UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

SVM-LIKE ALGORITHMS AND ARCHITECTURES FOR EMBEDDED COMPUTATIONAL INTELLIGENCE

Aliaksei Kerhet, Mingqing Hu, Francesco Leonardi,
Andrea Boni and Dario Petri

March 2007

Technical Report # DIT-07-012

SVM-Like Algorithms and Architectures for Embedded Computational Intelligence

Technical Report, March 2007

Aliaksei Kerhet, Mingqing Hu, Francesco Leonardi,
Andrea Boni, Dario Petri
Department of Information and Communication Technology
University of Trento, Italy
e-mail: {kerhet, hu, francesco.leonardi, andrea.boni, petri}@dit.unitn.it *

Abstract

This paper considers implementation of computational intelligence paradigms on resource-constrained platforms, an issue of the day for the age of invisible computing and smart sensors. This necessitates the development of "light" yet efficient hardware-friendly algorithms relying on a scanty power supplies and able to operate in stand-alone manner. The scope of the presented work is twofold. Firstly, we propose a new SVM-like approximated algorithm suitable for embedded systems due to good robustness and sparsity properties. Support vectors are considered as parameters of the outer optimization problem, whereas the inner one is solved using the primal representation, which reduces computational complexity and memory usage. Along with classical Gaussian kernel, a recently proposed hardware-friendly kernel, whose calculation requires only shift and add operations, is considered. Experimental results on several well-known data sets demonstrate the validity of the proposed approach, which in many cases outperforms the original RSVM using the same number of vectors. Secondly, we implement such kind of algorithm on a resource-constrained device such as a simple 8-bit microcontroller. The case-study considered further in this work is the design of a node of a wireless video-sensor network performing people detection, and a simple resource-constrained FPSLIC-based platform from Atmel is considered as a target device.

Keywords: Sparse classifiers design, support vector machines (SVMs), power-aware design, embedded systems, image processing.

*This work was supported in part by Italian Ministry of Education, University, and Research under grant 2005-099215.

1 Introduction

Invisible Computing is a term proposed by Donald Norman for a set of task-specific computing devices which are so spread and highly optimized that we use them without perceiving their presence in the environment [1]. One area in which this invisibility of the underlying embedded system is widely present even today is that of electronic sensors, i.e. sensors that measure some physical quantity and translate it into electronic signals that can be processed and transmitted to other devices. The local processing can be done by analogue circuits or by converting analogue signals into digital values that can be operated with digital circuits. The amount of processing can vary greatly depending on the task at hand. When there is a fair amount of processing capability associated with the sensor itself, the term *smart sensor* is used to indicate that the sensor is capable to produce high-level data.

Smart sensors have been used in a variety of tasks: vibration monitoring, temperature measurement, gas sensing, image acquisition – just to cite a few examples. In some cases, smart sensor systems have to be able to operate in a stand-alone manner and have to rely on scanty power supplies, such as those scavenged from the environment (for example, by converting light or kinetic energy). An exciting example is the Smart Dust project developed at the University of California at Berkeley [2], where about one cubic millimeter

of silicon is used to house a sensor, its computational circuits, the circuits providing power from the environment, and a wireless transmission module. Applications envisioned for the Smart Dust motes include remote sensing of large geographical areas, for which literally millions of motes would be distributed over the territory to be monitored. These applications often require the implementation of advanced signal processing algorithms (such as classification tasks for signal detection) on single nodes [3], [4]. In this context, the implementation of computational intelligence paradigms on resource-constrained platforms is becoming an interesting research area. Among others, one of the most interesting approaches that can be applied is the so-called Support Vector Machines (SVMs) [5, 6].

The scope of this paper is twofold. Firstly, we propose a new SVM-based approximated algorithm suitable for embedded systems implementation. Secondly, we implement such kind of algorithm on a resource-constrained device such as a simple 8-bit microcontroller that usually equips Wireless Sensor Networks (WSNs). In this context, a recently proposed hardware-friendly kernel is used and implemented in a fixed-point coding in order to spare resources [7].

As far as the building of resource-aware learning algorithms is concerned, SVMs are considered here due to their good robustness and sparsity properties. However, in order to design low-power and low-cost intelligent sensors, a deeper study on more efficient computational intelligent algorithms is required. This is why many schemes have been proposed to approximate the symmetric positive semi-definite matrix of the constrained quadratic optimization problem (CQP), which has to be solved to define the SVM classification function [8, 9]. Recently the Reduced Support Vector Machines (RSVMs) have also been proposed as a method to design learning machines by using very few input measures [10]. The key idea of RSVM is as follows: a very small portion of the input data set is randomly selected to generate, in primal constraints, a thin rectangular kernel matrix in place of the full dense square kernel matrix. It has been shown that computational time as well as memory usage upon the learning phase are much less demanding for RSVM than those required by a standard SVM using the full kernel matrix [10].

More recently different approximation algorithms have been proposed in order to obtain sparse classifiers [11], [12]. In particular in [12] the Support Vectors (SVs) are considered as free parameters to be optimized. Thus, new *Expansion Vectors* (EVs), obtained as a result of the outer optimization problem, are considered in the classification function. There authors propose an algorithm by exploiting the properties of the dual formulation. Alternatively, here we propose the use of the primal form, reducing computational complexity and memory usage of the optimization problem. Experimental results on several well-known data sets demonstrate the validity of our approach, as in many cases this new formulation outperforms the original RSVM using the same number of vectors. Of course, the cost is an increased complexity of the learning phase, which is usually executed off-line, and therefore does not influence run-time system performance.

Finally, the design of a node in Wireless Video-Sensor Networks (WVSNs) for people detection on a simple resource-constrained FPSLIC-based platform from Atmel [13] is considered as the case-study in this work. Several implementation issues strictly related to the use of a fixed-point math on such a platform are discussed.

The paper is organized as follows. Section 2.1 provides a brief formulation of SVM. RSVM is described in Section 2.2, while Sections 2.3 and 2.4 describe in detail the new RSVM formulation and the hardware-friendly kernel respectively. The issues related to implementation on a microcontroller-based platform are considered in Section 3. Section 4 presents the validation of the proposed method on standard data sets. The case-study considered in this work with the corresponding experimental results is presented in section 5. Finally, Section 6 reports the conclusions.

2 SVM-like Algorithms for Embedded Systems

SVMs are new classification techniques, which in the last few years have attracted the attention of machine learning community. They are derived from Statistical Learning Theory and based on the idea of Structural Risk Minimization principle [14]. SVMs have been shown to provide higher performance than traditional learning machines in various applications, and thus have been proved to be a powerful tool for solving classification problems. Input patterns are firstly mapped onto a high-dimensional feature space where the separating hyperplane that maximizes the margin between two classes is found. Using Lagrange multipliers theory, finding the optimal hyperplane can be reduced to solving CQP. Furthermore, such a hyperplane is found without requiring explicit knowledge of the mapping thanks to the use of functions that implicitly realize dot-product in the feature space. Such functions are denoted as *kernels*. The solution corresponding to the optimal hyperplane is written as a combination of a few input patterns called SVs.

