Proceedings of International Joint Conference on Neural Networks, San Jose, California, USA, July 31 – August 5, 2011

Parallel Semiparametric Support Vector Machines

Roberto Díaz-Morales, Harold Y. Molina-Bulla and Ángel Navia-Vázquez

Abstract— **In recent years the number of cores in computers has increased considerably, opening new lines of research to adapt classical techniques of machine learning to a parallel scenario. In this paper, we have developed and implemented with the multi-platform application programming interface OpenMP a method to train Semiparametric Support Vector Machines relying on Sparse Greedy Matrix Approximation (SGMA) and Iterated Re-Weighted Least Squares algorithm (IRWLS). We take advantage of the matrix formulation of SGMA and IRWLS. We recursively apply the partitioned matrix inversion lemma and other matrix decompositions to obtain a simple procedure to parallelize SVMS with good performance and computational efficiency.**

I. Introduction

Recently, kernel methods [19] have become very popular in the machine learning community due to their ability to transform the input data onto a high dimensional space where inner products between projected vectors can be computed using a kernel function, thereby easily providing nonlinear solutions.

Support Vector Machines (SVMs) [21] are considered the "state-of-art" to solve classification problems due to their performance working with high dimensional data in a wide variety of fields and their ability to adjust the machine size once its hyperparameters are set. The classifier size usually results very high, which motivates many research lines about performance and scalability.

Some methods propose to reduce the machine size using a reduced set [2][18][13][20], they start from a full SVM machine and solve a preimage problem [7]. In [14][10] a procedure to iteratively grow the architecture is proposed, so machine size and complexity are kept under control. In [14] Sparse Greedy Matrix Approximation (SGMA) is used to select in every iteration the candidate that gets the highest possible error descent. These methods are known as semiparametric SVMs.

There are several papers about parallel SVMs. Early works divide the data set in groups to train different SVMs and results are finally combined. [4] uses a multilayer perceptron and [5] uses the support vectors to train another SVM. In [3][6] a parallel implementation of "Sequential Minimal Optimization" SMO is developed for multiprocessors systems. These algorithms are based on decomposing the training process by dividing the data set into groups and combining the results to get a full solution. The main problem is the possibility of falling into a local minima if the subsets are not large enough to represent the statistics of the set.

The use of shared memory allows us to divide the training process in different threads that are able to access the full data set having a single instance in memory. OpenMP [12] is an application programming interface that supports multi-platform shared memory multiprocessing programming.

In this paper we derive a new method to train semiparametric SVMs based on SGMA and Iterated Re-Weighted Least Squares (IRWLS), directly implementable on a parallel architecture with shared memory.

II. Algorithm

In what follows we will describe the Parallel-Semiparametric SVM (PS-SVM) training method. We use the SGMA algorithm we obtain the base elements that our model is going to use and then the IRWLS algorithm is used to train the weights. In order to obtain parallel versions of both algorithms we take advantage of matrix decompositions that allow us to compute partial results on different processors in parallel.

A. Parallel Sparse Greedy Matrix Approximation

A kernel evaluation $k(x_i, .)$ can be approximated as a linear combination of other kernels:

$$k(x_i, .) = \sum_{j=1}^{n} \alpha_{ij} k(c_j, .) \tag{1}$$

The approximation error for a set $\{x_1,...,x_m\}$ once the weights α_{ij} are chosen is then:

$$Err(\alpha) = tr\mathbf{K} - \sum_{i=1}^{m}\sum_{i=1}^{n} \alpha_{i,j} k(x_i, c_j) \tag{2}$$

Where $\mathbf{K}$ is the kernel matrix m x m. When we add a new element c_{n+1} to the base the new error can be expressed as a function of the previous error [19]:

$$Err(\alpha^{m,n+1}) = Err(\alpha^{m,n}) - \eta^{-1}||\mathbf{K^{m,n}z} - \mathbf{k_{Sm}}||^2 \tag{3}$$

where:

$$(\mathbf{k_{Sm}})_i = k(x_i, c_{n+1}) \tag{4}$$
$$\mathbf{z} = \mathbf{K_C^{-1}} \cdot \mathbf{k_{mC}} \tag{5}$$
$$\eta = 1 - \mathbf{z^T k_{mC} z} \tag{6}$$
$$(\mathbf{k_{mC}})_i = k(c_i, c_{n+1}). \tag{7}$$

$\mathbf{K^{m,n}}$ is the kernel matrix m x n of the data set and the base elements and $\mathbf{K_C}$ is the kernel matrix n x n of the base elements.

