PERGAMON

Pattern Recognition 36 (2003) 1985–1996

PATTERN RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

An iterative algorithm learning the maximal margin classifier

Vojtěch Franc*, Václav Hlaváč

Center for Machine Perception, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, 121 35 Prague 2, Karlovo náměstí 13, Czech Republic

Received 15 January 2003; accepted 15 January 2003

Abstract

A simple learning algorithm for maximal margin classifiers (also support vector machines with quadratic cost function) is proposed. We build our iterative algorithm on top of the Schlesinger–Kozinec algorithm (S–K-algorithm) from 1981 which finds a maximal margin hyperplane with a given precision for separable data. We suggest a generalization of the S–K-algorithm (i) to the non-linear case using kernel functions and (ii) for non-separable data. The requirement in memory storage is linear to the data. This property allows the proposed algorithm to be used for large training problems.

The resulting algorithm is simple to implement and as the experiments showed competitive to the state-of-the-art algorithms. The implementation of the algorithm in Matlab is available. We tested the algorithm on the problem aiming at recognition poor quality numerals.

Keywords: Pattern recognition; Linear classifier; Supervised learning; Support vector machines; Kernel functions

1. Introduction

Maximal margin classifiers find a decision strategy which separates the training data with the maximal margin. This formulation leads to minimization of the structural risk and to favorable generalization capabilities of the classifier which is justified by both the theoretical studies [1,2] and the experimental evidence. The Support Vector Machines [2] transform the supervised learning of the maximal margin classifier to the quadratic programming problem. Although the quadratic programming is well explored, complicated optimization algorithms and computationally demanding processing of a large amounts of data is needed.

Our work builds on a simple on-line supervised learning algorithm which finds a linear maximal margin classifier with a prescribed precision. This fast iterative algorithm with proved monotone convergence was invented by Schlesinger et al. [3]. However, the S–K-algorithm has not been widely known. Schlesinger's approach relaxes the requirement for the exact solution to the ε-solution which yields a linear decision rule with the margin differing from the optimal one by at most a prescribed constant ε. The algorithm is iterative and uses the update rule proposed earlier by Kozinec [4]. We will call the algorithm the Schlesinger–Kozinec algorithm to pay tribute to its authors. The abbreviation S–K-algorithm will be used thereinafter. A wider analysis of the S–K-algorithm can be found in the monograph [5].

The S–K-algorithm has several advantages: (a) fast convergence to the solution, (b) the ability to process data on-line, and (c) it can be simply programmed. The disadvantage of the S–K-algorithm is that only linear classifiers can be found and linearly separable data are required.

Our contribution reported here removes the mentioned disadvantage. The S–K-algorithm is generalized to cope with the non-linear decision problem by incorporating kernel functions [6] and solves the non-separable case. Our experience shows that the proposed algorithm is competitive with Support Vector Machines while its programming is considerably simpler.

Results presented in this paper are related to some previous work on support vector machines. The sequential min-

* Corresponding author. Tel.: +420-2-2435-7305; fax: +420-2-2435-7385.

E-mail addresses: xfrancv@cmp.felk.cvut.cz (V. Franc), hlavac@cmp.felk.cvut.cz (V. Hlaváč).

0031-3203/03/$30.00
doi:10.1016/S0031-3203(03)00060-8

imal optimization (SMO) algorithm [7] and decomposition based strategies, e.g. as implemented in SVM^{light} [8], belong to the standard approaches seeking the SVM classifiers. We found two other papers which are probably closest to our work. The first paper by Li and Long [9] describes a training algorithm which yields a linear separating rule. The algorithm is called relaxed online maximum margin algorithm (ROMMA) and copes with the non-linear case. The proposed adaptation rule is similar to ours. However, the ROMMA algorithm does not consider bias in the decision function and does not use the concept of margin to define the stopping condition. The second paper by Keerthi [10] publishes a very nice iterative algorithm for support vector machine classifier design. The method was derived by adapting algorithms from computational geometry seeking the nearest points between two convex polytopes. The derived adaptation rule is more complex than ours. The advantage of Keerthi's method is that less support vectors are generated. For other related work we refer to Refs. [11,12].

This paper is organized as follows. Section 2 defines the tasks of finding the separating hyperplanes. Section 3 describes the use of kernel functions leading to the generalization of separating hyperplanes to the non-linear decision rules. In Section 4 we first gradually explain the Schlesinger–Kozinec algorithm which constitutes the basis of our new extension and second we introduce steps which lead to a generalization which solves the non-linear and non-separable cases. The relation of the proposed algorithm to other approaches is described in Section 5. The conducted experiments are given in Section 6 and Section 7 concludes the paper.

2. Separating hyperplane

The sets $X_1 = \{x_i : i \in I_1\}, X_2 = \{x_i : i \in I_2\}$ denote patterns from the training multi-set for the first and the second class, respectively. The $I = I_1 \cup I_2$ are the corresponding sets of indices. The patterns are assumed to be from an n-dimensional vectors space $\mathcal{X}$. The numbers $y_i = +1$, $i \in I_1$ and $y_i = -1$, $i \in I_2$ denote labels of the patterns.

The classes, i.e., vector sets X_1 and X_2 are linearly separable if there exist a vector w and a threshold b for which the set of inequalities

$$\begin{aligned} &\langle w, x_i \rangle + b > 0, \quad i \in I_1, \\ &\langle w, x_i \rangle + b < 0, \quad i \in I_2, \end{aligned} \tag{1}$$

holds. The notation $\langle\,\rangle$ stands for the dot (scalar) product. The vector w and the threshold b determine a separating hyperplane $\langle w, x \rangle + b = 0$. If task (1) has a solution then there is an infinite number of solutions corresponding to possible separating hyperplanes. There is one separating hyperplane among them which maximizes distance to the sets X_1 and X_2. The distance between a hyperplane and the sets X_1, X_2 is determined by the points which are the closest to the hyperplane (called support vectors in the support vector machines literature),

$$m(w, b) = \min_{i \in I_1 \cup I_2} \left(\frac{y_i \cdot (\langle w, x_i \rangle + b)}{\|w\|} \right).$$

The *optimal separating hyperplane* [2] is introduced which is the result of

$$(w^*, b^*) = \operatorname*{argmax}_{w,b} m(w, b) \tag{2}$$

and maximizes the distance $m(w, b)$. The real positive number $m^* = m(w^*, b^*)$ is the maximal margin between the sets X_1 and X_2. Task (2) can be equivalently expressed as

$$(w^*, b^*) = \operatorname*{argmin}_{w,b} \tfrac{1}{2} \|w\|^2 \tag{3}$$

subject to

$$\begin{aligned} &\langle w, x_i \rangle + b \geqslant +1, \quad i \in I_1, \\ &\langle w, x_i \rangle + b \leqslant -1, \quad i \in I_2, \end{aligned} \tag{4}$$

where the canonical expression of the hyperplane is used. If w^* and b^* solve task (3) then the optimal margin is equal to $m(w^*, b^*) = 1/\|w\|$ which is obvious if we divide (4) by $\|w\|$.

If the sets X_1 and X_2 are not linearly separable then conditions (4) cannot be satisfied and the optimal hyperplane does not exist. The *generalized optimal hyperplane* [2] is introduced as a solution of

$$(w^*, b^*, \xi_i^*) = \operatorname*{argmin}_{w,b,\xi_i} \frac{1}{2} \|w\|^2 + C \sum_{i \in I_1 \cup I_2} (\xi_i)^d \tag{5}$$

subject to

$$\begin{aligned} &\langle w, x_i \rangle + b \geqslant +1 - \xi_i, \quad i \in I_1, \\ &\langle w, x_i \rangle + b \leqslant -1 + \xi_i, \quad i \in I_2, \\ &\xi_i \geqslant 0, \qquad\qquad\quad i \in I_1 \cup I_2. \end{aligned} \tag{6}$$

The inequalities (6) are relaxed by positive slack variables ξ_i. Their sum is weighted by a prescribed regularization constant C added to the objective function (5). The sum $\sum_i (\xi_i)^d$ is the cost function for patterns whose distance to the hyperplane is less than margin $1/\|w\|$. The upper index ^{d} denotes the power of the cost function. The cost functions are linear (for $d = 1$) or quadratic (for $d = 2$). Let us note that the generalized hyperplane is sought in the support vector machines [2] in which the dual formulation of task (5) is used. The data x_i appears only in the form of *dot products* in the dual form. This property allows us to use the *kernel functions* for finding non-linear classifiers.