2.1 The Standard Support Vector Machine

Here the formulation of standard SVM classifier is outlined. A training set of labeled examples $Z = \{(\boldsymbol{x}_i, y_i),\ i = 1, \ldots, l\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ is given. This is a typical classification problem. The pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, l$ are generated according to some unknown probability density function $p(\boldsymbol{x}, y)$. The standard SVM with quadratic cost function is formulated in the primal as follows [6]:

$$\begin{array}{c} \min\limits_{\boldsymbol{\omega_c}, b, \boldsymbol{\xi}} \dfrac{1}{2}{\boldsymbol{\omega_c}}^T \boldsymbol{\omega_c} + C \sum\limits_{i=1}^{l} \xi_i^2 \\ \text{subject to} \quad y_i({\boldsymbol{\omega_c}}^T \boldsymbol{\varphi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ i = 1, \ldots, l \end{array} \tag{1}$$

in order to define the classification function

$$f_c(\boldsymbol{x}) = {\boldsymbol{\omega_c}}^T \boldsymbol{\varphi}(\boldsymbol{x}_i) + b\,. \tag{2}$$

Function $\boldsymbol{\varphi}$ maps each input onto a new high-dimensional feature space where a hyperplane with normal vector $\boldsymbol{\omega_c}$ is found by solving the above problem. This formulation provides a trade-off between maximization of the margin and minimization of the errors on the training set [6]. Standard methods solve (1) through its dual obtained by defining a Lagrangian function. Mapping $\boldsymbol{\varphi}$ is performed implicitly, using a positive-definite kernel $K_c(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{\varphi}(\boldsymbol{u}) \cdot \boldsymbol{\varphi}(\boldsymbol{v})$, such as the polynomial or the Gaussian one:

$$\begin{array}{l} K_c\,(\boldsymbol{u}, \boldsymbol{v}) = (1 + \boldsymbol{u} \cdot \boldsymbol{v})^p \\ K_c\,(\boldsymbol{u}, \boldsymbol{v}) = e^{-\gamma \|\boldsymbol{u} - \boldsymbol{v}\|^2}\,. \end{array} \tag{3}$$

As a consequence, vector $\boldsymbol{\omega_c}$ is represented as a combination of the mapped input patterns, and the classification function (2) can be written as a combination of kernels:

$$\boldsymbol{\omega_c} = \sum_{i=1}^{N_{sv}} \alpha_i y_i \boldsymbol{\varphi}(\boldsymbol{x}_i^{sv}) \tag{4}$$

$$f_c(\boldsymbol{x}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i K_c(\boldsymbol{x}, \boldsymbol{x}_i^{sv}) + b \tag{5}$$

where each α_i is a Lagrange multiplier found after solving the dual form, and N_{sv} is the number of SVs $\boldsymbol{x}_i^{sv}$, which are the only input patterns that are important for performing the classification.

Instead of using this formulation we prefer to follow [10], and add a term $b^2/2$ to the objective function (1), which in this case can be rewritten as follows:

$$\begin{array}{c} \min\limits_{\boldsymbol{\omega}, b, \boldsymbol{\xi}} \dfrac{1}{2}\boldsymbol{\omega}^T \boldsymbol{\omega} + C \sum\limits_{i=1}^{l} \xi_i^2 \\ \text{subject to} \quad y_i(\boldsymbol{\omega}^T \boldsymbol{\phi}(\boldsymbol{x}_i)) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ i = 1, \ldots, l \end{array} \tag{6}$$

where $\boldsymbol{\omega} = ({\boldsymbol{\omega_c}}^T, b)^T$, $\boldsymbol{\phi}(\boldsymbol{x}) = (\boldsymbol{\varphi}(\boldsymbol{x})^T, 1)^T$, $\boldsymbol{\phi}(\boldsymbol{u}) \cdot \boldsymbol{\phi}(\boldsymbol{v}) = K_c(\boldsymbol{u}, \boldsymbol{v}) + 1 = K(\boldsymbol{u}, \boldsymbol{v})$, and formulas (2, 4, 5) are transformed to

$$f(\boldsymbol{x}) = \boldsymbol{\omega}^T \boldsymbol{\phi}(\boldsymbol{x}_i)\,, \tag{7}$$

$$\boldsymbol{\omega} = \sum_{i=1}^{N_{sv}} \alpha_i y_i \boldsymbol{\phi}(\boldsymbol{x}_i^{sv})\,, \tag{8}$$

$$f(\boldsymbol{x}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i K(\boldsymbol{x}, \boldsymbol{x}_i^{sv})\,. \tag{9}$$

Thus, the difference between (1) and (6) are that two different kernels are used. As indicated in [15], both problems have similar classification performance.

2.2 The Reduced Support Vector Machine

In this section RSVM is briefly described; more details can be found in [10, 15]. The main idea consists in a random selection of N_{ev} patterns from Z which are considered as EVs in the classification function. Obviously RSVM can only achieve a suboptimal solution as compared to the standard SVM (6). We denote the chosen EVs as $\{\boldsymbol{x}_i^{ev},\ i = 1, \ldots, N_{ev}\}$. To be remarked is the fact that for RSVM the structure of $\boldsymbol{\omega}_{ev}$ is defined a priori upon a random selection of N_{ev} patterns, and according to Representer Theorem [6] it can be expressed as follows:

$$\boldsymbol{\omega}_{ev} = \sum_{i=1}^{N_{ev}} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i^{ev}) \tag{10}$$

which transforms (9) to

$$f(\boldsymbol{x}) = \sum_{i=1}^{N_{ev}} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i^{ev})\,. \tag{11}$$

Let us note that, in contrast to (7–9), here α_i are free parameters with the label information already incorporated.

The choice of N_{ev} is almost empirical [15, 10] and, of course, it depends on the available resources. For example, in case of microcontroller implementation it is upper bounded by the memory constrains.

By substituting (10) into (6) the following optimization problem is obtained:

$$\begin{aligned} &\min_{\boldsymbol{\alpha},\xi_i} \frac{1}{2}\boldsymbol{\alpha}^T Q^{ev}\boldsymbol{\alpha} + C\sum_{i=1}^{l}\xi_i^2 \\ \text{subject to}\quad &y_i\Big(\sum_{j=1}^{N_{ev}} \alpha_j K(\boldsymbol{x}_j^{ev}, \boldsymbol{x}_i)\Big) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ &\qquad i = 1, \ldots, l \end{aligned} \tag{12}$$

where $Q_{ij}^{ev} = K(\boldsymbol{x}_i^{ev}, \boldsymbol{x}_j^{ev})$, $i, j = 1, \ldots, N_{ev}$. Usually the size N_{ev} of the chosen subset is much less than the dimension of the training set l, so normally EVs are widely spaced. Since the kernel functions considered in this work decrease exponentially with the rise of the distance between patterns, the optimization problem (12) can be simplified by letting $Q^{ev} = I$ [15]. In the following sections we denote problem (12) with $Q_{ij}^{ev} = K(\boldsymbol{x}_i^{ev}, \boldsymbol{x}_j^{ev})$ as RSVM–Original (RSVMO), and with $Q^{ev} = I$ as RSVM–Simplified (RSVMS). It should be noted that the computational complexity of two optimization problems are not similar because simplifying Q^z to I completely changes the quadratic cost function.

