastic GD **instead of minimizing**

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

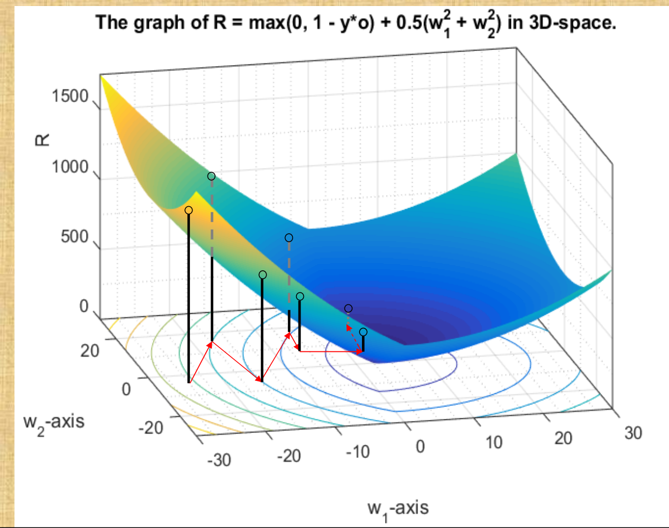
we minimize

$$\boxed{\underbrace{\min R_{rl}}_{(\mathbf{w},b)\in H_o x\Re} = CL\left(y_i, o_{(\mathbf{w},b)}(\mathbf{x}_i)\right) + \frac{1}{2}\|\mathbf{w}\|_2^2}$$

The adjective stochastic origins from the fact that we go from a data point $\mathbf{x}_i$ to the data point $\mathbf{x}_{i+1}$ randomly and **in each iteration step we minimize R_{rl}** by updating the weight vector **w** as in the good old perceptron, meaning by using gradient descent

$$\mathbf{w} = \mathbf{w} - \eta\frac{\partial R_{rl}}{\partial \mathbf{w}}$$

Stochastic Minimization



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

Update coefficients for all 8 models here:

$$\boxed{\begin{aligned}\alpha_i &= \alpha_i + \Lambda - \mathrm{P}\\ b &= b + \Lambda\beta\end{aligned}}$$

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

$\Lambda = \eta C y_i,$ $\qquad \mathrm{P} = \eta\alpha_i$

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

$\boxed{\Lambda = \eta C y_i(1 - y_i o_i), \qquad \mathrm{P} = \eta\alpha_i}$

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

$\Lambda = \eta C y_i,$ $\qquad \mathrm{P} = \eta\,\mathrm{sgn}(\alpha_i)$

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

$\Lambda = \eta C y_i(1 - y_i o_i),$ $\qquad \mathrm{P} = \eta\,\mathrm{sgn}(\alpha_i)$

Huberized hinge loss with Regularizer $\mathbf{w}'\mathbf{w}$

$y_i o_i \le 1-\delta \qquad \Lambda = \eta C y_i$

$1-\delta < y_i o_i \le 1 \qquad \Lambda = \dfrac{\eta C y_i(1 - y_i o_i)}{\delta}, \qquad \mathrm{P} = \eta\alpha_i$

Kernel logistic regression with Regularizer $\mathbf{w}'\mathbf{w}$

$\Lambda = \dfrac{\eta C y_i}{\left(1 + e^{y_i o_i}\right)}. \qquad \mathrm{P} = \eta\alpha_i$

Exponential loss with Regularizer $|\mathbf{w}|_1$ (Boosting)

$\Lambda = \dfrac{\eta C y_i}{e^{y_i o_i}}, \qquad \mathrm{P} = \eta\,\mathrm{sgn}(\alpha_i)$

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

$\boxed{\Lambda = \eta C y_i(1 - y_i o_i). \qquad \mathrm{P} = \eta\alpha_i}$

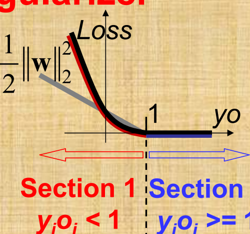
A curiosity: Compare L2 SVM and LS SVM both with $\mathbf{w}^t\mathbf{w}$ regularizer. **Updates** are **same**, **algorithms** are **not**! What's the difference? The LS SVM's loss function is both continuous and differentiable and, the tiny **loop $y_io_i > 1$** in the code **doesn't apply**. **Consequences**:

L2 SVM is sparse, LS SVM is dense!