3. Towards a non-linear decision rule via a kernel function

An algorithm which finds a separating hyperplane (e.g., by solving the tasks defined above) can be extended to seek a

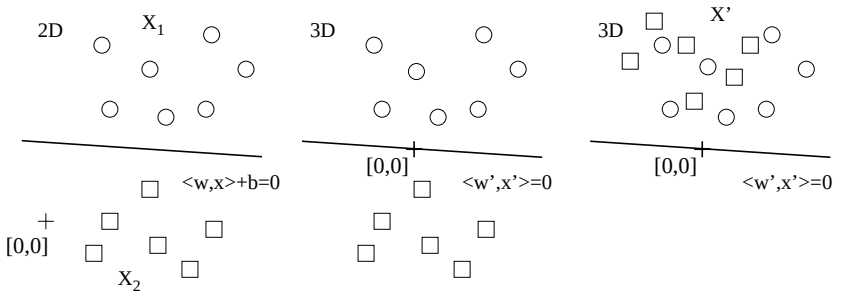

Fig. 1. A 2D example of the transformation introducing homogeneous coordinates and reflecting the set X_2 over the origin.

non-linear hypersurface defined as $f(x)=0$. This is possible if the non-linear function $f : \mathcal{X} \to \mathbb{R}$ can be expressed as a linear function in some new space $\mathcal{Y}$,

$$f(x) = \sum_{i=1}^{m} w_i \Phi_i(x) + b = \langle w, y \rangle + b. \tag{7}$$

Vectors y in the new m-dimensional space $\mathcal{Y}$ are images of the vectors x from the original n-dimensional space $\mathcal{X}$. The mapping $\Phi : \mathcal{X} \to \mathcal{Y}$ determines the non-linear hypersurface $f(x) = 0$. A linear algorithm can be employed in the new space $\mathcal{Y}$ to find the vector w and the threshold b which are the parameters of the non-linear hypersurface in the space $\mathcal{X}$.

The explicit mapping to the new m-dimensional space can be computationally intractable since the dimension m may be very high. This problem can be avoided if kernel functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ are used. The solution then becomes computationally feasible. The kernel functions correspond to the dot product of non-linearly mapped vectors $x_1, x_2 \in \mathcal{X}$,

$$k(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle.$$

Let us note that a linear algorithm has to use data in terms of dot products only to apply kernel functions. The dual form of the support vector machines is the example in which the data appear in terms of dot products and the kernel functions can be used.

4. The proposed algorithm as an extension of the Schlesinger–Kozinec algorithm

In this section we will first gradually explain the Schlesinger–Kozinec algorithm which constitutes the basis of our new extension. Second we demonstrate steps leading us to a generalization which solves the non-linear and non-separable cases.

We take into account that the Schlesinger–Kozinec algorithm is quite unknown. We will start with a simple algorithm which was originally designed to find a separating hyperplane (1). The algorithm can be simply modified to seek the optimal hyperplane (3) as well as the generalized optimal hyperplane (5) with quadratic cost function. Moreover, this algorithm can be expressed in terms of dot products which results in non-linear extensions if the kernel functions are used.

4.1. Kozinec's algorithm

The Kozinec algorithm finds the separating hyperplane $\langle w, x \rangle + b = 0$ which splits two vector sets X_1, X_2, i.e., the vector w and the scalar b are found which satisfy

$$\langle w, x_i \rangle + b > 0, \quad i \in I_1,$$

$$\langle w, x_i \rangle + b < 0, \quad i \in I_2. \tag{8}$$

Let us formally simplify this task to the equivalent problem by introducing new vectors

$$w' = [w, b], \quad X_1' = \{[x_i, 1] : i \in I_1\},$$

$$X_2' = \{[-x_i, -1] : i \in I_2\}, \tag{9}$$

where one new coordinate was added. This transformation can be geometrically seen as introducing homogeneous coordinates and reflecting the set X_2 over the origin as depicted in Fig. 1. Performing this transformation task (8) becomes equivalent to seeking the vector w' satisfying

$$\langle w', x' \rangle > 0, \quad i \in x' \in X'. \tag{10}$$

The vector w' sought determines a hyperplane $\langle w', x' \rangle = 0$ passing through the origin. The task separating finite sets X_1 and X_2 is modified to the task of putting set $X' = X_1' \cup X_2'$ on one side of the hyperplane, see Fig. 1. Further on we will use w, x and X instead of w', x' and X' to make notation simpler. The Kozinec algorithm is defined as follows.

Algorithm 1. Kozinec's algorithm

1. *Initialization*: Set w to any vector $x \in X$.
2. *Stopping condition*: Find any vector $x_t \in X$ which satisfies the inequality

   $$\langle w, x_t \rangle < 0.$$

If such a vector does not exist then the vector w solves task (10). Otherwise go to Step 3.

3. *Adaptation*: Compute the new value of vector w^{new} as

$$w^{new} = w \cdot (1 - q) + q \cdot x_t, \tag{11}$$

where

$$q = \underset{q \in (0,1]}{\operatorname{argmin}} \|w \cdot (1 - q) + q \cdot x_t\|. \tag{12}$$

Continue with Step 2.

The Kozinec algorithm begins iterations from an arbitrary point $x \in \bar{X}$, where $\bar{X}$ is the convex hull of the set X. For instance it can start from any point $x \in X$. In Step 2 a point x_t violating condition (10) is sought. If such a point is not found then w defines the separating hyperplane. Otherwise w is adapted so that its new value w^{new} is the closest point to the origin which lies on the abscissa between the old w and the vector x_t. The number q determines how much we must move from the vector w towards the vector x_t to find the new value the w^{new}. There is a simple closed form solution of task (12) and its result is the number q. The adaptation rule (12) ensures that the vector w remains in the convex hull $\bar{X}$ and its norm monotonically decreases. The mentioned iteration scheme is repeated until the stopping condition is satisfied.

Let us point out a similarity between the Kozinec algorithm and the Perceptron. Substituting the adaptation rule $w^{new} = w + x_t$ in Step 3, the Kozinec algorithm becomes the Perceptron algorithm. It was proved by Novikoff [13] that for the linearly separable case the Perceptron algorithm finds the separating hyperplane after D^2/m^2 steps at most, where number $D = \max_{x \in X} \|x\|$ and $m = \min_{x \in \bar{X}} \|x\|$. A similar theorem exists for the Kozinec algorithm [4] where the upper bound of the number of iterations is $D^2/m^2 \ln D^2/m^2$. Let us note, that this bound holds also when we return to the original formulation and seek for the hyperplane $\langle w, x \rangle + b = 0$. In this case the number $D = \max(\max_{x_1, x_2 \in X_1} \|x_1 - x_2\|, \max_{x_1, x_2 \in X_2} \|x_1 - x_2\|)$ and the $m = \min_{w_1 \in \bar{X}_1, w_2 \in \bar{X}_2} \frac{1}{2} \|w_1 - w_2\|$. The Kozinec algorithm can be slightly modified to find the optimal hyperplane (3) which is described in the following section.