2.3 A New Formulation of RSVM

In this section we provide a new formulation of RSVM. In order to achieve lower values of the objective function, we consider EVs as optimization variables. Such an idea has been proposed for the first time in [16] and further investigated in [12]. In that works authors use the dual form to solve the optimization problem and show that it is equivalent to designing a SVM with a modified kernel function. The feature space induced by such a kernel can be considered as a discriminating subspace, which is spanned by a small number of vectors, where the data can be linearly separated. Therefore, using the dual form involves optimization over all non-zero dual variables (let N_{dv} denote their number), which are then used to calculate N_{ev} expansion coefficients for (10). Let us notice that normally $N_{ev} \ll N_{dv}$.

Here we solve the primal form instead of the dual. The primal formulation does not require such an intermediate SVM design, and the optimization is performed over N_{ev} expansion coefficients directly. Of course our procedure somehow follows the idea presented in [16], but solving the primal leads to several improvements, first of all in terms of computational complexity and reduced memory usage, thus allowing the application of large scale RSVM training algorithms. It should be noticed that the idea of considering the primal formulation in the optimization phase has also been recently discussed in [17] for the SVM approach, where it is clearly indicated that the primal can be considered to design new families of

algorithms for large scale or sparse SVM training, as the one described in the following.

We rewrite problem (12) by introducing a new optimization variable $\boldsymbol{z} = (\boldsymbol{z}_1^T, \ldots, \boldsymbol{z}_{N_{ev}}^T)^T$, $\boldsymbol{z}_i \in \mathbb{R}^d$, $i = 1, \ldots, N_{ev}$ which at convergence will represent the set of EVs $\{\boldsymbol{x}_i^{ev},\ i = 1, \ldots, N_{ev}\}$:

$$\begin{aligned} &\min_{\boldsymbol{z},\boldsymbol{\alpha},\boldsymbol{\xi}} \frac{1}{2}\boldsymbol{\alpha}^T Q^z \boldsymbol{\alpha} + C\sum_{i=1}^{l}\xi_i^2, \\ &\text{subject to} \quad y_i(\sum_{j=1}^{N_{ev}}\alpha_j K(\boldsymbol{x}_i, \boldsymbol{z}_j)) \geq 1 - \xi_i,\ \xi_i \geq 0, \\ &\qquad\qquad i = 1, \ldots, l \end{aligned} \tag{13}$$

where $Q_{ij}^z = K(\boldsymbol{z}_i, \boldsymbol{z}_j)$, $i, j = 1, \ldots, N_{ev}$, in the original formulation of the problem and $Q^z = I$ in the simplified one. We call the two algorithms Expanded RSVMO – ERSVMO and Expanded RSVMS – ERSVMS, respectively.

Let us define $t_+ = max(t, 0)$ and apply the property that whenever $\xi_i > 0$, the i-th constraint in problem (13) must be active, then we obtain $\xi_i = [1 - y_i \sum_{j=1}^{N_{ev}} \alpha_j K(\boldsymbol{z}_j, \boldsymbol{x}_i)]_+$. The constrained problem (13) can be transformed into the following unconstrained one:

$$\min_{\boldsymbol{z},\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q^z \boldsymbol{\alpha} + C\sum_{i=1}^{l}[1 - y_i\sum_{j=1}^{N_{ev}}\alpha_j K(\boldsymbol{z}_j, \boldsymbol{x}_i)]_+^2 . \tag{14}$$

We can tackle problem (14) by separating $\boldsymbol{z}$ and $\boldsymbol{\alpha}$. Thus for any fixed $\boldsymbol{z}$ (14) becomes the traditional RSVM problem. Let $G(\boldsymbol{\alpha} \mid \boldsymbol{z})$ denote the objective function of problem (14) and let function $W(\boldsymbol{z})$ be defined as follows:

$$W(\boldsymbol{z}) = \min_{\boldsymbol{\alpha}} G(\boldsymbol{\alpha} \mid \boldsymbol{z}) . \tag{15}$$

Then problem (14) can be solved in the following way:

$$\min_{\boldsymbol{z}} W(\boldsymbol{z}) = \min_{\boldsymbol{z}} \min_{\boldsymbol{\alpha}} G(\boldsymbol{\alpha}|\boldsymbol{z}) . \tag{16}$$

Here we use a gradient-descent based method to solve $\min_{\boldsymbol{z}} W(\boldsymbol{z})$:

$$\boldsymbol{z}_i^{k+1} = \boldsymbol{z}_i^k - \eta\frac{\partial W\left(\boldsymbol{z}^k\right)}{\partial \boldsymbol{z}_i} \tag{17}$$

where $\eta > 0$ controls the step size. Since $W(\boldsymbol{z})$ can be non-convex with respect to $\boldsymbol{z}_i$, local minimum points can not be avoided. Different $\boldsymbol{z}$ deliver different solutions $\tilde{\boldsymbol{\alpha}}$ for $W(\boldsymbol{z})$, so the solution $\tilde{\boldsymbol{\alpha}}$ is a function of $\boldsymbol{z}$. The value of $W(\boldsymbol{z})$ can be computed directly by solving a standard RSVM problem (15). The training algorithm of this new RSVM formulation is summarized in Table 1. In practice it is composed of two different levels of optimization. The inner one consists in solving a standard RSVM problem in primal, while the outer one finds, after some iterations, new vectors considered as EVs. The details regarding gradient calculation in (17) can be found in the Appendix.

Table 1: The Proposed Training Algorithm, Step-by-Step Description

Step	Description
1.	`randomly choose` N_{ev} `patterns from` Z `as initial EVs and solve problem (15);`
2.	`obtain the solution` $\tilde{\alpha}$ `of (15) and compute the gradient using eq. (25);`
3.	`update the EVs by using eq. (17)`
4.	`return to step (2)until the minimum objective value` $W(\boldsymbol{z})$ `is achieved`

Table 2: Computational Complexity Comparison for Primal and Dual Formulations. l Denotes the Number of Training Samples. N_{dv} Denotes the Number of Non-Zero Dual Variables. Normally $N_{ev} \ll N_{dv}$

Type	Gradient calculation	Inner optimization
Primal	$\mathcal{O}(lN_{ev})$	$\mathcal{O}(lN_{ev} + N_{ev}^3)$
Dual	$\mathcal{O}(l^2N_{ev}^3)$	$\mathcal{O}(lN_{dv} + N_{dv}^3)$

Table 2 provides the results of computational complexity comparative analysis for primal and dual formulations (obeying the analysis introduced in [17]).