Example: Derivation of an OL Algorithm for L2 SVM with $||w||^2$ Regularizer

Minimize

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



Section 1, y_io_i < 1:

$$\frac{\partial R_{rl}}{\partial \mathbf{w}} = -Cy_i\left(1 - y_i(\mathbf{w}^t\mathbf{x}_i + b)\right)\mathbf{x}_i + \mathbf{w}$$

$$\frac{\partial R_{rl}}{\partial b} = -Cy_i\left(1 - y_i(\mathbf{w}^t\mathbf{x}_i + b)\right)$$

Section 2, y_io_i >= 1: $\dfrac{\partial R_{rl}}{\partial \mathbf{w}} = \mathbf{w}, \qquad \dfrac{\partial R_{rl}}{\partial b} = 0$

Having the gradients, use the SGD equation $\mathbf{w} = \mathbf{w} - \eta\dfrac{\partial R_{rl}}{\partial \mathbf{w}}$ (slide 18), and an equivalent one for *b*, and you'll get the update coefficients for the L2 SVM in a kernel space as

$$\boxed{\begin{aligned}\Lambda &= \eta C y_i(1 - y_i o_i)\\ \mathrm{P} &= \eta\alpha_i\end{aligned}}$$

and the updates for the coefficients of expansions are

$$\boxed{\begin{aligned}\alpha_i &= \alpha_i + \Lambda - \mathrm{P}\\ b &= b + \Lambda\beta\end{aligned}}$$

SGD has been **known for "ages"** !!!
СГД был известен "возрастов" !!!

The whole NN field, and its famous EBP, in 1980s, and later, is based on the good old SGD. This includes the newest algorithm dubbed "Deep Learning" (DL). Here, at the conference Neuroinformatics 2016 in Moscow, there are both a plenary presentation & papers on **DL** and many articles on **NN**.

What is then, if anything,
new, different, exclusive,
in proposed **OLLA here?**

Actually, a few, but important things:

Classic NNs

1 The **risk** of classic NNs is typically **loss only**, which is sum-of-errors squares $R = \Sigma(y_i - o_i)^2$. Sure, they were regularizing too and this was dubbed ridge regression i.e., weight decay.
2 In addition, a classic NN optimizes both the HL weights (which define the positions & shapes of basis functions) and the OL ones. This then often leads to **non-convex** optimization.
3 Model's **sparsity**, i.e. size, represented by the # of HL neurons doesn't result from the learning. Crossvalidation is needed.

4 – For SVMs the **risk** is composed of both, basically, novel losses and, essentially, old regularizers but with a new meaning of e.g., $||\mathbf{w}||$.
5 - The so-called OL weights are optimized only while the HL ones are kept predefined i.e., fixed. **Convexity** ☺ .
6 - Also, kernels are applied here, which brought a lot of novel insights into the whole ML learning field.
7 – Model's **sparsity** (# of SVs/neurons) arises from the learning.

Novel SVMs

Issues (in fact, **Plenty of Issues**)

- **A: Related to Classifiers and Risks**
  - **1**: Should one use bias term *b* or not (if not, $\beta = 0$, otherwise $\beta = 1$) ?
  - **2**: Should one use regularizer and if so which one $||\mathbf{w}||_2$ or $|\mathbf{w}|_1$ or, ..???, & if not, P = 0, otherwise it is as given on slide 20) ?
- **B: Related to the SGD procedure**
  - **3**: Should the updates proceed cyclically
  - **4**: or, by using the worst violator ?
  - **5**: How many epochs (some experimental answers are in [14] and [15]) ?
  - **6**: What is the stopping strategy (some experimental answers are in [14,15])
  - **7**: How to calculate the final weights vector **α** (taking the last, averaging **α** over all iterations, averaging over the last 10% of **α** changes or…?[14]
  - **8**: How to choose learning rate η ? As $1/t$, or $2/\sqrt{t}$, or … ?
  - **9**: Can the use of *momentum* improve the convergence ?
  - **10**:Why not use the second order information i.e., why not perform a Newton-Raphson step instead of an SGD? Does it improve? **I think Yes!**
- **C: Related to the classic issues of kernelized classifiers**
  - **11**. Gaussian or Polynomial kernel, or, … ?
  - **12**. What about hyperparameter values *C, Gaussian's shape parameter σ, polynomial order,…*? Well, we always have **CROSSVALIDATION**

However, despite all of the mentioned, and not mentioned, issues, OLLA works fine ☺

OLLA performs really good on many datasets. Why & How ?