4.2. Schlesinger–Kozinec algorithm

We will show the modified stopping condition in Step 2 of the Kozinec Algorithm 1 which leads to the algorithm finding the optimal hyperplane

$$w^* = \underset{w}{\operatorname{argmin}} \|w\|^2 \tag{13}$$

subject to

$$\langle w, x_i \rangle \geqslant 1, \quad i \in I,$$

where transformation (9) was used. Let the vector w^* be a vector contained in the convex hull $\bar{X}$ and being the closest to the origin, i.e.,

$$w^* = \underset{w \in \bar{X}}{\operatorname{argmin}} \|w\|. \tag{14}$$

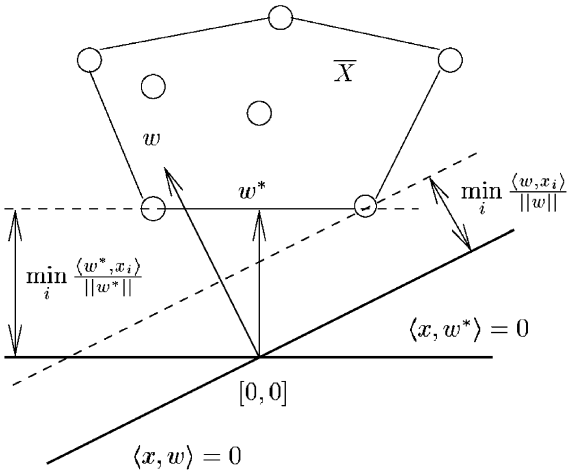

Fig. 2. The vector w^* determines the optimal hyperplane which is the solution of $w^* = \underset{w}{\operatorname{argmax}} \min_{i \in I} \langle w, x_i \rangle / \|w\|$.

Such a w^* also solves task (13) and defines the optimal hyperplane. We refer to Ref. [5] for the proof of this assertion.

As described above the Kozinec algorithm decreases the norm of the vector w in each iteration and the vector w remains in the convex hull $\bar{X}$. Disabling the stopping condition of the Kozinec algorithm would lead to the asymptotical convergence of the w to the w^* provided it does not stop in a finite number of steps. It implies that modification to the algorithm finding the optimal hyperplane requires the use of the stopping condition which ends the algorithm if the $\|w\|$ is close enough to the $\|w^*\|$.

Schlesinger [3] proposed stopping the Kozinec algorithm when the margin $m(w)$ of the current separating hyperplane is smaller than the optimal margin $m(w^*)$ by at most the prescribed ε. A hyperplane satisfying this condition is called *ε-optimal hyperplane* and is formally defined as a hyperplane $\langle w, x \rangle = 0$ satisfying

$$m(w^*) - m(w) \leqslant \varepsilon. \tag{15}$$

Let us note that for $\varepsilon = 0$ the ε-optimal hyperplane coincides with the optimal hyperplane. Condition (15) can be constructively evaluated even though the optimal w^* is not known beforehand. The idea is to use the norm $\|w\|$ as the upper bound of $m(w^*)$. This follows from the following relation:

$$m^* = \min_{i \in I_1 \cup I_2} \frac{\langle w^*, x_i \rangle}{\|w^*\|} = \|w^*\| \leqslant \|w\|.$$

The validity of this relation can be intuitively understood from Fig. 2. Using the upper bound $\|w\|$ we can assert that a hyperplane $\langle w, x \rangle = 0$ for which

$$\|w\| - \min_{i \in I} \frac{\langle w, x_i \rangle}{\|w\|} \leqslant \varepsilon \tag{16}$$

holds is the ε-optimal hyperplane satisfying (15). Substituting condition (16) (with the converse inequality sign) to Step 2 of the Kozinec Algorithm 1 we get the algorithm which finds the ε-optimal hyperplane. We arrived at the Schlesinger–Kozinec algorithm which we abbreviate as the S–K-algorithm.

Let us return to the original formulation in which we sought a hyperplane $\langle w,x\rangle + b = 0$ instead of the hyperplane $\langle w',x'\rangle = 0$ passing through the origin. As we mentioned above the hyperplane $\langle w,x\rangle + b = 0$ can be immediately obtained from the solution $\langle w',x'\rangle = 0$ of the S–K-algorithm since $w' = [w,b]$. However, experimental results show [5] that better performance can be achieved using the S–K-algorithm formulated directly to find $\langle w,x\rangle + b = 0$ instead of using equivalent expression (9) and seeking $\langle w',x'\rangle = 0$. The following lines describe the S–K-algorithm directly seeking the optimal hyperplane.

Task (14) for X' changes for X_1, X_2 to the task

$$(w_1^*, w_2^*) = \operatorname*{argmin}_{w_1\in\bar{X}_1, w_2\in\bar{X}_2} \|w_1 - w_2\|, \tag{17}$$

where $\bar{X}_1$ denotes the convex hull of the set X_1 and $\bar{X}_2$ the convex hull of the set X_2. The hyperplane perpendicular to the abscissa between vectors w_1^*, w_2^* and passing through the midpoint coincides with the optimal hyperplane (3). Formally written the optimal hyperplane $\langle w^*,x\rangle + b = 0$ is computed as

$$w^* = w_1^* - w_2^*, \quad b = \tfrac{1}{2}(\|w_2^*\|^2 - \|w_1^*\|^2). \tag{18}$$

The S–K-algorithm starts from arbitrary vectors $w_1 \in \bar{X}_1$, $w_2 \in \bar{X}_2$. The ε-optimality criterion is evaluated for the vector x_i, $i \in I_1 \cup I_2$ closest to the hyperplane. Let us assume, for instance, that the vector x_t, $t \in I_1$ was found and the ε-optimality criterion is violated. Then the vector w_2 is fixed and the vector w_1 is moved towards the vector w_2 using the Kozinec adaptation rule $w_1^{new} = w_1 \cdot (1-q) + x_t \cdot q$. The number q is determined such that the distance between the vectors w_1^{new} and w_2 is minimal. Similar adaptation is performed for the vector w_2. If x_t, $t \in I_2$ then w_1 is fixed and w_2 is moved towards w_1. The algorithm iterates until the ε-optimality criterion is satisfied. The ε-optimality condition $m(w^*) - m(w) < \varepsilon$ can be efficiently evaluated as

$$\|w_1 - w_2\| - \min\left(\min_{i\in I_1} \frac{\langle x_i - w_2, w_1 - w_2\rangle}{\|w_1 - w_2\|}, \min_{i\in I_2} \frac{\langle x_i - w_1, w_2 - w_1\rangle}{\|w_1 - w_2\|}\right) < \varepsilon. \tag{19}$$

See Fig. 5(a) for geometrical illustration.

Algorithm 2. Schlesinger–Kozinec algorithm

1. *Initialization*: Set the vector w_1 to any vector $x \in X_1$ and w_2 to any vector $x \in X_2$.
2. *Stopping condition*: Find a vector x_t closest to the hyperplane as $x_t = \operatorname{argmin}_{i\in I_1\cup I_2} m(x_i)$ where
$$m(x_i) = \begin{cases} \frac{\langle x_i - w_2, w_1 - w_2\rangle}{\|w_1 - w_2\|}, & \text{for } i \in I_1, \\ \frac{\langle x_i - w_1, w_2 - w_1\rangle}{\|w_1 - w_2\|}, & \text{for } i \in I_2. \end{cases}.$$
If the ε-optimality condition $\|w_1 - w_2\| - m(x_t) < \varepsilon$ holds then the vector $w = w_1 - w_2$ and $b = \frac{1}{2}(\|w_1\|^2 - \|w_2\|^2)$ defines the ε-solution. Otherwise go to Step 3.
3. *Adaptation*: If $x_t \in X_1$ then set $w_2^{new} = w_2$ and compute
$$w_1^{new} = w_1 \cdot (1-q) + q \cdot x_t, \quad \text{where}$$
$$q = \min\left(1, \frac{\langle w_1 - w_2, w_1 - x_t\rangle}{\|w_1 - x_t\|^2}\right).$$
Otherwise if $x_t \in X_2$ then set $w_1^{new} = w_1$ and compute
$$w_2^{new} = w_2 \cdot (1-q) + q \cdot x_t, \quad \text{where}$$
$$q = \min\left(1, \frac{\langle w_2 - w_1, w_2 - x_t\rangle}{\|w_2 - x_t\|^2}\right).$$
Continue with Step 2.

