Enhanced Least Squares Support Vector Machines for Decision Modeling in a Multi-Sensor Fusion Framework

Andreas Klausner
Institute for Technical Informatics
Graz University of Technology
Graz, AUSTRIA
klausner@iti.tugraz.at

Allan Tengg
Institute for Technical Informatics
Graz University of Technology
Graz, AUSTRIA
tengg@iti.tugraz.at

Bernhard Rinner
Institute of Networked and Embedded Systems
Klagenfurt University
Klagenfurt, AUSTRIA
bernhard.rinner@uni-klu.ac.at

Abstract

In this article we introduce a software framework for embedded online data fusion on different levels of data abstraction. We present our data oriented fusion model and introduce the main functional units. The paper is focused to the decision modeling process. In our approach we use Support Vector Machines (SVM) as well as Least Squares SVM (LS-SVM) for decision modeling. Due to the computation complexity and the necessary memory requirements we prefer LS-SVM for the classification tasks. The main disadvantage of LS-SVM is the loss of sparseness by using equality constraints instead of inequality constraints in the cost function. We introduce a novel method for intelligent data preselection (PTD LS-SVM) to compensate for this short coming. Experimental results demonstrate the feasibility of this approach.

1. Introduction

Currently there is a strong trend towards integration of sensor, computing and communication technology into everyday life. The ultimate goal is to provide as much support as possible while concealing the computing devices from the users. Our I-SENSE project [1] demonstrates the potential of combining scientific research areas, multi-sensor data fusion and pervasive embedded computing. The main idea is to provide a generic software framework which allows online data fusion on a distributed embedded system. This software fusion framework is implemented on an embedded device and was therefore limited in memory resources. In our multi-level data fusion framework *Support Vector Machines (SVM)*, proposed by Vapnik [2,3] are used as classification method for decision modeling. Necessary time and memory usage are the main bottlenecks for training kernel methods, such as SVM. For N>3000, where N is the number of training data sets, common SVM learning strategies are not feasible, especially on embedded platforms because of their memory usage and the time required. Therefore, a modified version of the original SVM, the so called *Least Squares Support Vector Machine (LS-SVM)* [4,5] is used for decision modeling in our framework. The main characteristic of LS-SVMs is the lower computational complexity compared with original SVMs, without any quality loss in the classification results. LS-SVM and the original SVM are based on the same principals. The main difference is, that LS-SVM formulation uses equality constraints instead of inequality constraints for the cost function, which have to be minimized. LS-SVM has the attractive feature, that training requires solving a set of linear equations, instead of a quadratic programming (QP) problem, which is required by standard SVM. Solving these linear equations is less complex than solving QP problem.

An attractive feature of SVM, namely the sparseness is lost by the LS-SVM formulation. In standard SVM many Lagrange multipliers are zero, leading to a smaller subset of learning data in order to build the decision boundary between the two classes. In LS-SVM almost all multipliers are non-zero, indicating that all training data sets will be used as support vectors. This extraction of support vectors from a given training dataset is comparable with the problem formulation of finding the most significant vectors in a given data set. The optimal solution for solving this task should combine the following features. It should (i) be fast, (ii) lead to a sparse solution (i.e. low number of support vectors) and (iii) produce good classification results.

We propose a method for an intelligent preselection of learning data in order to reduce the training set and therefore reduce the number of support vectors which are then used by the LS-SVM classifier. Therefore a modified nearest neighbour rule is used to select an optimized set of training data which is provided to the LS-SVM. We show, that using this approach leads to the following advantages: (i) a reduced number of support vectors, which have to be stored for the classification task, (ii) the extraction of support vectors from a given dataset is

easier when the number of possible vectors is reduced, and (iii) the computation time, necessary to evaluate a new vector, is reduced.