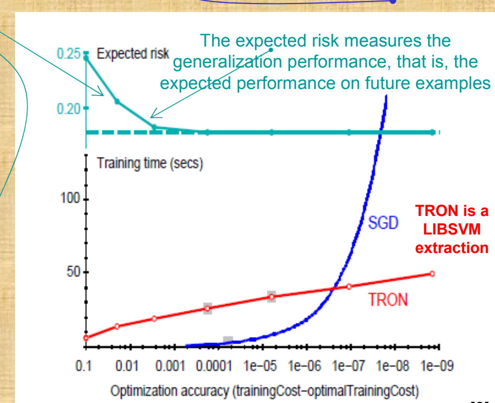
SGD based OLLA works very well indeed
Why & How?

bias

$$\min \quad \mathcal{E} = \mathcal{E}_{\mathrm{app}} + \mathcal{E}_{\mathrm{est}} + \mathcal{E}_{\mathrm{opt}}$$

variance

"Although the stochastic gradient algorithms, SGD … are **clearly the worst optimization algorithms**, they need less time than the other algorithms to **reach a predefined expected risk**. Therefore, in the large scale setup, that is, when the limiting factor is the computing time rather than the number of examples, the SGD performs asymptotically better!" See [9].



see [9]

Claims about OLLA feasibility for large datasets 1

- First, **what is the large dataset**?
- It's a **changing notion**, but as of **today** we'll use adjective *large* whenever, say, one has around, more or much more than 1 million samples, or

  stated in PC's CPU training time, whenever CPU time goes over, say, 1 day, or

  in terms of PC's memory, whenever we can't store data and/or design matrices in memory,

Claims about OLLA feasibility for large datasets 2

Major speeding up "trick" is implemented when instead of a 'classic' calculation of o_i by computing a vector, *dot,* product $\boldsymbol{k}_i$*$\boldsymbol{\alpha}$ as

$$o_i = \mathbf{k}\left(\mathbf{x}_i,:\right)\boldsymbol{\alpha} + b$$

we used a scalar-vector product in the line

$$\mathbf{o} = \mathbf{o} + \left(\Lambda - \mathrm{P}\right)\mathbf{k}\left(:,\mathbf{x}_i\right) + \Lambda\beta$$

For large datasets, this is an even larger "trick" ☺

Claims about OLLA feasibility for large datasets 3

The second speeding up "trick" is implemented by expressing update coefficients **Λ in terms of** (already calculated) $\boldsymbol{y_i o_i}$ **value** for all models

(except for L1 SVM. Its Λ is constant i.e., it doesn't depend upon $\boldsymbol{y_i o_i}$).

Check Λ-s on slide 20.

Claims about an OLLA feasibility for large datasets 4

The proposed OLLA produces (mostly, but not always & not for the last 2 classifiers) **sparse models** (**many $\boldsymbol{\alpha_i}$ = 0**) and an efficient code shall readily perform both an iterative calculation of only $\mathbf{k}(:, \mathbf{x}_i)$ & an effective cashing strategy for $\mathbf{k}_i$s.

Calculation of $\mathbf{k}_i$ is the most time consuming operation. For 1 million samples, $\mathbf{k}_i$ is an $(10^6,1)$ vector. (Parallelization, Use of GPUs or Clouds might help).

Experimental Results 1:

OL_Classifiers

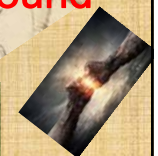
Comparisons of performances of SMO and OL L1 SVM Algorithm

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 **2,427** / 20	36.05	**14.55**	5.67
Prototask, **8,192** / 21	33.57	**11.28**	5.61
Reuters **11,069** / 8,315	5.65	**4.48**	4.10
Chess board, **8,000** / 2	13.62	**4.27**	7.65
Two normally distributed classes, **15,000** / 2	28.25	**7.82**	41.50

Results of 5-fold CV. OL L1 SVM **WITHOUT** Regularizer

Setting the stage for the next experiments and results





Но люди - все грешные души.
У многих глаза - что клыки.
С соседней деревни Криуши
Косились на нас мужики.
Житье у них было плохое -
Почти вся деревня вскачь
Пахала одной сохою
На паре заезженных кляч.