The hyperplane (18) found is the optimal one (with given precision ε) but it is not in the canonical form (3) since the training vectors closest to the hyperplane satisfy $\langle w^*,x\rangle + b = \|w^*\|/2$ unlike in the canonical representation where $\langle w_c^*,x\rangle + b_c = 1$. To obtain the canonical form, the hyperplane found must be rescaled by a proper constant. It is straightforward that

$$w_c^* = 2\frac{w^*}{\|w^*\|}, \quad b_c = 2\frac{b}{\|w^*\|}. \tag{20}$$

4.3. Kernel Schlesinger–Kozinec algorithm

We have arrived at the point where we start explaining our novel contribution. This section describes how to generalize the S–K-algorithm, see Algorithm 2, to train a non-linear discriminant functions. The underlying idea is to express the S–K-algorithm in terms of dot products and use the kernel trick which has already been mentioned in Section 3.

The iterated vectors w_1 and w_2 are explicitly expressed as the convex combination of point sets X_1 and X_2. We introduce multipliers α_i, $i \in I_1 \cup I_2$ which correspond to the training data x_i, $i \in I_1 \cup I_2$. The relation between the multipliers α_i and the vectors w_1 and w_2 is given as

$$w_1 = \sum_{i\in I_1} \alpha_i \cdot x_i, \quad w_2 = \sum_{i\in I_2} \alpha_i \cdot x_i, \quad \sum_{i\in I_1} \alpha_i = 1,$$
$$\sum_{i\in I_2} \alpha_i = 1, \quad \alpha_i \geqslant 0. \tag{21}$$

First we will show that the adaptation step of the S–K-algorithm can be performed by updating the coefficient

vectors α_1, α_2. Let us assume that the algorithm arrived at Step 3 and the coefficient q and vector $x_t \in X_1$ have already been determined. The adaptation of the vector w_1 is computed as

$$w_1^{new} = w_1 \cdot (1-q) + q \cdot x_t. \tag{22}$$

Substituting Eq. (21) to Eq. (22) we get

$$\sum_{i\in I_1} \alpha_i^{new} \cdot x_i = (1-q) \cdot \sum_{i\in I_1} \alpha_i \cdot x_i + q \cdot x_t$$

$$= \sum_{i\in I_1} [(1-q) + q \cdot \delta_{i,t}] \cdot \alpha_i \cdot x_i,$$

where we used the Kronecker delta $\delta_{i,t} = 0$ for $i \neq t$ and $\delta_{i,t} = 1$ for $i = t$. This means that the adaptation step for $\alpha_i,\ i \in I_1$ can be performed as

$$\alpha_i^{new} = \alpha_i \cdot (1-q) + q \cdot \delta_{i,t} \tag{23}$$

and similarly in the case for which $\alpha_i, i \in I_2$ are to be adapted.

For evaluation of the stopping condition in Step 2 and computation of the number q in Steps 3 and 4 it is sufficient to know the values of the dot products

$$\langle w_1, w_1\rangle = \sum_{i\in I_1}\sum_{i\in I_1} \alpha_i \cdot \alpha_j \langle x_i, x_j\rangle = A,$$

$$\langle w_2, w_2\rangle = \sum_{i\in I_2}\sum_{i\in I_2} \alpha_i \cdot \alpha_j \langle x_i, x_j\rangle = B,$$

$$\langle w_1, w_2\rangle = \sum_{i\in I_1}\sum_{i\in I_2} \alpha_i \cdot \alpha_j \langle x_i, x_j\rangle = C,$$

$$\langle w_1, x_i\rangle = \sum_{j\in I_1} \alpha_j \cdot \langle x_i, x_j\rangle = D_i,$$

$$\langle w_2, x_i\rangle = \sum_{j\in I_2} \alpha_j \cdot \langle x_i, x_j\rangle = E_i, \tag{24}$$

where we denoted the dot products for later use. The data now appears only in terms of dot products. To classify a pattern x it is sufficient to check on which side of the separating hyperplane the pattern x lies. This can be determined according to the sign of the function

$$f(x) = \langle w_1 - w_2, x\rangle + b = \sum_{i\in I, \alpha_i \neq 0} \alpha_i y_i \langle x_i, x\rangle + b.$$

After substituting the kernel functions $k(x_i, x_j)$ for all the dot products used in the algorithm its non-linear version is obtained which finds the non-linear decision function

$$f(x) = \sum_{i\in I, \alpha_i \neq 0} \alpha_i y_i k(x_i, x) + b \tag{25}$$

corresponding to the optimal hyperplane in the non-linear space $\mathcal{Y}$.

The Kernel S–K-algorithm has the same framework as the original S–K-algorithm, see Algorithm 2. The only difference is in using the new adaptation rule defined by Eq. (23). Moreover, Eqs. (24) are used to compute dot products. The memory requirements are linear since the algorithm uses only these dot products their number is $3 + 2 \times |I|$, where $|I|$ equals to the number of training patterns. It is obvious that the computation of these dot products is the bottleneck of the algorithm since it requires $O(l^2)$ evaluations of both the dot products and the kernel functions. Fortunately this problem can be avoided by caching the values of dot products and employing a particularly simple adaptation rule for their update. Later we will use the notation for A, B, C, D_i and E_i which was introduced in Eq. (24) for brevity. The update of the dot product $D_i^{new} = \sum_{i\in I_1} \alpha_i \langle x_i, x\rangle$ after the adaptation of $\alpha_i^{new} = \alpha_i(1-q) + q\delta_{i,t},\ i \in I_1$ can be expressed as

$$D_i^{new} = \sum_{i\in I_1} \alpha_i^{new} \langle x_i, x\rangle$$

$$= \sum_{i\in I_1} [\alpha_i(1-q) + q\delta_{i,t}] \langle x_i, x\rangle$$

$$= (1-q) \cdot D_i + q \cdot \langle x_t, x\rangle.$$

Similarly the remaining dot products can be updated as

$$A^{new} = C \cdot (1-q)^2 + 2 \cdot (1-q) \cdot q \cdot D_t + q^2 \cdot \langle x_t, x_t\rangle,$$

$$C^{new} = C \cdot (1-q) + q \cdot D_t.$$

The case in which w_2 or $\alpha_i,\ i \in I_2$, are adapted is treated likewise.

We name the S–K-algorithm expressed in the terms of dot products the *Kernel S–K-algorithm*. This Kernel S–K-algorithm is introduced next and we will use $k(x_i, x_j)$ instead of the dot product $\langle x_i, x_j\rangle$.

Algorithm 3. Kernel Schlesinger–Kozinec algorithm

1. *Initialization*: Set up $\alpha_{i_1} = 1$ for any $i_1 \in I_1$, $\alpha_{i_2} = 1$ for any $i_2 \in I_2$, and the remaining multipliers $\alpha_i = 0$, $i \in I_1 \cup I_2$, $i \neq i_1, i \neq i_2$. Initialize the cache of dot products corresponding to the multipliers α_i as

   $$A = k(x_{i_1}, x_{i_1}), \quad B = k(x_{i_2}, x_{i_2}), \quad C = k(x_{i_1}, x_{i_2}),$$

   $$D_i = k(x_{i_1}, x_i), \quad E_i = k(x_{i_2}, x_i), \quad i \in I_1 \cup I_2.$$

2. *Stopping condition*: Find the index of the closest point to the hyperplane as $t = \operatorname{argmin}_{i\in I_1\cup I_2} m(i)$ where

   $$m(i) = \begin{cases} \dfrac{D_i - E_i + B - C}{\sqrt{A + B - 2C}}, & \text{for } i \in I_1, \\ \dfrac{E_i - D_i + A - C}{\sqrt{A + B - 2C}}, & \text{for } i \in I_2. \end{cases}$$