2.4 Hardware-Friendly Kernel

Recently a new CORDIC-like algorithm for computing the feed-forward phase of SVM in fixed-point arithmetic has been proposed [7]. The main characteristic of such an algorithm consists in using only shift and add operations, thus avoiding resource-consuming multiplications. The core of the algorithm is the use of a new hardware-friendly kernel suitable for implementation on embedded systems. The proposed kernel is:

$$K_c\left(\boldsymbol{u}, \boldsymbol{v}\right) = 2^{-\gamma\|\boldsymbol{u}-\boldsymbol{v}\|_1} \tag{18}$$

where the hyper-parameter γ is an integer power of two. As indicated in [7], the main differences with respect to the Gaussian kernel are: the use of the L_1 norm, as in the Laplacian kernel [6], and the base change from e to 2. The norm change allows one to avoid the multiplications in computing the exponent of (18), while the base change allows to easily extend the convergence range of the CORDIC algorithm. As reported in [7], the use of this kernel does not affect the accuracy performance of SVM.

3 Microcontroller Fixed-Point Implementation

The classification algorithms described in the previous section have been implemented on a simple 8-bit microcontroller (AVR present inside all FPSLIC family devices from Atmel [13]). FPSLIC is a small programmable system-on-chip that integrates on the same device FPGA and AVR-based core. An interesting feature of this product is that it is designed for low-power applications, such as portable or handled devices, providing several low-power operating modes. Low-power often means small amount of available resources, which in turn implies that an accurate optimization of the code is required. Indeed, in FPSLIC there are 4-16 KB of Data RAM and 16-32 KB (depending on the configuration) of Program RAM, while the external memory can not be directly accessed. Therefore both program and data have to meet this strong limitation. The Data RAM is both accessible from FPGA side and AVR data memory bus, thus it can be used as an interface between the programmable logic and the microcontroller.

The macro blocks of the presented architecture are depicted in Fig. 1. FPGA controls the sensor and extracts the feature vector to be stored in a proper reserved space of the SRAM data memory. The estimation function of SVM is divided in two main blocks: the first one is used to load all the SVM parameters in the memory. This process is executed at the very beginning (power on) or whenever one wants to dynamically reconfigure

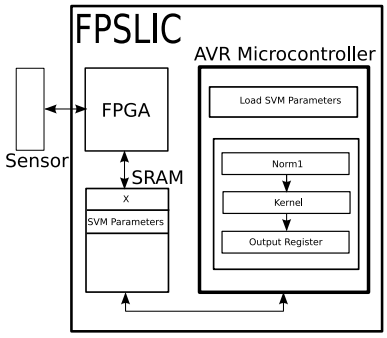

Figure 1: General architecture of microcontroller implementation.

the device with a new set of machine parameters (e.g. to detect some other kind of object).

The second function starts each time a new vector has to be classified. It reads the previously generated feature vector and provides a classification. For the sake of simplicity this function has been divided into three subparts: $norm1$, $kernel$, and $output\ register$. In $norm1$ L_1 norm is computed:

$$norm1_i = \sum_{j=1}^{d} \left| x_{ij} - x_{ij}^{ev} \right|$$

As regards the $kernel$ unit, the following expression is computed through a CORDIC-based algorithm:

$$kernel_i = \alpha_i 2^{-\gamma \cdot norm1_i}$$

In this way, as suggested in [7], the convergence of the algorithm is guaranteed.

The last module adds or subtracts all $kernel_i$ results and the bias according to the related label y_i.

The code has been written in C and then compiled using avr-gcc [18]. The amount of memory required for instructions is 622 bytes. As regards the memory used for data, its amount (in bytes) depends on number of EVs (N_{ev}) and on number of features (d) as follows:

$$M_{data} = d + d \cdot N_{ev} + N_{ev} + 5 \qquad (19)$$

where the first term corresponds to the vector to be classified, the second term is the memory

Table 3: References and Characteristics of the Data Sets Used for Benchmarks. For Each Problem. #training and #test Denote the Number of Training and Test Samples for Each of 10 Training-Test Set Pairs

Label	Problem	#training	#test	#features	Reference
1	heart	170	100	13	[20]
2	australian	414	276	14	[21]
3	diabets	468	300	8	[20]
4	breastcancer	200	77	9	[20]
5	banana	400	4900	2	[22]
6	thyroid	140	75	5	[20]
7	ringnorm	400	7000	20	[20]
8	acoustic	815	1222	50	[3]

needed to store EVs, then N_{ev} bytes are used to store all the α_i, and the other bytes are used for bias, γ, and temporary variables such as iterators.

The following approximate equation, which correlates the number of clock cycles to perform a classification with d and N_{ev}, has been empirically obtained by averaging over different trials:

$$N_{clk} = c_1 + c_2 \cdot N_{ev} + c_3 \cdot d \cdot N_{ev} \qquad (20)$$

where on the average $c_1 = 120$, $c_2 = 880$, $c_3 = 88$. Here the first term represents the time needed for function call/return and variables initialization. The second term designates the clock cycles used upon running *kernel* and *output register* blocks. Finally the third term concerns the time spent for computing $norm1$.

4 Validation on Standard Data Sets

In order to solve optimization problems previously described, the software package provided by Lin in [15] has been modified. Moreover, LBFGS algorithm [19] has been used in combination with our new RSVM implementation. Extensive experiments have been made on 8 common used data sets whose main characteristics are summarized in Table 3. They represent binary classification problems and have been used in order to compare a standard SVM algorithm with respect to the proposed versions of RSVM using both Gaussian and hardware-friendly kernels.

A random partition of data into training and test set has been used, as detailed in Table 3. Actually 10 different pairs of training and test sets have been randomly generated for each problem. In order to find the best hyper-parameters, that is C in (6, 12, 14) and γ in (3, 18), a 5-fold cross validation on a fixed range of the hyper-parameters has been performed, as suggested in [22]. For each pair a classifier has been trained on a training set realization, and its accuracy has been computed on the corresponding test set. The average value of these 10 accuracies and the average of the number of SVs (only for SVM because for RSVM-based algorithms this value is fixed a priori) have been calculated and reported in tables as described below.

Tables 4 and 5 present the accuracy of both Gaussian and hardware-friendly floating-point implementations. Here the SVM is considered as a reference useful for evaluating the accuracy of the proposed approaches. These results prove the accuracy of our approaches since ERSVM mostly outperforms classical RSVM, and the obtained classification error is close to the one obtained with classic SVM. Moreover, it can be seen that the hardware-friendly kernel does not significantly reduce the classification performance with respect to the Gaussian kernel. It should be noticed that the results obtained with ERSVMS are similar to that obtained with ERSVMO. This means that in most cases simplifying Q^z as I does not influence the classification in a relevant way, while it is obviously useful from a memory requirement and a computational point of view.