Well, there are still villages around

Каких уж тут ждать обилий, -
Была бы душа жива.
Украдкой они рубили
Из нашего леса дрова.
Однажды мы их застали ...
Они в топоры, мы тож.
От звона и скрежета стали
По телу катилась дрожь.

АННА СНЕГИНА, СЕРГЕЙ ЕСЕНИН

Statisticianskoye

where Statisticians, Operation Researchers and Mathematicians live

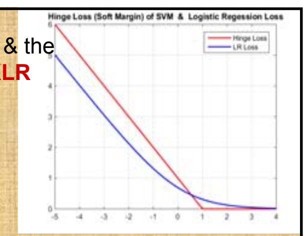
and, they fight (compete) very fiercely with ideas, experiments and papers.


There are no axes (yet), but there is still a kind of "**По телу катилась дрожь**"

Engineerskoye

where Computer Scientists, Data miners, Physicist and Engineers live

Some discussion on Kernel Logistic Regression & the Import Vector Machine [13] and how, actually, **RKLR can become sparse by using OLLA**



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

Claims, or 'common knowledge', about RKLR

Advantages of RKLR:

•Offers a natural estimate of the class probability p(**x**).

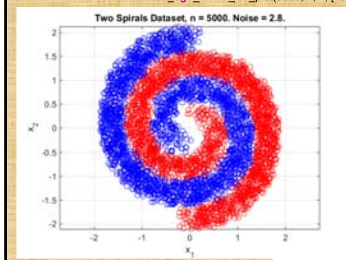
•Can naturally be generalized to the M-Class case through kernel multi-logit regression.

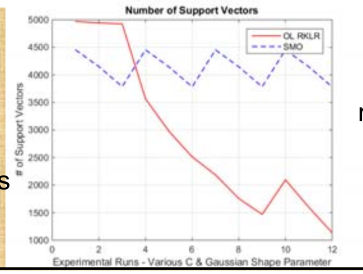
Disadvantages of RKLR:

•RKLR's solution is $o(\mathbf{x}) = \Sigma\alpha_i k(\mathbf{x}, \mathbf{x}_i)$ but all the α_i are nonzero i.e., the solution **is not sparse.** So, [13] **proposed IVM. Quite a sophisticated (read, complicated) algorithm.** Believe it, or try it!!!

Experimental Results 2:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e2 1e3 1e4 1e5], 2, 1, [0.02 0.025 0.03],1,1, 1, 2, 1, 1, 1., 1.,1,0);





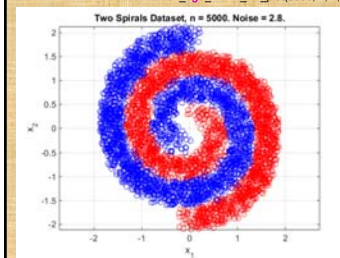
WITH regularization

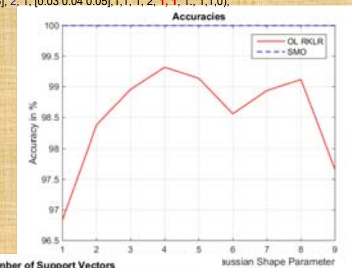
WITH bias

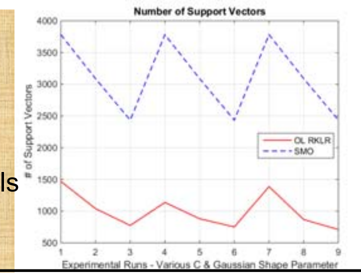
OLLA produces **sparse RKLR** models **naturally**

Experimental Results 3:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e4 1e5 1e6], 2, 1, [0.03 0.04 0.05],1,1, 1, 2, 1, 1, 1., 1.,1,0);