This algorithm aims at selecting the best base elements to be used in the semiparametric model. We have to identify the elements of the training data whose projections most accurately represent the projected SVs.

1) Description: The procedure to be followed to obtain the new base elements, given a training set $\{\mathbf{x_r}\}$ r=1,...,S, is as follows.

- Step 0:Initialization
  We choose a subset of M patterns from the training data, $\{\mathbf{x_m}\}$ m=1,...,M. In [19][20] is shown that a random subset size of $log0.05/log0.95 = 59$ obtains an estimate that with probability 0.95 is among the best 0.05 of all estimates. In our case we choose M=64 because it is the first power of two greater than 59.
  The pattern with higher norm $(\mathbf{k_{Sm}})_\mathbf{i} = k(x_i, x_m)$ is chosen to be the first element of the base $\{\mathbf{c_i}\}$.
- Step 1: Compute kernel matrices.
  $(\mathbf{K_C})_{\mathbf{i,j}} = k(c_i, c_j)$
  $(\mathbf{K_{SC}})_{\mathbf{r,i}} = k(x_r, c_i)$
- Step 2: We choose a subset of M elements from the training data. $\{\mathbf{x_m}\}$ m=1,...,M. In our case M=64.
- Step 3: For every element of the subset $\{\mathbf{x_m}\}$. The Error Decrease is calculated.
  $(\mathbf{k_{mC}})_\mathbf{i} = k(x_m, c_j)$
  $(\mathbf{k_{Sm}})_\mathbf{i} = k(x_i, x_m)$
  $\mathbf{z} = \mathbf{K_C^{-1}} \cdot \mathbf{k_{mC}}$
  $\eta = 1 - \mathbf{z^T k_{mC} z}$
  $ED_m = \eta^{-1}||\mathbf{K_{SC}z} - \mathbf{k_{Sm}}||^2$
- Step 4: The element with the largest ED is chosen as a new base element.

2) Convergence: To obtain good base elements, the training set has been divided in groups, every group contains the elements of one of the classes of the training set.

For every group a number of L base elements has been added to the base using SGMA, in our case L is the maximum of 0.1% of the group size and 5. After that, 10% of the training set has been taken to compute the weights using IRWLS and another 10% of the training set has been taken to evaluate the accuracy. This process is repeated until the accuracy saturates.

3) Parallelization: The main computational cost (step 3) has been parallelized by distributing the evaluation of the error decreases among the available cores in the system.

In step 1, matrix updating takes negligible computational cost because the new columns and rows of the matrices have been computed in the previous error descent. It is necessary to evaluate the inverse matrix of K_C. In [19] the block matrix inversion is used, because once K_C^{-1} is known, the cost of computing K_{C+1}^{-1} is O(n^2).

$$\mathbf{K_{C+1}} = \begin{pmatrix} \mathbf{K_C} & \mathbf{k_{mC}} \\ \mathbf{k_{mC}^T} & k(C_{i+1}, C_{i+1}) \end{pmatrix} \tag{8}$$

$$\mathbf{K_{C+1}^{-1}} = \begin{pmatrix} \mathbf{K_C^{-1}} + \mathbf{K_C^{-1} k_{mC}^T k_{mC} K_C^{-1}} & -\frac{1}{k}\mathbf{K_C^{-1} k_{mC}} \\ -\frac{1}{k}\mathbf{k_{mC}^T K_C^{-1}} & 1/k \end{pmatrix} \tag{9}$$

$$k = k(C_{i+1}, C_{i+1}) - \mathbf{k_{mC}^T K_C^{-1} k_{mC}} \tag{10}$$

The resulting operations are products of matrices and vectors, as shown in [17], matrix products can be parallelized in an efficient way in an environment with shared memory distributing the rows of the matrix result to be computed among the number of cores.

B. Iterated Re-Weighted Least Squares algorithm (IRWLS)

This algorithm is responsible for calculating the optimal weights of the semiparametric SVM and has been successfully applied in the past for training SVMs [14][10][11][15]. It consists of formulating the SVM training problem as a Weighted Least Squares one and repeating iteratively until convergence the weights updating and LS solving.