The remainder of the paper is organized as follows: Section 2 discusses our I-SENSE project as a framework for multi-sensor data fusion. Section 2.1 deal with the software fusion framework and introduce the main functional units. In section 3 we present the decision modeling process where section 3.1 gives an overview of common least squares SVM and section 3.2 presents our scientific contribution, namely the intelligent preselection of training data in order to compensate the main disadvantage of standard LS-SVM - the loss of sparseness. In section 4 we present some experimental results of our approach, and section 5 concludes the paper with a short discussion of our approach.

2. The I-SENSE fusion framework

I-SENSE is used as an acronym for intelligent sensing and our project is targeted at various classification applications. Case studies in vehicle classification as well as in waste separation and food classification are used to demonstrate the feasibility of the proposed fusion framework. As the name multi-senor data fusion implies, it is a technique by which data from several sensors are combined through a data processor to provide comprehensive and accurate information. Although the provision of a single data stream from multiple inputs is advantageous, the powerful potential of this technology stems from its ability to track changing conditions and anticipate impacts more consistently than can traditionally be done with a single data source. Thus, the aim of our multi-sensor fusion framework is to create a synergistic process in which the consolidation of individual data creates a combined resource with a productive value greater than the sum of its parts.

To achieve this aim, our approach is to perform multi-level data fusion by combining data from different sensors at different levels of abstraction. The I-SENSE fusion-framework supports three basic levels of data fusion. These fusion levels are differentiated according to the amount of information they provide. The most basic level is called *raw-data fusion.* At this level, only raw, uncorrelated data is provided to the user. In comparison, level two data fusion provides a higher level of inference and delivers additional interpretive meaning suggested from the raw data and individual features are extracted from observed objects based on the information of one sensor. These features from multiple data sources are fused on feature level in order to obtain a combined feature vector of an observed object. Therefore this level is called *feature-level fusion*. Level three data fusion is designed to make assessments and provide recommendations to the user and is called *decision fusion.* Thus, each jump between data fusion levels represents a corresponding leap in technological complexity to produce increasingly valuable informational detail. In our approach data fusion is performed on the individual sensor nodes using data from the local sensors as well as abstracted sensor data from the adjacent sensor nodes.

2.1. Fusion Model

Figure 1 presents the detailed, data oriented *software fusion model* in our I-SENSE approach. This model consists of three basic levels and functions. Namely, the *Sensor Control & Management Unit*, the *Sensing Unit* and the *Fusion Layer*.

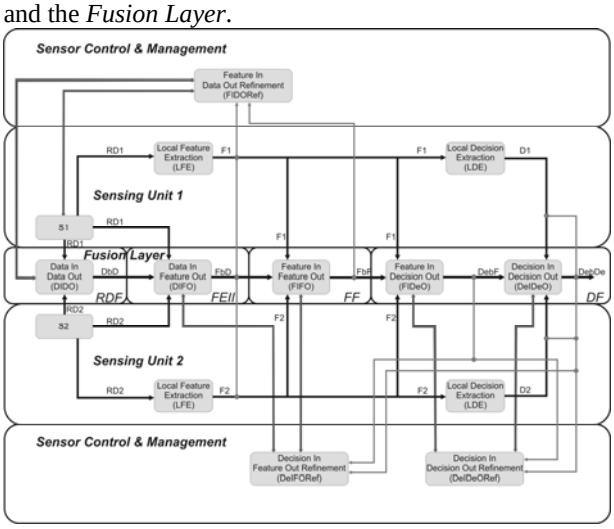

Figure 1: Detailed data oriented Fusion Model characterized by input- and output data

The figure presents an example of two physical sensors (i.e., audio sensor and visual sensor). The *Sensor Control & Management Unit* is responsible for the sensor identification as well as to provide the interface to other sensing nodes, human observers and actuators.

Furthermore, this unit controls the overall fusion process and provides access to a database where resource requirements for the different fusion computations are stored.