As a final remark, Table 6 demonstrates that the accuracies of the proposed approaches are comparable to that of standard SVM, while the number of EVs is much lower than the number of SVs.

Before implementing the proposed algorithms on FPSLIC-based platform, their behavior has been studied when a fixed-point coding were used. Table 7 summarizes the obtained results for the hardware-friendly kernel, which has been actually used for the final implementation. 16-bit coding has been used, where 8 bits represented the fractional part. As one can see, the results are com-

Table 4: Accuracy and Standard Deviation (in Brackets) for Benchmark Data Sets, Gaussian Kernel. l Denotes the Number of Training Samples

Problem		1	2	3	4	5	6	7	8
SVM	*accuracy*(%)	81.8 (2.6)	86.1 (1.7)	77.4 (1.4)	71.3 (4.7)	88.4 (0.6)	94.8 (2.8)	98.5 (0.1)	92.9 (0.6)
$N_{ev}/l = 2\%$	RSVMS	75.4 (5.6)	85.4 (1.9)	76.2 (2.3)	69.4 (4.0)	73.6 (7.2)	76.9 (10.5)	96.3 (1.1)	87.8 (0.7)
	ERSVMS	81.8 (2.2)	86.5 (1.5)	75.8 (1.6)	70.9 (3.9)	88.6 (0.7)	92.3 (3.2)	98.3 (0.1)	92.9 (0.5)
	RSVMO	75.3 (5.2)	85.2 (2.1)	76.4 (2.3)	69.1 (4.2)	73.0 (7.0)	76.9 (10.5)	96.3 (1.1)	88.0 (0.9)
	ERSVMO	81.7 (3.3)	86.5 (1.5)	77.2 (1.7)	71.7 (3.9)	83.3 (3.1)	92.1 (3.2)	98.2 (0.1)	91.7 (0.6)
$N_{ev}/l = 4\%$	RSVMS	77.7 (4.0)	86.3 (1.7)	77.0 (1.7)	70.3 (4.9)	86.6 (1.9)	88.3 (7.5)	97.8 (0.3)	89.8 (0.6)
	ERSVMS	81.0 (2.7)	86.4 (1.4)	76.2 (1.8)	71.6 (3.6)	89.1 (0.7)	94.7 (1.0)	98.2 (0.2)	93.2 (0.7)
	RSVMO	77.1 (4.8)	86.7 (1.7)	76.6 (2.0)	69.9 (4.1)	88.1 (1.2)	88.4 (7.4)	97.8 (0.3)	90.1 (0.6)
	ERSVMO	80.4 (2.9)	86.4 (1.7)	75.9 (1.8)	71.8 (3.9)	88.5 (0.8)	94.8 (2.4)	98.3 (0.1)	92.9 (0.6)

Table 5: Accuracy and Standard Deviation (in Brackets) for Benchmark Data Sets, Hardware-Friendly Kernel. l Denotes the Number of Training Samples

Problem		1	2	3	4	5	6	7	8
SVM	*accuracy*(%)	81.6 (2.9)	86.5 (1.9)	75.8 (1.2)	74.4 (3.0)	88.8 (0.5)	96.5 (1.2)	98.4 (0.1)	93.1 (0.6)
$N_{ev}/l = 2\%$	RSVMS	77.5 (6.2)	85.7 (2.0)	75.3 (2.3)	69.4 (4.5)	70.5 (6.8)	77.6 (10.3)	95.4 (0.9)	88.0 (1.1)
	ERSVMS	82.2 (2.4)	86.5 (1.3)	75.9 (1.9)	72.2 (3.7)	85.8 (2.4)	91.9 (3.3)	98.0 (0.2)	91.7 (0.7)
	RSVMO	77.5 (6.3)	85.8 (1.9)	75.4 (2.4)	70.0 (4.7)	70.5 (6.8)	77.6 (10.3)	95.4 (0.9)	87.9 (1.0)
	ERSVMO	81.3 (2.9)	86.4 (1.0)	76.1 (1.4)	72.9 (4.9)	85.6 (2.5)	92.7 (3.2)	98.0 (0.2)	91.5 (0.7)
$N_{ev}/l = 4\%$	RSVMS	80.7 (5.1)	86.1 (1.7)	76.2 (1.7)	71.7 (4.3)	81.4 (3.5)	84.4 (8.7)	97.2 (0.4)	90.0 (0.7)
	ERSVMS	81.4 (3.3)	86.7 (1.4)	76.0 (1.6)	74.8 (4.4)	87.4 (1.8)	93.9 (3.2)	98.1 (0.2)	92.2 (0.7)
	RSVMO	80.7 (4.9)	86.7 (1.9)	76.4 (1.6)	72.5 (4.5)	81.4 (3.6)	84.1 (8.7)	97.2 (0.4)	90.0 (0.7)
	ERSVMO	81.8 (3.6)	86.5 (1.9)	76.1 (1.7)	73.6 (5.2)	87.5 (1.4)	93.9 (2.3)	98.1 (0.2)	91.4 (0.6)

Table 6: Number of SVs and EVs for Benchmark Data Sets. l Denotes the Number of Training Samples. $\bar{N}_{sv}$ Is the Average of 10 Training-Test Set Pairs

Problem		1	2	3	4	5	6	7	8
SVM, Gaussian kernel	$\bar{N}_{sv}$	116.3	145.4	261.2	113.9	115.5	58.9	228.7	212.3
SVM, hardware-friendly kernel	$\bar{N}_{sv}$	84.6	349.9	306.8	134.9	295.8	101.3	297.0	310.4
ERSVM ($N_{ev}/l = 2\%$)	N_{ev}	3	8	9	4	8	2	8	16
ERSVM ($N_{ev}/l = 4\%$)	N_{ev}	6	16	18	8	16	5	16	32

Table 7: Fixed-Point Accuracy and Standard Deviation (in Brackets) for Benchmark Data Sets, Hardware-Friendly Kernel. Data Width Is 16 Bit, Fractional Part Width Is 8 Bit. l Denotes the Number of Training Samples

Problem		1	2	3	4	5	6	7	8
$N_{ev}/l = 4\%$	ERSVMS	80.7 (3.7)	87.3 (0.9)	76.3 (1.7)	74.3 (4.1)	86.9 (1.6)	94.0 (2.9)	97.1 (3.1)	84.0 (1.9)
	ERSVMO	81.8 (3.2)	87.6 (0.7)	72.0 (4.3)	73.8 (4.7)	75.1 (5.6)	93.3 (2.9)	97.2 (3.3)	84.7 (4.8)

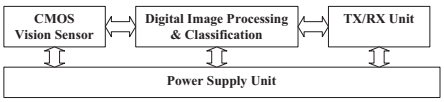

Figure 2: General architecture of a video-sensor node.

parable with those provided by a floating-point representation, thus indicating that the proposed method and architecture can be effectively implemented on simple microcontrollers.