WITH regularization

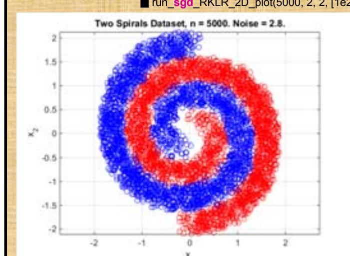
WITH bias

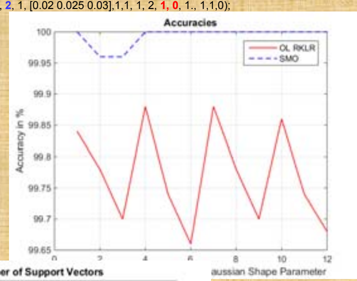
Same as previous slide but slightly different C and sigma

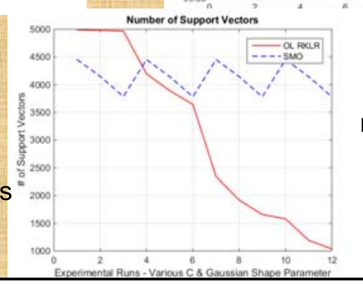
OLLA produces **sparse RKLR** models **naturally**

Experimental Results 4:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e2 1e3 1e4 1e5], 2, 1, [0.02 0.025 0.03],1,1, 1, 2, 1, 0, 1., 1.,1,0);







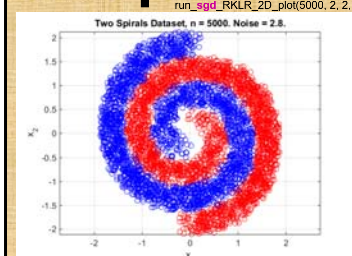
WITH regularization

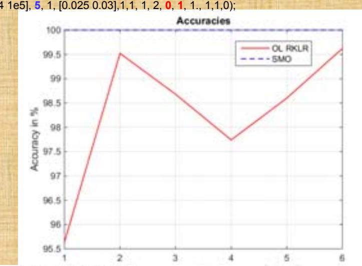
WITHOUT bias

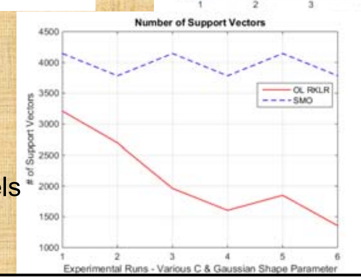
OLLA produces **sparse RKLR** models **naturally**

Experimental Results 5:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e3 1e4 1e5], 5, 1, [0.025 0.03],1,1, 1, 2, 0, 1, 1., 1.,1,0);







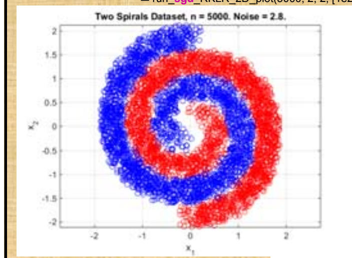
WITHOUT regularization

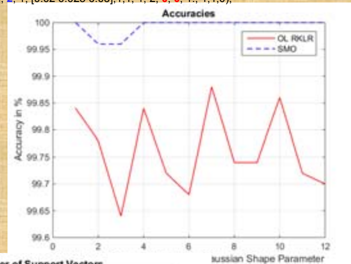
WITH bias

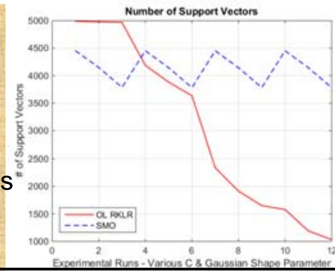
OLLA produces **sparse RKLR** models **naturally**

Experimental Results 6:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e2 1e3 1e4 1e5], 2, 1, [0.02 0.025 0.03],1,1, 1, 2, 0, 0, 1., 1,1,0);







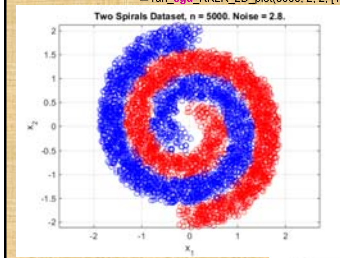
OLLA produces **sparse RKLR** models **naturally**