Three essential functional blocks are identified on this layer to allow an online refinement of the overall fusion process:

(i) *DeIDeORef*: *Decision in decision out refinement;* This functional unit allows to refine the decision extraction and decision fusion algorithms dynamically during the fusion process, based on the generated output decisions of *FIDeO* and *DeIDeO*. Based on statistical output information the process of decision modeling can be modified and refined during runtime.

(ii) *DeIFORef: Decision in feature out refinement;* This functional unit allows refining the data allocation and raw-data based fusion algorithms dynamically during

the fusion process based on the generated output features of *FIDeO* and *DeIDeO*.

(iii) *FIDORef: Feature in data out refinement;* This functional unit deals with refining the feature extraction and feature fusion algorithms dynamically during the fusion process based on the generated output features of *FIFO* and *DIFO*.

The *Sensing Unit* represents the intelligent sensor which consists of the physical sensor itself and a suitable data pre-processor (e.g. resolution based down-sampling, automatic gain control, …). A *Local feature extraction Unit (LFE)* is used to extract a single-source feature vector based on color information, structural information, spectral information or acoustic information of an observed object. This means, each sensor provides an estimate of the position of an object with extracted features, based only on its own single source data. These individual feature vectors are input to a data fusion process, namely the *Feature in feature out (FIFO)* process, in order to achieve a joint feature vector estimate based on multiple sensors. A *Local decision extraction Unit (LDE)* is used to extract local decision from individual objectives features (e.g. classification of objectives identity).

The heart of the framework is the *Fusion Layer* including the following five functional units:

(i) *DIDO: Data in data out unit*; This functional unit is also called *Raw-Data Fusion unit (RDF)*, and raw uncorrelated data will be fused from different and/or similar numerous sensors there. This raw data streams are labelled with *RDx*. At raw-data based fusion each sensor performs a single-source estimate in the sensor state space. These estimates are then combined to an aggregate estimate. In our case study for similar sensor types the data will be combined to single data stream based on numerous data streams from different sensors (e.g. visual sensor & infrared spectral imager, …) by using wavelet based image fusion techniques.

(ii) *DIFO: Data in feature out unit*; this is our so called *Feature extraction II unit (FEII)*, where raw data from the individual sensors and/or fused raw-data (i.e. *DbD: Raw data based on raw data)* is used to extract suitable features of the individual tracked objects. These features are found by experimental analyses and/or physical modeling. The output data are feature vectors for each detected object in the observed area.

(iii) *FIFO: Feature in feature out unit*; this is our so called *Feature fusion unit (FF)*, where features will be fused to a resulting overall feature vector for each individual detected object. Therefore it is necessary to find the corresponding objects in the individual sensor spaces. In our framework we have implemented methods for similar sensor types, where simple object overlapping calculation is performed in order to find the corresponding elements. Furthermore, a time stamping mechanism was developed to find corresponding objectives in different sensor spaces. The output data of this fusion process are feature vectors based on features (*FbF*) extracted by the *LFE* unit or features extracted by the *DIFO* unit.

(iv) *FIDeO: Feature in decision out unit*; this functional unit is a part of our *decision fusion unit (DF)*, where, a classifier, based on SVM, is trained with previously recorded and classified sequences. In the next step this SVM is used as a classifier to derive classification decisions based on previously extracted single source feature vectors ore joint feature vectors (*FbF*) from the *FIFO* unit. Outputs are decisions based on Features (*DbF*), and a probability interval of this decision.

(v) *DeIDeO: Decision in decision out unit*; this functional unit is the second part of our *decision fusion unit*, where extracted decisions (*Dx*) will be fused from multiple sensors from the *LDE* unit with fused data from *FIDeO*, based on statistical methods (i.e. Dempster-Shafer methods [6,7]). Outputs are decisions, based on multiple decisions.

3. Decision modeling

In our fusion framework decision modeling is driven by Support Vector Machines (SVM). LS-SVM is a faster learning strategy than standard SVM.