As previously specified, the feed-forward part of the proposed method consists of several RSVM iterative solutions. At each step new EVs are computed in order to improve the objective function (13). From our experiments we found out that on the average 10 to 50 iterations are required.

5 Case-Study: People Detection on WVSNs

This section regards the specific case-study considered in this paper: people detection using WVSNs. People detection is a challenging issue emerging upon facing such problems as video surveillance, pedestrian detection, etc. [23, 24]. The present framework (limited computational and power resources, runtime operation) leads to the necessity of properly posing the problem in order to be able to use specific hardware-friendly techniques as RSVM and ERSVM.

Each node executes a local classification and sends back the result to a base station, which collects data from all the nodes in the network providing the supervision of the whole distributed system.

As this article primarily regards classification algorithms, the complete description of WVSN functioning and nodes interaction goes beyond its scope. Below in this section a single video node is considered, whose position is fixed. The addressed problem is: having the image of a fixed size $i_x \times i_y$, with the palette of 256 shades of gray (such an image is referred to as *sample*), detect whether it is a well-scaled and centered image of person.

5.1 Node Architecture

In general, a node in WVSNs, has a structure of a feed-forward chain made of four main blocks (see Fig. 2). The first element is a CMOS vision sensor, which includes a set of sensing devices, analog electronic modules, and A/D converters. The output of this block is a digitized signal, which is the input of a processing unit. Usually this unit can be composed of a 8-bit microcontroller connected to RAM and FLASH memory. A simple low-power Field Programmable Gate Array (FPGA) can be present for co-processing as well. A transceiver is responsible for transmitting/receiving information to/from other nodes. In this paper we focus on the processing unit. Such a unit performs different tasks such as supervision and coordination of the whole system on the node, preprocessing of the acquired image, and the final classification. In this context we have to face two main problems: firstly, suitable classification methods must be designed in order to fit the available resources (this step has been described in Section 2). Secondly, the designed classification methods must be implemented on the available processing unit, usually characterized by low capacity memory, lack of floating point units and multipliers, etc. (this step has been faced in Section 3).

5.2 Image Preprocessing

In order to perform classification, each sample is converted into corresponding feature vector. On the one hand, the importance of this issue is evident, and applying some sophisticated algorithms might provide better results, but on the other hand these algorithms would go beyond the restrictions imposed by the hardware platform. This work considers two-step transformation: background subtraction and projection of obtained image onto horizontal and vertical axes.

During background subtraction the image that represents the difference between the original image and the background image is generated. So, it is assumed that the background image is available (i.e. the image made at the moment when there are no any additional, temporary objects within the camera field of view), and that the camera is

Figure 3: An example of background subtraction functioning, left to right, top-down: original image, background image, subtracted image.

not moving (moving requires updating the background image). The example of background subtraction is represented in Fig. 3. As one can see, an object presence (person in this case) leads to the brighter zones in the subtracted image.

In order to generate feature vectors from subtracted samples obtained at the previous step, the samples are "projected" onto horizontal and vertical axes as follows: first i_x features of the vector are mean gray values for columns of pixels; the remaining i_y features are mean gray values for the rows. Therefore, such a projection leads to the reduction of feature number from $i_x \times i_y$ to $i_x + i_y$. Besides, the features are normalized to $[0, 1]$ range.

5.3 Experiments and Results

The initial images have been acquired during 4 different sessions. The sessions differ by place, time, and lighting conditions. For example, some places simultaneously had two different kinds of illumination sources: artificial (daylight lamps) and natural (windows). So the people passing behind provoked soft shadows in the camera field of view, and partial cloudiness added a slight brightness fluctuation. Therefore the resulting data sets are characterized by sufficiently high level of heterogeneity and soundness. As the result, 219 positive samples have been generated. Negative objects (like boxes, hall trees, etc.) were less numerous. In order to create balanced data, additional negative samples have been generated. To this aim poorly scaled or centered images of people have been used (e.g. people located too close or too far). In total, 438 samples have been obtained. 140 randomly chosen ones have been preserved for test set, whereas the rest 298 ones have been used for training. The images size is $i_x \times i_y = 15 \times 30$ pixels.

Table 8: Accuracy for Person Detection Problem, Gaussian Kernel (Floating Point) and Hardware-Friendly Kernel (HFK, Floating Point and Fixed Point). l Denotes the Number of Training Samples. Fixed Point: Data Width Is 16 Bit, Fractional Part Width Is 8 Bit

Problem		Gaussian	HFK	HFK, fixed point
SVM	$accuracy(\%)$	91.4	94.3	95.7
$N_{ev}/l = 2\%$	RSVMS	83.6	84.3	82.1
	ERSVMS	94.3	94.3	93.6
	RSVMO	82.1	82.9	82.1
	ERSVMO	95.0	95.0	94.3
$N_{ev}/l = 4\%$	RSVMS	88.6	87.9	85.7
	ERSVMS	92.9	94.3	96.4
	RSVMO	88.6	87.9	86.4
	ERSVMO	93.6	93.6	92.1

As for standard data sets, 5-fold cross validation has been used for model selection. The summary of results obtained for both kernels is presented in Table 8. As for standard data sets, ERSVM provided higher accuracy than RSVM. Besides, ERSVM accuracy remained almost the same when N_{ev} has been decreased (this time from 11 to 5), and this accuracy is compatible with that provided by classic SVM, whereas N_{ev} is much lower than N_{sv}: 5 versus 145 for the Gaussian kernel, and 5 versus 221 for the hardware-friendly kernel. Thus, in accordance with Section 3, classification of a sample requires less than 1 KB of memory (data + instructions) and around 24 320 microcontroller clock cycles (around 6.1 ms at 4 MHz, around 1.3 ms at 18.432 MHz).

In order to estimate minimal number of SVs that can be delivered by standard SVM, alternative model selection has been also performed: 5-fold cross validation has been used, but number of SVs (instead of accuracy) has been considered as the criterion. The best models were charac-

Figure 4: The most frequently misclassified samples. Two leftmost: false negative misclassifications, two rightmost: false positive misclassifications.

terized by 46 SVs (Gaussian kernel) and 87 SVs (hardware-friendly kernel), both providing 92.9% accuracy. Summing up, even in this case the number of SVs is 9-17 times higher than that provided by ERSVM, whereas the accuracy is lower.