1) Algorithm: Assuming we have P training data and R base elements selected using SGMA, the algorithm is as follows.

- Step 0: Initialization

$$\begin{aligned} & a_i = 1, \forall i = 1, ..., P \\ & (\mathbf{K_{SC}})_{\mathbf{i,j}} = k(x_i, c_j); i = 1, ..., P; j = 1, ..., R \\ & (\mathbf{K_C})_{\mathbf{i,j}} = k(c_i, c_j); i = 1, ..., R; j = 1, ..., R \\ & (\mathbf{D_a})_\mathbf{i} = a_i; i = 1, ..., P \\ & \mathbf{1} = [1, ..., 1]^T \\ & \mathbf{y} = [y_1, ..., y_p]^T \end{aligned} \tag{11}$$

- Step 1: Obtain optimal weights and bias

$$\mathbf{K_1} = \begin{pmatrix} \mathbf{K_{SC}^T D_a K_{SC}} + \mathbf{K_C} & \mathbf{K_{SC}^T D_a 1} \\ \mathbf{1^T D_a K_{SC}} & \mathbf{1^T D_a 1} \end{pmatrix} \tag{12}$$

$$\mathbf{k_2} = \begin{pmatrix} \mathbf{K_{SC}^T D_a y} \\ \mathbf{1 D_a y} \end{pmatrix} \tag{13}$$

$$\begin{pmatrix} \boldsymbol{\beta} \\ b \end{pmatrix} = (\mathbf{K_1^{-1} k_2}) \tag{14}$$

- Step 2: Compute errors

$$\mathbf{o}(x_i) = \sum_{r=1}^{R} \beta_i k(x_i, c_r) \tag{15}$$

$$e_i = y_i - o(x_i) \tag{16}$$

- Step 3: Update Weighting values

$$a_i = \begin{cases} 0 & \text{if} \quad e_i y_i < 0 \\ M & \text{if} \quad 0 < e_i y_i < \frac{C}{M}; M = 10^9 \\ \frac{c}{e_i y_i} & \text{if} \quad e_i y_i > \frac{C}{M} \end{cases} \tag{17}$$

- Step 4: Evaluate convergence

$$\begin{cases} ||\boldsymbol{\beta}(k+1) - \boldsymbol{\beta}(k)||_2 + \\ +||b(k+1) - b(k)||_2 <= 10^{-3} \textit{Stop the algorithm} \\ \\ ||\boldsymbol{\beta}(k+1) - \boldsymbol{\beta}(k)||_2 + \\ +||b(k+1) - b(k)||_2 > 10^{-3} \textit{Go to step } 1 \end{cases} \tag{18}$$

2) Parallelization: Step 1 has the highest computational cost. This cost is O(R^2P) with P >> R, the parallelization to obtain $\mathbf{K_1}$ as well as $\mathbf{k_2}$ has been done by dividing among the cores the different rows to be obtained in both matrices.

To obtain a completely parallel code, it is necessary to implement a parallel inversion of $\mathbf{K_1}$. Traditional methods to invert matrices are sequential, so it is impossible their parallelization. To perform the matrix inversion, K_1 has been splitted into four submatrices (quadtree) and we have used the block matrix pseudoinversion because it can be divided into two subprocesses.

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} (A-BD^{-1}C)^{-1} & -A^{-1}B(D-CA^{-1}B)^{-1} \\ -D^{-1}C(A-BD^{-1}C)^{-1} & (D-CA^{-1}B)^{-1} \end{pmatrix} \tag{19}$$

The first process obtains $(A - BD^{-1}C)^{-1}$ and then $-D^{-1}C(A - BD^{-1}C)_{-1}$ and the second process obtains $(D - CA^{-1}B)^{-1}$ and $-A^{-1}B(D - CA^{-1}B)^{-1}$. Each process has operations of multiplication and matrix inversion, which can be further divided recursively using smaller quadtrees until all the cores in the system are used.