3.1. Least squares support vector machines

The least squares support vector machine classifier [4,8] is a modification of standard SVM. A training data set is given by $\{(x_i, y_i)\}_{i=1}^{N}$ with the inputs $x_i \in \Re^d$ and class labels $y_i \in \{1,-1\}$. The idea of SVM classifier is to find the linear separating hypersurface $\omega^T \varphi(x) + b = 0$ in the feature space *F* that separates the mapped data $\{(\varphi(x_1), y_1),...,(\varphi(x_N), y_N)\}$. According to statistical learning theory [2,3] a good generalization is given if one demands that both classes are separated with a certain margin. The goal is to find the appropriate weight vector ω and the scalar bias term *b*, such that the following relations hold $\forall i = \{1,...,N\}$:

$$\begin{cases} \omega^T \varphi(x_i) + b \geq +1 \rightarrow \text{if } y_i = +1 \\ \omega^T \varphi(x_i) + b \leq -1 \rightarrow \text{if } y_i = -1 \end{cases} \quad (1)$$

Instead of building one hyperplane as in standard SVM (cp. figure 2a), LS-SVM builds two parallel hyperplanes; one for the positive class and one for the negative class as is indicated in figure 2b. The distance between these hyperplanes $\omega^T \varphi(x) + b = +1$ and $\omega^T \varphi(x) + b = -1$ in the feature space is called the

separating margin. Finding the separating hyperplane deals with the problem that this margin has to be maximized.

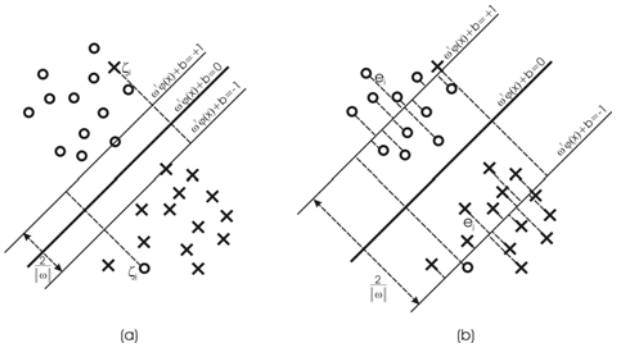

Figure 2: Classification behavior in *Feature Space* of (a) Standard SVM with the margin for the separating hyperplane and the misclassification measure ζ_i and (b) LS-SVM with two parallel hyperplanes and the error e_i attached to each point

Using Vapnik's formalism [2,3] from standard SVM leads to a constraint quadratic programming (QP) problem. In order to avoid this optimization problem which is sometimes difficult to solve, LS-SVM uses equality constraints instead of inequality constraints to find the decision hyperplanes. The difference is compensated by adding an extra term to the cost function that penalizes the deviations from the two hyperplanes, for each point of the learning data. The deviations are given by the scalar error $e_i \forall i \in \{1,...,N\}$. The training problem is given by:

$$\begin{cases} \min_{\omega,e,b} \quad \frac{1}{2}\|\omega\|_2^2 + \frac{\gamma}{2}\|e\|_2^2 \\ \text{s.t. } y_i\left(\omega^T\varphi(x_i)+b\right) = 1-e_i \quad \forall i \in \{1,...,N\} \end{cases} \tag{2}$$

where γ plays the role of a regularization parameter between the two quadratic terms in the *primal problem formulation* (2), and characterizes the relative importance of the terms. The first term aims to maximize the distance between the two hyperplanes, while the second term aims to minimize the slack variable e_i. This addition of the two quadratic terms is also responsible for the name least squares SVM.