The most frequently misclassified samples for both SVM and ERSVM cases are presented in Fig. 4. As regards false negatives (two leftmost images), the first sample corresponds to the situation when the contrast between the person's clothes and the background is low. Background subtraction procedure led to a very "dark" sample, which has been misclassified. The second image represents a tricky situation: a person with extended arms, when fitting the entire person led to non-standard scale. As regards false positives (two rightmost images), the first image has been considered as a negative sample because of poor person centering and additional object presence (at the left). The object in the second image is a hall tree with a pullover and a backpack put on, which is undoubtedly a tricky and controversial negative objects, and SVM could permit itself to be mistaken in this case. So, the major part of these samples can be considered as outliers.

6 Conclusions

In this work we have proposed SVM-like algorithms suitable for the design of embedded computational intelligence on resource-constrained hardware platforms. The training phase is composed of two different levels of optimization. The inner one consists in solving a standard RSVM problem in primal, while the outer one considers SVs as free parameters to be optimized. Solving primal instead of dual provides such advantages upon the training phase as reduced computational complexity and memory usage. A recently proposed hardware-friendly kernel, whose calculation requires only shift and add operations, has been considered along with classical Gaussian kernel.

The validity of the proposed approach has been demonstrated on several well-known data sets. The obtained experimental results have shown that ERSVM mostly outperformed classical RSVM, and the obtained classification error is close to the one obtained with standard SVM, while the number of EVs were by order of magnitude less than the number of SVs. Moreover, using the hardware-friendly kernel in place of the Gaussian one did not affect the classification performance significantly.

Before implementing the proposed algorithms on FPSLIC-based platform, their behavior has been studied when a fixed-point coding were used. The results indicate that the proposed method and architecture can be effectively implemented on simple microcontrollers.

The case-study considered in this work is the design of a node in a distributed WVSN for people detection. Background subtraction and image projection have been considered as the preprocessing steps. The heterogeneous data sets have been generated, and the obtained results suggest that the present technology allows the design of simple intelligent systems able to execute local classification tasks, thus incrementing the notion of invisible and pervasive computing.

Appendix
Derivation of Gradient of $W(\boldsymbol{z})$

Formulas (14-15) can be rewritten in the following way:

$$W(\boldsymbol{z}) = \min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q^z \boldsymbol{\alpha} + C\sum_{k=1}^{l}[t_k(\boldsymbol{\alpha}, \boldsymbol{z})]_+^2 \quad (21)$$

where $t_k(\boldsymbol{\alpha}, \boldsymbol{z}) = 1 - y_k \sum_{j=1}^{N_{ev}} \alpha_j K(\boldsymbol{z}_j, \boldsymbol{x}_k)$, and the derivative is calculated as follows:

$$\begin{aligned}\frac{\partial W(\boldsymbol{z})}{\partial \boldsymbol{z}_i} &= \frac{\partial(\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q^z \boldsymbol{\alpha} + C\sum_{k=1}^{l}[t_k(\boldsymbol{\alpha},\boldsymbol{z})]_+^2)}{\partial \boldsymbol{z}_i} \\ &= \frac{\partial(\frac{1}{2}\tilde{\boldsymbol{\alpha}}^T Q^z \tilde{\boldsymbol{\alpha}} + C\sum_{k=1}^{l}[t_k(\tilde{\boldsymbol{\alpha}},\boldsymbol{z})]_+^2)}{\partial \boldsymbol{z}_i} \\ &= \frac{1}{2}\frac{\partial \tilde{\boldsymbol{\alpha}}}{\partial \boldsymbol{z}_i}^T Q^z \tilde{\boldsymbol{\alpha}} + \frac{1}{2}\tilde{\boldsymbol{\alpha}}^T Q^z \frac{\partial \tilde{\boldsymbol{\alpha}}}{\partial \boldsymbol{z}_i} + \frac{1}{2}\tilde{\boldsymbol{\alpha}}^T \frac{\partial Q^z}{\partial \boldsymbol{z}_i}\tilde{\boldsymbol{\alpha}} \\ &+ \frac{\partial(C\sum_{k=1}^{l}[t_k(\tilde{\boldsymbol{\alpha}},\boldsymbol{z})]_+^2)}{\partial \boldsymbol{z}_i} \\ &+ \frac{\partial(C\sum_{k=1}^{l}[t_k(\boldsymbol{\alpha},\boldsymbol{z})]_+^2)}{\partial \boldsymbol{\alpha}}|_{\boldsymbol{\alpha}=\tilde{\boldsymbol{\alpha}}}\frac{\partial \tilde{\boldsymbol{\alpha}}}{\partial z_i} \\ &= \frac{\partial \tilde{\boldsymbol{\alpha}}}{\partial \boldsymbol{z}_i}\{Q^z\tilde{\boldsymbol{\alpha}} + \frac{\partial(C\sum_{k=1}^{l}[t_k(\boldsymbol{\alpha},\boldsymbol{z})]_+^2)}{\partial \boldsymbol{\alpha}}|_{\boldsymbol{\alpha}=\tilde{\boldsymbol{\alpha}}}\} \\ &+ \frac{1}{2}\tilde{\boldsymbol{\alpha}}^T \frac{\partial Q^z}{\partial \boldsymbol{z}_i}\tilde{\boldsymbol{\alpha}} + \frac{\partial(C\sum_{k=1}^{l}[t_k(\tilde{\boldsymbol{\alpha}},\boldsymbol{z})]_+^2)}{\partial \boldsymbol{z}_i} \quad (22)\end{aligned}$$

where $\tilde{\boldsymbol{\alpha}}$ is the solution of problem (15) given by a fixed $\boldsymbol{z}$, and $\frac{\partial Q^z}{\partial \boldsymbol{z}_i}$ is a $N_{ev} \times N_{ev}$ dimension matrix whose components are d-dimensional vectors $[\frac{\partial Q^z}{\partial \boldsymbol{z}_i}]_{mn} = \frac{\partial Q^z_{mn}}{\partial \boldsymbol{z}_i}$, $m,\ n = 1, \ldots, N_{ev}$.

Since $\tilde{\boldsymbol{\alpha}}$ is the solution of problem (15), the expression in curly brackets in (22) is equal to zero. In addition, for the symmetrical kernels of the form $K(\boldsymbol{u},\boldsymbol{v}) = K(\boldsymbol{u}-\boldsymbol{v})$, which are considered in this article, the second summand of (22), i.e. $\frac{1}{2}\tilde{\boldsymbol{\alpha}}^T \frac{\partial Q^z}{\partial \boldsymbol{z}_i}\tilde{\boldsymbol{\alpha}}$, is equal to zero as well. So, (22) transforms to

$$\frac{\partial W(\boldsymbol{z})}{\partial \boldsymbol{z}_i} = \frac{\partial(C\sum_{k=1}^{l}[t_k(\tilde{\boldsymbol{\alpha}},\boldsymbol{z})]_+^2)}{\partial \boldsymbol{z}_i}\,. \quad (23)$$