   If the ε-optimality condition $\sqrt{A + B - 2C} - m(t) < \varepsilon$ holds then the multipliers $\alpha_i, i \in I_1 \cup I_2$, correspond to the ε-optimal hyperplane. Otherwise go to Step 3.
3. *Adaptation*: If $t \in I_1$ then adapt $\alpha_i,\ i \in I_1$ as

   $$\alpha_i^{new} = \alpha_i \cdot (1-q) + q \cdot \delta_{i,t}, \quad i \in I_1,$$

where

$$q = \min\left(1, \frac{A - D_t + E_t - C}{A + k(x_t,x_t) - 2\cdot(D_t - E_t)}\right)$$

and update cached dot products

$$A^{new} = A\cdot(1-q)^2 + 2\cdot(1-q)\cdot q\cdot D_t + q^2\cdot k(x_t,x_t),$$

$$C^{new} = C\cdot(1-q) + q\cdot E_t,$$

$$D_i^{new} = D_i\cdot(1-q) + q\cdot k(x_i,x_t), \quad i\in I_1\cup I_2.$$

Otherwise if $t\in I_2$ then adapt $\alpha_i, i\in I_2$ as

$$\alpha_i^{new} = \alpha_i\cdot(1-q) + q\cdot\delta_{i,t}, \quad i\in I_2,$$

where

$$q = \min\left(1, \frac{B - E_t + D_t - C}{B + k(x_t,x_t) - 2(E_t - D_t)}\right)$$

and update cached dot products

$$B^{new} = B\cdot(1-q)^2 + 2\cdot(1-q)\cdot q\cdot E_t + q^2\cdot k(x_t,x_t),$$

$$C^{new} = C\cdot(1-q) + q\cdot D_t,$$

$$E_i^{new} = E_i\cdot(1-q) + q\cdot k(x_i,x_t), \quad i\in I_1\cup I_2.$$

Continue with Step 2.

The multipliers α_i, $i\in I_1\cup I_2$ found by the Kernel S–K-algorithm determine the non-linear decision function (25). The threshold b is computed analogically to Eq. (18),

$$b = \frac{1}{2}\left(\sum_{i\in I_2}\sum_{j\in I_2}\alpha_i\cdot\alpha_j\cdot k(x_i,x_j) - \sum_{i\in I_1}\sum_{j\in I_1}\alpha_i\cdot\alpha_j\cdot k(x_i,x_j)\right) = \frac{1}{2}(B-A),$$

where the dot products updated in the algorithm were used. Similarly the canonical expression of the found hyperplane can be obtained using Eq. (20), i.e. by multiplying $\alpha_i, i\in I_1\cup I_2$ and b by the number

$$\frac{2}{\sqrt{\sum_{i\in I_1}\sum_{j\in I_2} y_i\cdot y_j\cdot\alpha_i\cdot\alpha_j\cdot k(x_i,x_j)}} = \frac{2}{\sqrt{A+B-2C}}. \tag{26}$$

Let us mention a connection between the multipliers α_i and the dual form [2] of task (3) used in the SVM algorithms to find the optimal hyperplane. Such a dual task is defined as

$$(\alpha_i, i\in I) = \operatorname*{argmax}_{\alpha_i\in I}\sum_{i\in I}\alpha_i - \frac{1}{2}\sum_{i\in I}\sum_{j\in J}\alpha_i\cdot\alpha_j\cdot y_i\cdot y_j\cdot k(x_i,x_j) \tag{27}$$

subject to

$$\sum_{i\in I}\alpha_i\cdot y_i = 0, \quad \alpha_i\geqslant 0,\ i\in I.$$

The multipliers α_i solving Eq. (27) coincide with the multipliers computed by the Kernel S–K-algorithm after normalization (26). Let us emphasize that even though the Kernel S–K-algorithm optimizes the multipliers α_i of the dual task its stopping criterion is defined in terms of the original (primal) task, i.e., as the difference between the margin of the found and the optimal hyperplane.

4.4. Non-separable case

We will show how the S–K-algorithm can be applied to the situation in which the data are not separable.

The idea is to use transformation [11] which expresses the task seeking the optimal generalized hyperplane defined for the non-separable case as the task seeking the optimal hyperplane in which the data are linearly separable.

Here the input n-dimensional space $\mathcal{X}$ is transformed to a new $n+l$-dimensional space $\mathcal{X}'$ where l is the number of training patterns $l = |I_1| + |I_2|$. The vector $x_i\in\mathcal{X}$ is mapped to a new vector $x_i'\in\mathcal{X}'$ so that

$$x_i' = [x_i,\ 0,\ \ldots,\ \tfrac{y_i}{\sqrt{2C}},\ \ldots,\ 0],$$

$$w' = [w,\ \sqrt{2C}\xi_1,\ \ldots,\ \sqrt{2C}\xi_i,\ \ldots,\ \sqrt{2C}\xi_l]. \tag{28}$$

The added coordinates of the vector w' correspond to the slack variables. If we substitute w' and x_i' to Eq. (3) then it turns into task (5) with quadratic cost function, i.e., $d=2$. Let us note that the input data in the transformed space $\mathcal{X}'$ must be linearly separable since the non-zero coordinate added to each vector x_i' distinguishes this vector from the others.

To apply the mentioned transformation to the kernel S–K-algorithm, we must only change the kernel function $k(x_i,x_j)$ to a new $k'(x_i,x_j)$ which is defined as

$$k'(x_i,x_j) = k(x_i,x_j) + \delta_{i,j}\frac{1}{2C}. \tag{29}$$

Using kernel (29) the Kernel S–K-algorithm will find the generalized optimal hyperplane with the quadratic cost function (5) in the non-linear space induced by the kernel $k(\cdot,\cdot)$.

5. Relation to other algorithms

5.1. Nearest point algorithm

A similar approach to the proposed algorithm is the nearest point algorithm (NPA) by Keerthi et al. [10]. This method combines two algorithms which, similarly to the Schlesinger–Kozinec algorithm, finds the nearest vectors from convex hulls $\bar{X}_1$ and $\bar{X}_2$. The first one is the Gilbert's algorithm [14] and the second one is algorithm by Mitchell et al. [15]. The Gilbert's algorithm iteratively solves

$$w^* = \operatorname*{argmin}_{w\in\bar{Z}}\|w\|,$$

where $\bar{Z}$ is a convex hull of the difference set $Z = \{x_i - x_j : i \in I_1, j \in I_2\}$. The vector w^* equals to the $w_1^* - w_2^*$ [10], where w_1^* and w_2^* are the nearest vectors from convex hulls $\bar{X}_1$ and $\bar{X}_2$ sought in the Schlesinger–Kozinec algorithm. The Gilbert's algorithm solves the one convex hull problem using the same adaptation rule (11) as in the Kozinec's algorithm. The difference set Z need not be constructed explicitly since in each step only a vector z_t violating optimality condition is needed. The vector $z_t = x_i - x_j$ is determined by the vectors $x_i \in X_1$ and $x_j \in X_2$ having the shortest distance to the hyperplane. In contrast, the Schlesinger–Kozinec algorithm needs only one vector for adaptation. The second algorithm, proposed by Mitchell et al., explicitly uses the representation of w as convex combination of the training vectors, i.e., $w = \sum \alpha_i y_i x_i$. This algorithm tries to find such representation of w that all the training vectors, having $\alpha_i \neq 0$ are those which minimize the distance to the found hyperplane since such representation define the optimal hyperplane [10]. The NPA algorithm combines the both mentioned algorithms plus some ideas of the SMO [7]. This results in a fast algorithm which is, however, more complicated for implementation than the original ones.