Since the dimension of the feature space is high, possibly infinite, this problem is difficult, if not impossible, to solve. Furthermore, the mapping of $\varphi(.)$, corresponding to a kernel, is not always known. For solving an optimization problem the Lagrangian is constructed. The optimality conditions of this constrained optimization problem are given by the saddle point of this Lagrangian also known as the Karush-Kuhn-Tucker conditions [9]. The Lagrangian is given by:

$$\begin{aligned} L(\omega,b,e,\alpha) &= \frac{1}{2}\|\omega\|_2^2 + \frac{\gamma}{2}\|e\|_2^2 \\ &\quad - \sum_{i=1}^{N}\alpha_i\left(y_i\left(\omega^T\varphi(x_i)+b\right)-1+e_i\right), \end{aligned} \tag{3}$$

where α is the vector of the Lagrange multipliers. Using the Karush-Kuhn-Tucker conditions substituting the result of the linear equations into (3); ω and b can be eliminated and the *Dual optimization problem* is given by:

$$\begin{cases} \max_{\alpha} \quad -\frac{1}{2}\sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j \left(k(x_i,x_j) + \frac{1}{\gamma}\vartheta_{i,j} \right) + \sum_{i=1}^{N}\alpha_i \\ \text{s.t.} \quad \sum_{i=1}^{N}\alpha_i y_i = 0, \qquad \vartheta_{i,j} = \begin{cases} 1 \to i = j \\ 0 \to i \neq j \end{cases} \end{cases} \tag{4}$$

This formulation has three important advantages. First we see that the dimensionality of the optimization problem is equal to the number of data points *N*, indicating that the training process is neither dependent on the dimensionality of the *feature space*, nor the dimension of the *input space*. Second, the kernel is an inner product in feature space and opens therefore a lot of opportunities. By applying the kernel trick one can show that for different kernels the general optimization problem remains the same. Third, these optimization problems are convex. The Hessian matrix is a full rank positive definite matrix, which guarantees a unique solution. The solution of the *dual optimization problem* can be given as a linear system:

$$\begin{bmatrix} 0 & y^T \\ y & \mathbf{H} \end{bmatrix}\begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ 1_N \end{bmatrix} \tag{5}$$

where the matrix **H** obeys the Mercer theorem [9], that deals with the conditions a function must have to be a kernel function, and *y* denotes the column vector formed by the labels of the training points. After the optimal parameters are found the classifier is given in the form:

$$f(x) = sign\left(\sum_{i=1}^{N}\alpha_i y_i k(x,x_i) + b\right) \tag{6}$$

Notice, in LS-SVM all Lagrangian multipliers are non-zero, because all training data sets are used as support vectors for identifying the class separation surface.

3.2 Training data preselection for LS-SVM

In this section we describe a method for selecting vectors out of the training dataset which are likely to be support vectors in a LS-SVM and therefore describe the individual classes best. This is done by an intelligent data preselection algorithm based on a modified nearest neighbor technique, leading to a smaller set of samples which have to be stored for the classification task - resulting in quite similar classification accuracies.

After this preselection, the remaining datasets are used as support vectors for a LS-SVM classifier to find the decision boundary between two classes in the learning process. Using our approach leads to a sparse LS-SVM classifier with good classification results and lower computational and memory costs than standard SVM. In embedded systems the memory resources are quite restricted and therefore the proposed approach is advantageous in comparison to standard SVM. The training data preselection algorithm consists of three main stages as described in the following:

A given training dataset T is given by the training samples s. The training samples can be divided into two subsets $A \in \{a_0,\ldots,a_n\}$ and $B \in \{b_0,\ldots,b_n\}$ characterizing the two involved classes.

Stage 1:

1. For each sample a_i out of A find the nearest neighbor sample b_j from B and vice versa by computing the Euclidean distance $d(ai,bj)$ $\forall a_i,b_j$ until the distance is a minimum.
2. Using the result from the first step, sort the distance $d(ai,bj)$ in ascending order.
3. The first distance sample is stored in an initially empty set Ω_{nn}.
4. The next distance samples are iteratively added to Ω_{nn} and classified with a simple nearest neighbor rule in the way:

   If $\min(d(s,a_i)) < \min(d(s,b_j))$ $\forall a_i,b_j \in \Omega_{nn}$ *and* $\forall s \in T$

   $s \in A$

   else

   $s \in B$

   If the classification is wrong add the distance sample to Ω_{nn} else quit.