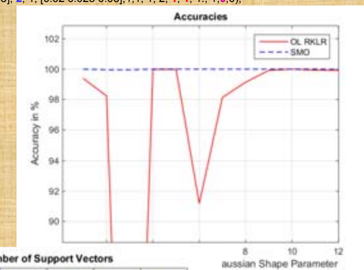
WITHOUT regularization

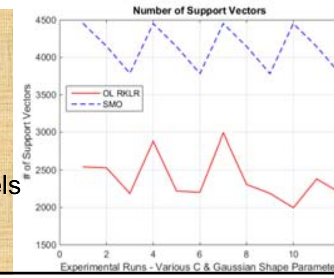
WITHOUT bias

Experimental Results 7:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e2 1e3 1e4 1e5], 2, 1, [0.02 0.025 0.03],1,1, 1, 2, 1, 1, 1., 1,0,0);







OLLA produces **sparse RKLR** models **naturally**

WITH regularization

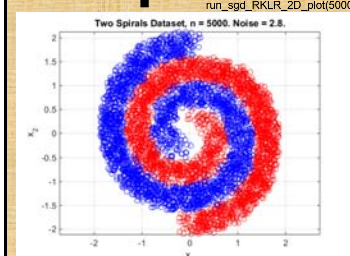
WITH bias, SGD

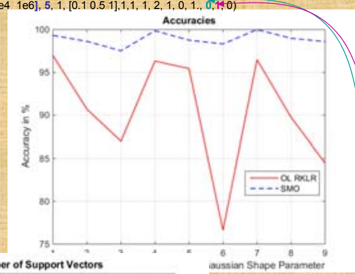
NO CYCLING

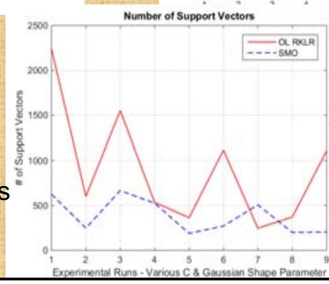
MAX Violator

Experimental Results 8:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e2 1e4 1e6], 5, 1, [0.1 0.5 1],1,1, 1, 2, 1, 0, 1., 0,1, 0)







OLLA produces **sparse RKLR** models **naturally**

WITH regularization

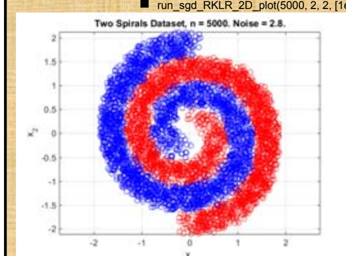
WITHOUT bias,

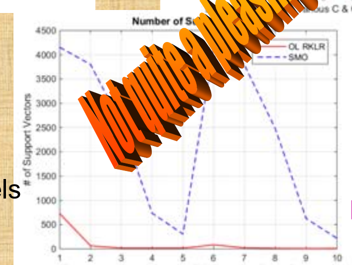
Newt-Raphs

CYCLING

Experimental Results 9:

run_sgd_RKLR_2D_plot(5000, 2, 2, [1e2 1e2], 2, 1, [0.025 0.03 0.05 0.1 0.25],1,1, 1, 2, 1, 1, 1., 0,0);





OLLA produces **sparse RKLR** models **naturally**

WITH regularization

WITH bias,

Newt-Raphs

NO CYCLING

MAX Violator

LET's WRAP IT UP !!!

And, here it is again:
A unique single SGD based
OLLA for many NL classifiers

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

(1) $\Lambda = \eta C y_i$, $\quad \mathrm{P} = \eta\alpha_i$

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

(2) $\Lambda = \eta C y_i (1 - y_i o_i)$, $\quad \mathrm{P} = \eta\alpha_i$

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

(3) $\Lambda = \eta C y_i$, $\quad \mathrm{P} = \eta\,\mathrm{sgn}(\alpha_i)$

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

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

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

(5) $y_i o_i \le 1-\delta \quad \Lambda = \eta C y_i$

$1-\delta < y_i o_i \le 1 \quad \Lambda = \dfrac{\eta C y_i (1 - y_i o_i)}{\delta}$, $\quad \mathrm{P} = \eta\alpha_i$

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

(6) $\Lambda = \dfrac{\eta C y_i}{\left(1 + e^{y_i o_i}\right)}$, $\quad \mathrm{P} = \eta\alpha_i$

Exponential loss with Regularizer $|\mathbf{w}|_1$ (Boosting)

(7) $\Lambda = \dfrac{\eta C y_i}{e^{y_i o_i}}$, $\quad \mathrm{P} = \eta\,\mathrm{sgn}(\alpha_i)$