5.2. Relaxed online maximal margin algorithm

The relaxed online maximal margin algorithm (ROMMA) by Li and Long [9] is another closely related approach. This algorithm iteratively searches for the linear separating hyperplane $\langle w, x\rangle = 0$ (bias is not considered) which eventually converges to the optimal separating hyperplane. The optimal separating hyperplane in canonical form (3) which is to be found must have the minimal norm $\|w\|$ provided the linear constrains $y_i\langle w, x_i\rangle \geqslant 1$, $i \in I$ hold. The ROMMA goes through the training vectors trying to find such x_t which violates the condition $y_i\langle w, x_t\rangle \geqslant 1$. When x_t is found then a new w^{new} is computed as

$$w^{new} = \underset{w \in H}{\operatorname{argmin}} \|w\|, \tag{30}$$

where subset $H = \{w: \langle w^{new}, w\rangle \geqslant \|w^{new}\|^2\} \cap \{w: y_t\langle w, x_t\rangle \geqslant 1\}$ approximates the set of all linear constrains $y_i\langle w, x_i\rangle \geqslant 1$, $i \in I$. The optimization problem (30) has an analytical solution [9]

$$w^{new} = a \cdot w + b \cdot x_t, \tag{31}$$

where

$$a = \frac{\|x_t\|^2\|w\|^2 - y_t\langle w, x_t\rangle}{\|x_t\|^2\|w\|^2 - \langle w, x_t\rangle},$$

$$b = \frac{\|w\|^2(y_t - \langle w, x_t\rangle)}{\|x_t\|^2\|w\|^2 - \langle w, x_t\rangle}.$$

This iterative process is repeated until convergence. We implemented the ROMMA according to [9] and, moreover, we used two improvements: (i) the dot products $\langle w, w\rangle$ and $\langle w, x_i\rangle$, $i \in I$ are cached and efficiently updated due to the simple adaptation rule (similarly to the Kernel S–K-algorithm), (ii) one constant coordinate is added to the data and the corresponding coordinate of the vector w is used as the bias (see transformation (9)) which can be efficiently done using kernel function $k'(x_i, x_j) = k(x_i, x_j) + 1$.

5.3. Stopping condition

Finally, let us compare the specific ε-optimality stopping condition of the proposed Kernel S–K-algorithm to a commonly used stopping condition, e.g., in the SMO algorithm [7], SVM^{light} [8]. The common stopping condition follows from the Karush–Kuhn–Tucker optimality conditions and for the separable SVM optimization problem (3) can be written as

$$y_i(\langle w_c, x_i\rangle + b_c) = 1 \pm \bar{\varepsilon}, \quad \text{for } \alpha_i > 0,$$

$$y_i(\langle w_c, x_i\rangle + b_c) > 1 \pm \bar{\varepsilon}, \quad \text{for } \alpha_i = 0, \tag{32}$$

where the $\bar{\varepsilon}$ defines precision. We denote (32) $\bar{\varepsilon}$-optimality stopping condition. As mentioned (see Section 4.3), by scaling the hyperplane obtained from the Kernel S–K-algorithm by the factor $2/\|w\|$ we obtain its canonical representation. From the geometrical interpretation (see Fig. 5) it is clear, that we can see $\bar{\varepsilon}$-optimality condition as the ε-optimality condition applied to the canonical hyperplane. More precisely, the following stopping condition:

$$2 - 2 \cdot \min\left(\min_{i \in I_1} \frac{\langle x_i - w_2, w_1 - w_2\rangle}{\|w_1 - w_2\|^2}, \min_{i \in I_2} \frac{\langle x_i - w_1, w_2 - w_1\rangle}{\|w_1 - w_2\|^2}\right) \leqslant \bar{\varepsilon}, \tag{33}$$

implemented to the Kernel S–K-algorithm is equivalent to the $\bar{\varepsilon}$-stopping condition (32). By comparing Eq. (33) to the ε-optimality condition (19) we get relation

$$\varepsilon = \frac{\|w\|}{2} \cdot \bar{\varepsilon}.$$

6. Experiments

We conducted several experiments in order to (i) investigate the influence of the ε parameter determining the precision of the Kernel S–K-algorithm, (ii) to compare the Kernel S–K-algorithm with the current representative algorithms, and (iii) to evaluate the proposed algorithm in a practical problem of digits recognition from a video records. The results on the item (i) are presented in Section 6.1. The practical problem and comparison to SMO, SVM^{light}, ROMMA and k-NN classifier is described in Section 6.2.

6.1. Influence of the precision parameter ε

We tested influence of the precision parameter ε on both synthetic and real benchmark data sets. We selected the Ripley's data set [16] as the synthetic problem and

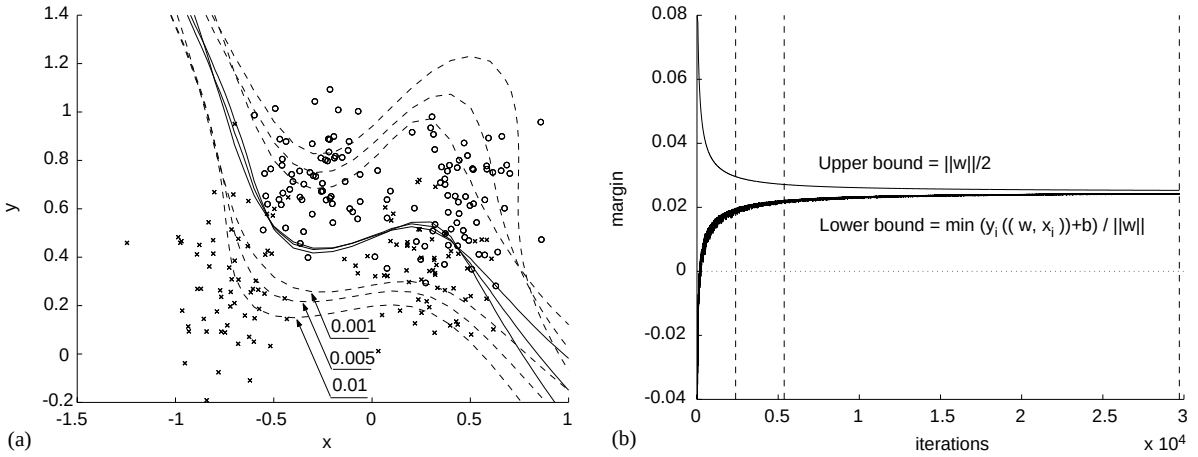

Fig. 3. The separation hypersurface for the RBF function with $\sigma = 0.5$ trained on the Ripley's data set. Three hypersurfaces depicted in (a) are computed for precision parameter $\varepsilon = [0.01, 0.005, 0.001]$. (b) Contains evolution of the upper and lower bounds on the optimal margin. The vertical dashed lines denote the number of iterations needed to compute hypersurfaces with precisions $\varepsilon = [0.01, 0.05, 0.001]$.

Table 1
Influence of the precision parameter ε of the Kernel S–K-algorithm tested on selected benchmark datasets

Precision ε	Class. error (test) (%)	Class. error (train) (%)	Margin $\pm\varepsilon/2$	Kernel evaluations	Time (s)	SVs (%)
Ripley data set, *RBF kernel with* $\sigma = 0.5$, $C = 10$						
0.01	9.4	10.8	0.024	0.640×10^6	0.18	44.4
0.005	9.4	10.8	0.024	1.396×10^6	0.32	46.0
0.001	9.4	10.8	0.024	7.536×10^6	1.72	51.2
Pima Indians Diabetes, *RBF kernel with* $\sigma = 1000$, $C = 10$						
0.01	20.8	26.3	0.092	2.313×10^6	0.64	96.4
0.005	21.6	26.3	0.086	5.111×10^6	1.41	97.7
0.001	21.1	25.8	0.082	27.188×10^6	7.52	97.9
Wisconsin Breast Cancer, *RBF kernel with* $\sigma = 10$, $C = 1$						
0.01	1.5	3.5	0.235	0.333×10^6	0.10	27.8
0.005	1.5	3.5	0.235	0.656×10^6	0.19	29.5
0.001	1.5	3.5	0.235	3.279×10^6	0.95	32.5

the Pima Indians Diabetes data set and the Wisconsin Breast Cancer data set from the UCI data repository [17] as real problems. All the data sets used correspond to binary classification problems. The Ripley's data set contains the testing and training part. The remaining two data sets were partitioned into the testing and training part in proportion 50%:50%.

We used the radial basis function (RBF) kernel $k(x_i, x_j) = e^{-\|x_i - x_j\|^2/(2\sigma^2)}$. To solve the SVM problem we have to determine the regularization constant C and the kernel width σ for the RBF kernel. For each data set we run the Kernel S–K-algorithm with $\varepsilon = 0.01$ for $C = [1, 2, 5, 10, \ldots, 10000]$ and $\sigma = [0.1, 0.2, 0.5, 1, 2, 5, 10, \ldots, 5000]$, i.e., 195 possible combinations. We selected the parameters for which the smallest classification error on the test set was obtained. The best selected parameters were used for training the Kernel S–K-algorithm with precisions $\varepsilon = [0.01, 0.005, 0.001]$.