Stage 2: The reduced nearest neighbor rule [10] is used to obtain the reduced subset Ω_{rnn} from Ω_{nn}:

1. Initially copy Ω_{nn} *to* Ω_{rnn}
2. Remove the first/next pattern (a_i,b_j) of Ω_{rnn}
3. Use nearest neighbor rule to check if Ω_{rnn} classify all pattern correct in Ω_{nn}
   a. If all patterns classified correct go to 4)
   b. Else return the removed pattern to Ω_{rnn} and go to 4)
4. If every pattern has removed once go to 5) else go to 2) and remove the next pattern
5. The remaining subset is given by: $\Pi = T - \Omega_{rnn}$

Stage 3: At the last stage the final preselection subset T_{ps} is computed to obtain the samples Π which are closest to Ω_{rnn}.

1. Take the first/next pattern $(a_i,b_j) \in \Omega_{rnn}$ select k-nearest samples $(a_m,b_n) \in \Pi$ in a way that
   a. $\min (d(a_i,a_m))$
   b. $\min (d(b_j,b_n))$
   c. $\min (d(a_i,b_n))$
   d. $\min (d(b_j,a_m))$

   are given.
2. copy k samples to T_{ps} and go to 1) until all samples from Ω_{rnn} were considered.

The value k can be set by the user, but our experiments have shown that *3%* from the number of samples in T is a good initial value. The resulting subset of the training data, namely T_{ps}, is provided to the LS-SVM classifier.

Figure 3 shows an example random Gaussian distribution and the reduced subset Ω_{rnn} after *Stage 2* of the proposed algorithm. Figure 4 shows the overall results of the *training data preselection* algorithm with different values of *k*.

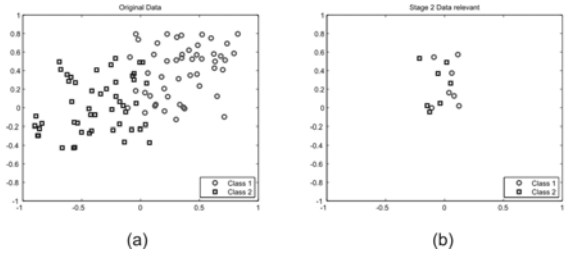

Figure 3: (a) Random Gaussian distribution, (b) reduced remaining subset Ω_{rnn}

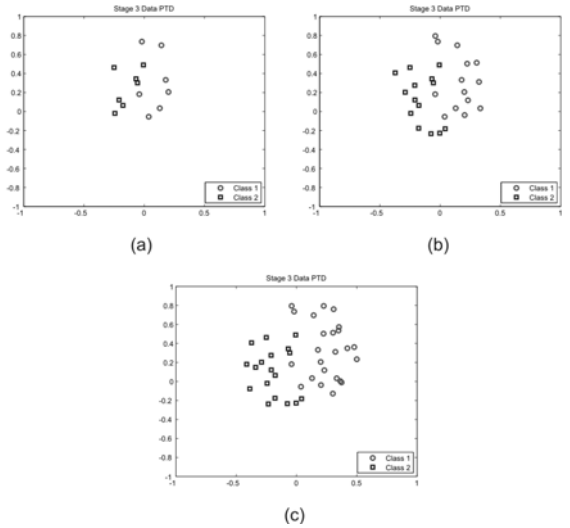

Figure 4: Results of the training data preselection algorithm for the distribution shown in figure 3a with variable k. (a) k=1, (b) k=2 and (c) k=3

4. Experimental results

In this section we present the results of three different experiments. Firstly we generate two random Gaussian distributions, each for an individual class. The distributions show a tendency to a high level of overlap as indicated in figure 5. Secondly we use a “real-world experiment” from our waste separation case study. We have implemented a simple multiclass classifier by using One-against-All technique.