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

(8) $\Lambda = \eta C y_i (1 - y_i o_i)$, $\quad \mathrm{P} = \eta\alpha_i$

Define the number of epochs N, l is a number of data

Shuffle the dataset & Initialize $\boldsymbol{\alpha} = \mathbf{0}$, $\mathbf{o} = \mathbf{0}, b = 0, i = 0$

$for\ t = 1,\ldots,Nl$

$\quad \eta = \sqrt{2/t}$, or $\eta = 1/t$, or $\eta = \ln(t)/t$, or,…,something else

$\quad i = i+1, \quad if\ i = l, \quad i = 0, \quad end$

$\quad$ Take $y_i o_i$ & calculate Λ & P

$\quad$ if $y_i o_i > 1$, $\Lambda = 0$, end $\leftarrow$ Only for nondifferentiable risks (1)-(5)

$\quad \mathbf{o} = \mathbf{o} + (\Lambda - \mathrm{P})\mathbf{k}(:, \mathbf{x}_i) + \Lambda\beta$

$\quad \alpha_i = \alpha_i + \Lambda - \mathrm{P}$

$\quad b = b + \Lambda\beta$

end

$\beta = 0$ if without bias b

Λ denotes a scalar originating from **L**oss,
P (**rho**) stands for a scalar coming from **R**egularizer

Once the learning is over, for any new x $\Longrightarrow$ $o(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b$

Basic **Remarks** and **Comments** 1

- The **unique** and **single** online, SGD based, learning algorithm for a variety of nonlinear (kernel) classifiers is derived and introduced.
- Algorithm works for all datasets, but it is aimed for large ones.
- There are many open issues related to the new algorithm, but the very first experimental results point to both an efficient/fast and accurate algorithm.
- Some theoretical insights and experiments on real datasets should possibly resolve some issues satisfactorily.
- The approach used & presented may be helpful in developing learning algorithms for other classifiers or regression models.
- An efficient output vector $\mathbf{o}(\mathbf{x})$ update and cashing of kernel columns $\mathbf{k}_i$ is crucial for achieving the faster speed.

Basic **Remarks** and **Comments** 2

- Well, ComputeristaS ☺
- Final Thoughts  & Recommendations 

- **A unique** and **single** learning algorithm for **various nonlinear i.e., kernelized, classifiers** is presented & recommended here.
- It is **extremely simple**, and all what it is asking for is a **great enthusiastic PROGRAMMER & a lot of EXPERIMENTING**.
- C++, R, Python, Java, CUDA, Julia, Hadoop & Hive supported by Kafka or Storm, Scala & who-knows-which-one-else, …, (including a honorably mentioned MATLAB), are offering themselves.
- Sure, some (already mentioned) practical and theoretical issues will be met and addressed during the course of implementation **but**

THe VeRy fiRst EXperimental ResUlts arE veeeerY PromisinG

Самые первые экспериментальные результаты являются весьма перспективными

Заключительная мысль
The final thought

SGD based OLLA seems really to be The Algorithm ☺

It is very new and hence there are not too many results.

But, it is simpler, faster and more accurate than, or equal to, the other well established machine learning tools.

Consequently,

it deserves more thorough theoretical investigation as well as more experimenting, applications and comparisons.

References

1 Vapnik, V.N., A.Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities. (in Russian), Doklady Akademii Nauk SSSR, 181, (4), 1968.

2 Cortes, C., Vapnik, V.N., Support Vector Networks. Machine Learning 20:273-297, 1995.

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., A. J. Smola. 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., Y. Singer, N. Srebro. Pegasos: Primal estimated subgradient solver for SVM. Proc. 24th Intl. Conf. on ML (ICML'07), pp. 807-814. ACM, 2007.

8 Shalev-Shwartz S., S. Ben-David, 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, pp. 177-187, Springer, 2010.

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 21st 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, Journ. of Computational and Graphical Statistics, V. 14, No: 1, pp. 185-20, 2005.

14 Melki G., Kecman V., Speeding-Up Online Training of L1 Support Vector Machines, IEEE SoutheastConf, Norfolk, VA, 2016.

15 Kecman V., Melki G, Fast Online Algorithm for SVMs, IEEE SoutheastConf, Norfolk, VA, 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, 27:1-27:27, 2011.