For the Ripley's data set we visualized the calculated separation hypersurface of the RBF kernel with respect to the precision ε of the Kernel S–K-algorithm. The result is displayed in Fig. 3(a). In Fig. 3(b) we depicted the upper and lower bound (y-axis) of the optimal margin with respect to the number of iterations (x-axis). As mentioned above the algorithm stops when the difference between these bounds is less than the prescribed ε. Three vertical dashed lines denote the numbers of iterations for the $\varepsilon = [0.01, 0.005, 0.001]$.

The summary of results is enlisted in Table 1. During the experiments we measured (i) testing classification error,

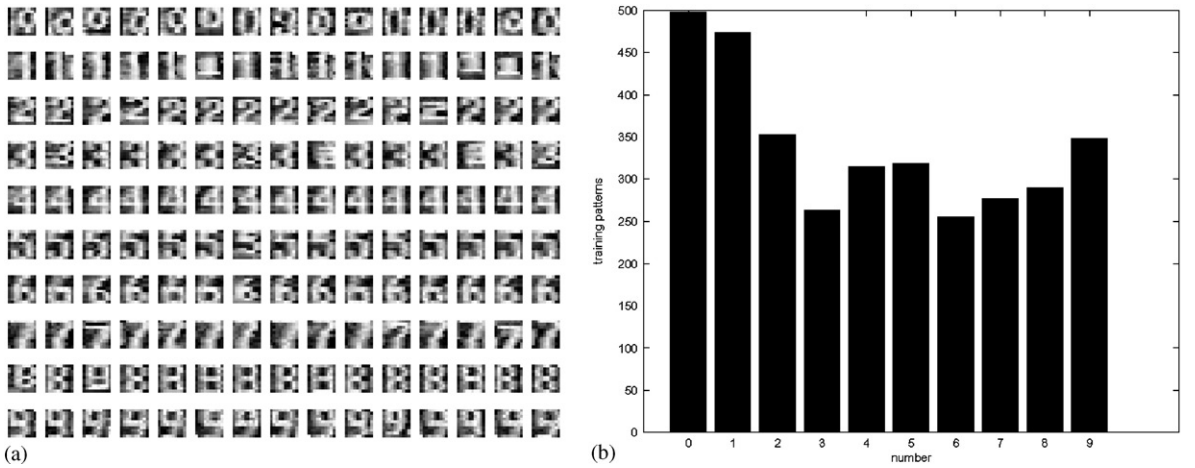

Fig. 4. The sample of gray-scale 10×10 images of digits captured from video records are depicted in (a). The distribution of digits per class is presented in (b).

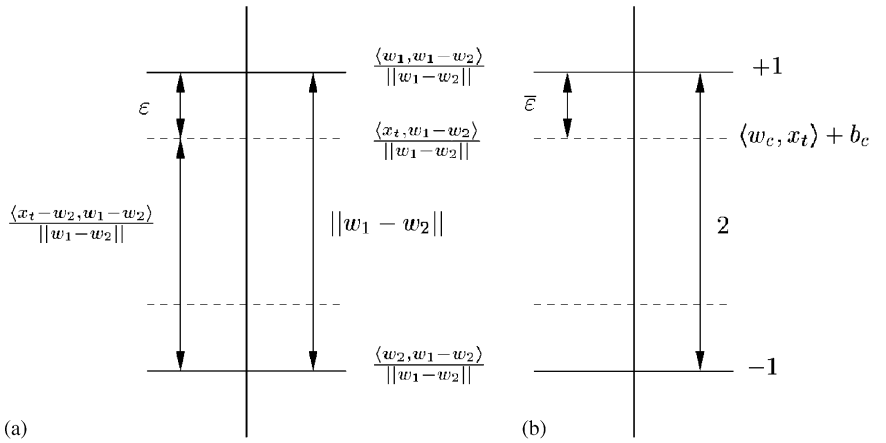

Fig. 5. Geometrical interpretation of ε-optimality (a) and $\bar{\varepsilon}$-optimality (b) stopping conditions. (a) Shows how can be ε-optimality condition evaluated from projections of vectors w_1, w_2 and x_t on the normal vector $w = w_1 - w_2$. By normalizing the projections by number $2/\|w_1 - w_2\|$ the ε-optimality changes to $\bar{\varepsilon}$-optimality stopping condition.

(ii) training classification error, (iii) margin of the found hyperplane, (iv) number of kernel evaluations, (v) training time on AMD K7/1200 MHz, and (vi) the percentage of the support vectors in the training data set. The results show that the Kernel S–K-algorithm converges very fast at first steps and slows down as the optimal solution is approached. However, almost the same classification error rates were obtained for all the parameters $\varepsilon = [0.01, 0.005, 0.001]$. It indicates that to find good classifier we do not need the extremely precise solution with $\varepsilon \to 0$.

6.2. Digits recognition from video records

The proposed algorithm was tested on the OCR problem from the real word scenario. We were motivated by our European Union project ISAAC (IST-2001-33266) which develops a tool for analyzing videosequences captured by an inspection tractor checking sewerage. One subproblem is to automatically recognize numeric labels characterizing videos. These numerals are of a poor quality.

The used data set contains 3390 gray-scale images of digits $0, 1, \ldots, 9$ of two different fonts (two types of video recorders). The sample of digits can be seen in Fig. 4(a). The distribution of individual digits in the data set is presented in Fig. 4(b). In the preprocessing step the images were (i) size-normalized to 10×10 pixels, (ii) the pixel gray-scale values from interval $[0, 255]$ were normalized to interval $[0, 1]$ and (iii) histogram equalization was used. Each image is represented as a $10 \times 10 = 100$-dimensional feature vector containing gray-scale pixel values.

Since the proposed algorithm is designed for the binary classification problem we used the one-against-one decom-

Table 2
Classification results obtained on the digit recognition problem

Algorithm	Cross-valid. error (%)	Kernel evaluations	Time (s)	SVs (%)
$\bar{\varepsilon}$=0.1				
SMO	0.74	13.6×10^6	12	29.4
SVM^{light}	0.71	1.7×10^6	21	27.8
ROMMA	0.77	25.3×10^6	37	52.3
KSK	0.74	4.7×10^6	12	22.3
$\bar{\varepsilon}$=0.05				
SMO	0.71	15.18×10^6	13	29.7
SVM^{light}	0.71	1.81×10^6	22	28.5
ROMMA	0.75	57.7×10^6	82	57.4
KSK	0.77	8.77×10^6	14	24.9
$\bar{\varepsilon}$=0.01				
SMO	0.71	37.21×10^6	31	29.7
SVM^{light}	0.76	1.96×10^6	23	28.8
ROMMA	0.74	296.57×10^6	403	62.8
KSK	0.74	46.59×10^6	90	28.3

Comparison between the sequential minimal optimizer (SMO), SVM^{light}, the relaxed online maximum margin algorithm (ROMMA) and the Kernel S–K-algorithm (KSK).

position method [18]. This method decomposes the K-class classification problem (in our case $K = 10$) to $K(K-1)/2$ binary classifiers where each one is trained on data from two classes. We trained a binary SVM classifier $f_{i,j}(x)$ for the training data from the ith and jth classes. In the classification step the function $f_{i,j}(x)$ is evaluated for all pairs (i, j). If the function $f_{i,j}(x)$ classifies pattern x to the ith class then the vote for the ith class is incremented by one and vice versa for the jth class. Finally the pattern x is classified to the class with highest amount of votes.

We trained the binary classifier using the RBF kernel with the width $\sigma = [0.2, 0.5, 1, 2, 3, 4, 5, \ldots, 10]$. Since the problem is separable for the selected kernel we used the regularization constant $C = \infty$. The precision parameter of the Kernel S–K-algorithm was set to $\varepsilon = 0.01$. We evaluated the classification errors for all values of σ by the five-fold cross-validation on the whole dataset. The best cross-validation error rate 0.74% (percentage of misclassification) was obtained for the kernel width $\sigma = 2$.