Since u_+, $u \in \mathbb{R}$, is non-differentiable at $u = 0$, it has been approximated by the following function [15]:

$$\begin{aligned}P_\beta(u) &= u + \beta^{-1} log(1 + e^{-\beta u}) \\ \frac{\partial P_\beta(u)}{\partial u} &= 1 - \frac{e^{-\beta u}}{1 + e^{-\beta u}} \quad (24)\end{aligned}$$

which converges to u_+ when smoothing parameter β approaches infinity. Such an approximation transforms (23) to

$$\begin{aligned}\frac{\partial W(\boldsymbol{z})}{\partial \boldsymbol{z}_i} &= 2C\sum_{k=1}^{l} P_\beta(t_k)\frac{\partial P_\beta(t_k)}{\partial t_k}\frac{\partial t_k}{\partial \boldsymbol{z}_i} \\ &= -2C\tilde{\alpha}_i \sum_{k=1}^{l} y_k P_\beta(t_k)\frac{\partial P_\beta(t_k)}{\partial t_k}\frac{\partial K(\boldsymbol{z}_i,\boldsymbol{x}_k)}{\partial \boldsymbol{z}_i} \quad (25)\end{aligned}$$

The only term still undefined is $\frac{\partial K(\boldsymbol{z}_i,\boldsymbol{x}_k)}{\partial \boldsymbol{z}_i}$. In case of Gaussian kernel

$$\frac{\partial K(\boldsymbol{z}_i,\boldsymbol{x}_k)}{\partial \boldsymbol{z}_i} = -2\gamma e^{-\gamma(\|\boldsymbol{z}_i - \boldsymbol{x}_k\|^2)}(\boldsymbol{z}_i - \boldsymbol{x}_k)\,.$$

Using hardware-friendly kernel involves calculation of absolute value in odd power, which is non-differentiable. Therefore, the approach similar to that used in (24) has been applied, and absolute value $\|u\|$ has been approximated by the function

$$\begin{aligned}Q_\beta(u) &= P_\beta(u) + P_\beta(-u) \\ &= \beta^{-1}[log(1 + e^{-\beta u}) + log(1 + e^{\beta u})] \quad (26) \\ &= \beta^{-1}[log(2 + e^{-\beta u} + e^{\beta u})]\,.\end{aligned}$$

After several transformations the following result is obtained for hardware-friendly kernel:

$$\frac{\partial K(\boldsymbol{z}_i,\boldsymbol{x}_k)}{\partial \boldsymbol{z}_i} = -\gamma log(2)\cdot \tilde{K}(\boldsymbol{z}_i,\boldsymbol{x}_k)\cdot \boldsymbol{s\tilde{g}n}(\boldsymbol{z}_i - \boldsymbol{x}_k) \quad (27)$$

where

$$\begin{aligned}\tilde{K}(\boldsymbol{u},\boldsymbol{v}) &= 2^{-\gamma[Q_\beta(\boldsymbol{u}-\boldsymbol{v})]_1} \\ \boldsymbol{s\tilde{g}n}(\boldsymbol{u}) &= \frac{e^{\beta\boldsymbol{u}} - e^{-\beta\boldsymbol{u}}}{2 + e^{\beta\boldsymbol{u}} + e^{-\beta\boldsymbol{u}}}\end{aligned}$$

converge to (18) and sign function respectively as β approaches infinity. Two last formulas imply element-by-element (vectorized) division, exponent calculation, and calculation of Q_β.

References

[1] D. Norman, *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution.* Boston: MIT Press, 1999.

[2] B. Warneke, M. Last, B. Leibowitz, and K. Pister, "Smart Dust: Communicating with a cubic-millimeter computer," *IEEE Computer*, vol. 34, no. 1, pp. 44–51, Jan. 2001.

[3] M. Duarte and Y. H. Hu, "Vehicle classification in distributed sensor," *Journal of Parallel and Distributed Computing*, vol. 64, no. 7, pp. 826–838, 2004.

[4] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, "Habitat monitoring with sensor networks," *Commun. ACM*, vol. 47, no. 6, pp. 34–40, 2004.

[5] V. Vapnik, *The Nature of Statistical Learning Theory.* Springer, 1995.

[6] B. Schölkopf and A. Smola, *Learning with Kernels.* The MIT Press, 2002.

[7] D. Anguita, S. Pischiutta, S. Ridella, and D. Sterpi, "Feed-forward support vector machine without multipliers," *IEEE Trans. on Neural Networks*, vol. 17, no. 5, pp. 1328–1331, 2006.

[8] A. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proceedings of 17th International Conference on Machine Learning*, 2000, pp. 911–918.

[9] C. K. I. Williams and M.Seeger, "Using the nystrom method to speed upkernel machines," in *In Todd K. Leen, Thomas G.Dietterich,and Volker Tresp, editors,* Advances in Neural InformationProcessing Systems *13, Cambridge, MA, MIT Press.*, 2001, pp. 682–688.

[10] Y.-J. Lee and O.L.Mangasarian, "RSVM: Reduced support vector machines," in *First SIAM International Conference on Data Mining*, Chicago, Apr. 2001.

[11] S. Keerthi, O. Chapelle, and D. DeCoste, "Building support vector machines with reduced classifier complexity," *Journal of MachineLearning Research*, vol. 7, pp. 1493–1515, 2006.

[12] M. Wu, B. Schölkopf, and G. Bakir, "A direct method for building sparse kernel learning algorithms," *Journal of Machine Learning Research*, vol. 7, pp. 603–624, 2006.

[13] "Atmel web site." [Online]. Available: http://www.atmel.com/

[14] V. Vapnik, *Statistical Learning Theory.* Wiley, 1998.

[15] K.-M. Lin and C.-J. Lin, "A study on reduced support vector machines," *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1449–1459, 2003.

[16] M. Wu, B. Schölkopf, and G. Bakir, "Building sparse large marginclassifiers," in *Proceedings of the 22nd International Conferenceon Machine Learning*, 2005, pp. 1001–1008.

[17] O. Chapelle, "Training a support vector machine in the primal." [Online]. Available: http://www.kyb.mpg.de/publication.html?user=chap

[18] "Gnu project." [Online]. Available: http://www.gnu.org/

[19] D. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical Programming*, vol. 45, pp. 503–528, 1989.

[20] "Uci repository of machine learning databases." [Online]. Available: http://www.ics.uci.edu/ mlearn/

[21] D. Michie, D. Spiegelhalter, C. Taylor, and J. Campbell, *Machine learning, neural and statistical classification.* Ellis Horwood Upper Saddle River, NJ, USA, 1995.

[22] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for AdaBoost," *Machine Learning*, vol. 42, no. 3, pp. 287–320, 2001.

[23] M. Yokoyama and T. Poggio, "A contourbased moving object detection and tracking," in *Proceedings on 2nd Joint IEEE International Workshop on VS-PETS*, October 2005, pp. 271–276.

[24] S. Munder and D. M. Gavrila, "An experimental study on pedestrian classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 11, pp. 1863–1868, 2006.