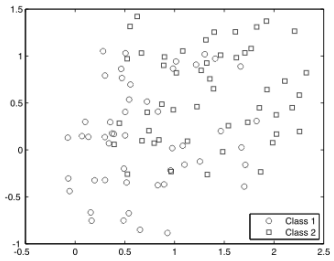

Figure 5: Random Gaussian distribution as experimental setup

The database consists of 6 different classes: (i) blue glass, (ii) green glass, (iii) brown glass, (iv) white glass, (v) ceramics, stones, porcelains (CSP) and (vi) plastics. In cooperation with our industrial partner we generate a database with about 200 samples for each class. Each sample is characterized by 5 histogram based color features, 3 structural features and 5 IR-histogram based features. For the feature extraction 2 different types of sensors were used: (i) CMOS visual sensors and (ii) IR-spectral imaging sensors.

Thirdly, we use again two random Gaussian distribution with increasing amount of data points for each class. We evaluate three different implementations in order to obtain a training time result.

For the first and the second experiment we use *Matlab*^{TM} implementations of 4 different algorithms: (i) standard SVM, (ii) Least squares SVM, (iii) LS-SVM with training data preselection (PTD-SVM) and (iv) a sparse LS-SVM, called LS^2-SVM [11].

The third experiment was performed using *C++* implementations and 3 algorithms were compared in the required time for training by an increasing large number of data points: (i) standard SVM, (ii) LS-SVM and (iii) LS-SVM with support vector preselection.

For all experiments we used *radial basis function (RBF)* as the kernel function. The results presented in the following tables (cp. Table 1 and Table 2) are average values from 20 runs with random selection of training datasets.

Table 1: Experiment 1 results; random Gaussian distribution

Algorithm	Training time (s)	Wrong classified (%)	Nb of SV	SV (%)
SVM	17.81±1.25	12.4±0.5	50.3±3.6	62.9±4.5
PTD LS-SVM	10.04±1.11	15.3±0.8	36.6±4.4	45.8±5.5
LS-SVM	3.85±0.47	14.2±0.5	80.0±0.0	100±0.0
LS²-SVM	4.13±0.64	12.8±0.6	52.8±4.2	66.0±5.3

Table 1 indicates that the standard SVM has the best training accuracy. Our proposed approach differs only about *3.1%* from the standard SVM, while being 45% faster in training than the standard approach. The results presented in this table also show that the training accuracy is quite similar for all training approaches. The fastest training strategy is the LS-SVM approach, followed by the LS²-SVM approach. Our proposed training data preselection LS-SVM approach is about 3 times slower than the fastest approach. The last column of the previous table shows us that our algorithm has detected a smaller amount of *Support Vectors (SV)* even in comparison to the standard SVM approach. In comparison to the LS-SVM approach our algorithm needed 45% of Support Vectors for a quite similar classification result.

Table 2: Experiment 2 results; waste separation

Algorithm	Training time (s)	Wrong classified (%)	SV (%)
SVM	328.3±4.4	2.8±0.2	40.8±4.5
PTD LS-SVM	158.4±3.2	3.1±0.3	42.3±3.2
LS-SVM	37.8±2.5	2.2±0.2	100±0.0
LS²-SVM	76.1±2.9	2.5±0.4	66.8±3.3

Table 2 confirms the tendencies from table 1. Our approach is about 50% faster than the Standard SVM approach with quite similar classification results. All presented approaches for training a Support Vector Machines seem to be high quality solutions for classification problems. The fastest training strategy is LS-SVM, followed by LS²-SVM as the table indicates. Our proposed approach is about 4 times slower than the fastest but a lower amount of Support Vectors have to be considered. Both tables show that the LS-SVM approach consider all training datasets as support vectors, which is the main disadvantage of this strategy.