Finally, we compared the Kernel S–K-algorithm against SMO, SVM^{light} and ROMMA as representatives of SVM algorithms and the k-NN classifier as a representative of non-SVM approach. We used the same stopping condition (32) with the parameters $\bar{\varepsilon} = [0.1, 0.05, 0.01]$. The obtained results are enlisted in Table 2. We measured (i) cross-validation error rate, (ii) number of kernel evaluations, (iii) mean training time needed for building the whole classifier (45 binary classifiers) on AMD K7/1200 MHz, and (iv) the percentage of unique support vectors in the training data set. It can be seen that the cross-validation error rates are almost equivalent for different values of $\bar{\varepsilon}$ (similarly to the result of the previous experiment). The best cross-validation error for k-NN ($k = [1, 2, \ldots, 10]$) was 1.2%, i.e. worse than all SVM algorithms. The SVM^{light} needed the smallest number of kernel evaluations, however, SVM^{light} uses memory cache for kernel evaluations unlike the other tested algorithms. The SMO and Kernel S–K-algorithm were faster for lower precisions $\varepsilon = [0.1, 0.05]$ than SVM^{light} and vice versa for higher precision $\varepsilon = 0.001$. The ROMMA algorithm was the slowest compared to the others. The Kernel S–K-algorithm yielded the rule with the smallest number of support vectors. The smaller number of the support vectors results to the faster classifier which is essential in our application where the videosequences must be processed in realtime. The ROMMA and Kernel S–K-algorithm are somewhat simpler for implementation than the SMO and essentially simpler than the SVM^{light}.

7. Conclusions

We have proposed a simple algorithm for training the support vector machine classifiers with the quadratic cost function. The algorithm is derived from the Kozinec algorithm which is very similar to the Perceptron. There is a nice modification that leads to the S–K-algorithm which finds the optimal hyperplane with a given precision. The S–K-algorithm possesses two restrictive properties, i.e., the involved separating function is linear, and the training set must be separable.

We contributed by generalizing the S–K-algorithm to the non-linear case by incorporating kernel functions. The extension to the non-separable case was suggested as well.

The proposed Kernel S–K-algorithm solves the considerably more general problem than the original algorithm while preserving its simplicity. The requirement in memory storage is linear in data. This property allows the use of the proposed algorithm for large training problems. The experiments conducted demonstrate that the proposed algorithm is competitive with state-of-the art algorithms. We have selected the SMO [7], SVM^{light} [8], ROMMA [9] and k-NN classifier for detailed comparison.

The proposed algorithm was included into the public domain Statistical Pattern Recognition Toolbox implemented in Matlab. It can be downloaded including the source code from http://cmp.felk.cvut.cz.

Acknowledgements

We like to express deep thanks to M.I. Schlesinger. We built the contribution described in this paper on top of his results. He kindly introduced to us his thoughts, lead us through the rich background of Ukrainian and Russian results in pattern recognition, and gave the initial impulse to the presented research. We would also like to thank the anonymous referees for valuable comments and suggestions that helped to improve this work.

This work was supported by the Internal Grant CTU 0208313, the Grant Agency of the Czech Republic under the grant GACR 102/00/1679, by the European Union grants IST-2001-32184 ActiPret and IST-2001-33266 ISAAC.

References

[1] V.N. Vapnik, A.Ya. Chervonenkis, The Theory of Pattern Recognition, Nauka, Moscow, 1974.

[2] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, Berlin, 1995.

[3] M.I. Schlesinger, V.G. Kalmykov, A.A. Suchorukov, Sravnitelnyj analiz algoritmov sinteza linejnogo reshajushchego pravila dlja proverki slozhnych gipotez (Comparative analysis of algorithms synthesising linear decision rule for analysis of complex hypotheses), Automatika 1 (1981) 3–9 (in Russian).

[4] B.N. Kozinec, Rekurentnyj algoritm razdelenia vypuklych obolochek dvuch mnozhestv (Recurrent algorithm separating convex hulls of two sets), in: V.N. Vapnik (Ed.), Algoritmy Obuchenia Raspoznavania (Learning algorithms in pattern recognition), Sovetskoje radio, Moskva, 1973, pp. 43–50 (in Russian).

[5] M.I. Schlesinger, V. Hlaváč, Ten Lectures on Statistical and Structural Pattern Recognition, Kluwer Academic Publishers, Dordrecht, 2002.

[6] M.A. Aizerman, E.M. Braverman, L.I. Rozoner, Theoretical foundations of the potential function method in pattern recognition learning, Autom. Remote Control 25 (1964) 821–837.

[7] J.C. Platt, Fast training of support vectors machines using sequential minimal optimization, in: B. Scholkopf, C.J.C. Burges, A.J. Smola (Eds.), Advances in Kernel Methods, MIT Press, Cambridge, MA, USA, 1998.

[8] T. Joachims, Advances in Kernel methods—support vector learning, in: B. Schlkopf, C. Burges, A. Smola (Eds.), Chapter Making Large-Scale Support Vector Machine Learning Practical, MIT Press, Cambridge, MA, 1999, pp. 169–184. Code downloadable from http://svmlight.joachims.org/.

[9] Yi Li, P.M. Long, The relaxed online maximum margin algorithm, in: S.A. Solla, T.K. Leen, K.R. Muller (Eds.), Advances in Neural Information Processing Systems, Vol. 12, MIT Press, Cambridge, MA, USA, 2000, pp. 498–504.

[10] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, A fast iterative nearest point algorithm for support vector machine classifier design, IEEE Trans. Neural Networks 11 (1) (2000) 124–136.

[11] T.T. Friess, R. Harrison, Support vector neural networks: the Kernel Adatron with bias and soft-margin, Technical Report 725, University of Sheffield, UK, 1998.

[12] A. Kowalczyk, Advances in large margin classifiers, in: A.J. Smola, P.J. Bartlett, B. Schõlkopf, D. Schuurmans (Eds.), Chapter, Maximal Margin Perceptron, MIT Press, Cambridge, MA, 1999.

[13] A. Novikoff, On convergence proofs for perceptrons, in: Proceeding of the Symposium on the Mathematical Theory of Automata, Polytechnic Institute of Brooklyn Vol. 12, 1963, pp. 615–622.

[14] E.G. Gilbert, Minimizing the quadratic form on a convex set, SIAM J. Control 4 (1966) 61–79.

[15] B.F. Mitchell, V.F. Demyanov, V.N. Malozemov, Finding the point of a polyhedron closest to the origin, SIAM J. Control 12 (1974) 19–26.

[16] B.D. Ripley, Neural networks and related methods for classification (with discussion), J. R. Statist. Soc. Ser. B 56 (1994) 409–456.

[17] UCI-benchmark repository of artificial and real data sets, University of California Irvine, http://www.ics.uci.edu/∼mlearn.

[18] U. Kresel, Advances in Kernel methods–support vector learning, in: B. Schõlkopf, C. Burges, A. Smola (Eds.), Chapter Pairwise classification and support vector machines, MIT Press, Cambridge, MA, 1999, pp. 255–268.

About the Author—VOJTĚCH FRANC was born in 1976 in the Czech Republic. He received his M.Sc. degree in cybernetics from the Czech Technical University in Prague in 2000. Currently he is a Ph.D. student at the same institution under the supervision of the second author. V. Franc's research interest are mainly in statistical pattern recognition. He is an author of the Statistical Pattern Recognition Toolbox which is a public domain tool comprising many pattern recognition methods.

About the Author—VÁCLAV HLAVÁČ was born in 1956 in the Czech Republic. He is a professor of cybernetics at the Czech Technical University in Prague where he is a head of the Center for Machine Perception. He received his M.Sc. in control engineering in 1981 and a Ph.D. in cybernetics in 1987 from the same institution. His research interests are in 3D computer vision, statistical and structural pattern recognition, and also in bioinformatics. He is a co-author of the book Šonka M., Hlaváč V., Boyle R.D.: Image Processing, Analysis, and Machine Vision, 1993, second extended edition PWS Boston, 1999.