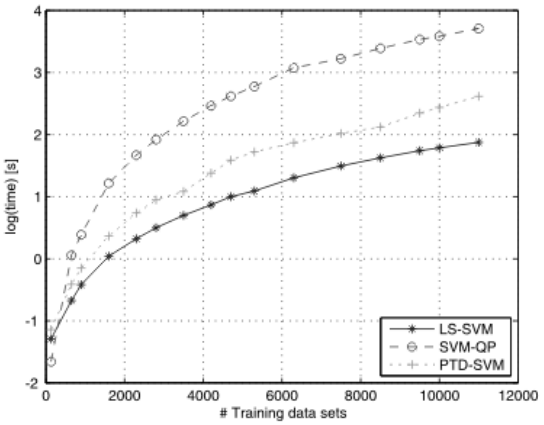

Figure 6: Comparison of three different learning strategies in required training time vs. number of training samples; SVM-QP approach (standard SVM), LS-SVM approach (least squares SVM) and PTD-SVM approach (preselection of training data LS-SVM)

Figure 6 shows a comparison of three training strategies for an increasing number of training samples.

The required training time is an averaged value of (i) 10 experiments for less than 5000 training samples and (ii) 3 experiments for more than 5000 training samples. The standard SVM approach is the slowest and is therefore not advisable for large training sets. The fastest approach is the LS-SVM approach, followed by our proposed approach. Both algorithms might be used for large training data sets.

5. Discussion

In embedded systems, learning of large training datasets with SVM is difficulty, because of their restricted memory resources. Our experiments show that training for over 3000 training samples with a standard SVM is not feasible because of the memory requirements.

LS-SVM helps to reduce the memory requirements and is much faster than standard SVM, due to the usage of equality constraints instead of inequality constraints. Therefore, training requires the solving of a set linear equations, instead of solving the quadratic programming (QP) problem. The main disadvantage of LS-SVM is the loss of sparseness, indicating that all training samples have to be stored for the classification task - leading to high memory requirements and slower classification.

Therefore, we described an algorithm for an intelligent training data preselection in order to identify a subset of training data which describes the whole dataset best. This approach reduces the number of vectors which have to be stored for later classification. The experimental results show that our approach leads to a *sparse SVM* with *accurate classification results* and *faster training time* than the standard SVM.

Future work will include the implementation of the proposed approach to a commercial product.

6. Acknowledgements

We thank the Austrian Research Promotion Agency for partially supporting this project.

7. References

[1] A. Klausner, B. Rinner, A. Tengg. “I-SENSE: Intelligent Embedded Multi-Sensor Fusion” In *Proceedings of the 4th IEEE International on Intelligent Solutions in Embedded Systems (WISES 2006)*. Vienna, Austria. June 2006.

[2] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[3] V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[4] J.A.K Suykens and J. Vandewalle. “Least squares support vector machine classifier.” *Neural Processing Letters*, 9(3):293–300, Jun 1999.

[5] J.A.K Suykens, P. Van Dooren, B. De Moor, and J. Vandewalle. “Least squares support vector machine classifiers: a large scale algorithm”. *European Conference on Circuit Theory and Design*, ECCTD’99, pages 839–842, 1999.

[6] J.D. Lawrence, and T.D. Garvey “Evidential Reasoning: A Developing Concept” In *Proceedings of the International Conference on Cybernetics and Society*, IEEE, 1982.

[7] A.P. Dempster. “A Generalization of Bayesian Inference” Journal of the Royal Statistical Society, vol. 30, pages 205-247, 1968.

[8] J.A.K Suykens and J. Vandewalle. “Multiclass least squares support vector machines”. In *IJCNN’99 International Joint Conference on Neural Networks*, Washington, DC, 1999.

[9] J. Nocedal and S. J.Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[10] G.W. Gates. “The reduced nearest neighbour rule“. In *IEEE Transactions on Information Theory*, 18(3), pages 431-433, May 1972.

[11] J. Valyon and G. Horvath. “A Sparse Least Squares Support Vector Machine Classifier“. In *IJCNN’04 International Joint Conference on Neural Networks*, 2004.