Computational cost of matrix inversion is R^3. Each subtask has to do two inversions and three products with matrices, 5 operations with computational cost $(R/2)^3$. That approximately represents 5/8 of the complete inversion, more than a half. This efficiency loss has been partially solved using LU inversion and back substitution, because it is possible to implement operations like $A^{-1}B$ with the same computational cost than A^{-1}[16]. LU inversion is not parallelizable, so we can only use it in the last recursion.

The product of $\mathbf{K_1}^{-1}$ and $\mathbf{k_2}$ has been parallelized by distributing the elements of the resulting vector to be obtained among the available cores.

Steps 2 and 3 have also been parallelized, dividing the training data set among the cores to evaluate their errors and weighting values.

III. Experiments

This algorithm has been implemented in C using the programming interface OpenMP [11] to parallelize its execution. It has been evaluated against the unreduced machines obtained with the library LIBSVM 2.91, because this library is commonly used and both are written in C. Both algorithms are executed on a Sun X4150 server with eight cores (2 Intel Xeon E5450 processors, each one with 4 cores of 3 GHz).

A. Adult database

The first data set used is the UCI [1] Adult data set. It contains a total of 32561 data patterns each one with 123 binary attributes. A Gaussian kernel was used with parameters γ = 0.5 and C = 100, which are not necessarily those who obtain a better classification error, but the purpose of the experiment is to evaluate the run times.

Table I shows the resulting machine sizes, the accuracy and the run time in milliseconds for LIBSVM and both parts (SGMA, IRWLS) of the developed algorithm when 1, 2, 4 and 8 cores of the server are used. For a single processor, the run time of LIBSVM and PS-SVM have the same order of magnitude. The classification error is quite similar too, having PS-SVM a slightly better accuracy. The machine size in the case of PS-SVM is two orders of magnitude lower, which is the main advantage of reduced-set SVMs against full SVMs. As it can be appreciated, doubling the number of cores reduces the run time about a half. To evaluate the parallelization quality two criteria have been used: speedup and efficiency.

TABLE I
Run time, machine size and accuracy of experiment 1

Algorithm	LibSVM	PSSVM 1 Core	PSSVM 2 Cores	PSSVM 4 Cores	PSSVM 8 Cores
SGMA(ms)		210048	105020	60158	31657
IRWLS(ms)		303768	151890	79100	40390
Run time(ms)	542564	513816	256910	139258	71047
Machine size	19059	126	126	126	126
Accuracy(%)	82.69	82.87	82.87	82.87	82.87

$$Speedup = \frac{Serial\ Run\ Time}{Parallel\ Run\ Time} \tag{20}$$

$$Efficiency = \frac{Speedup}{number\ of\ cores} \tag{21}$$

Figure 1a and 1b show, respectively, the speedup and efficiency of the implemented PS-SVM algorithm for 2, 4 and 8 cores. A measurement of the run time for every subtask has been taken too to evaluate which would be the speedup and efficiency in an "ideal" environment with no losses of efficiency due to the launching of the different processes on the different cores, this would be the maximum level that could be reached.

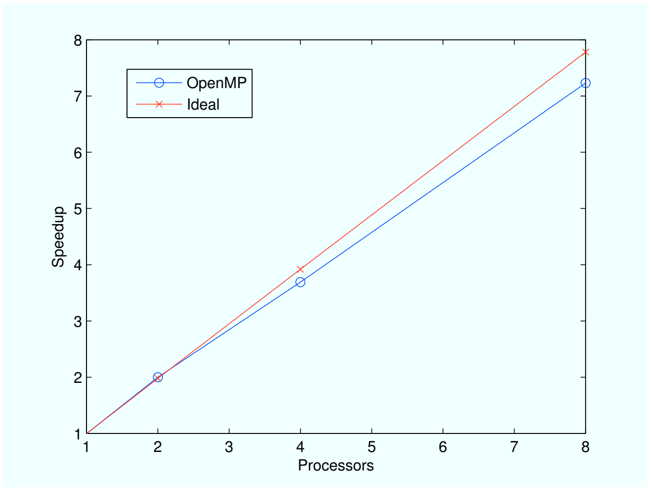
As can be seen, the efficiency in an ideal environment is close to one because almost the entire algorithm has been parallelized, the results of the implementation with OpenMP have an efficiency close to 0.9 for 8 cores, which means that launching the processes on the different cores represents a 10% loss of efficiency.

B. Web database

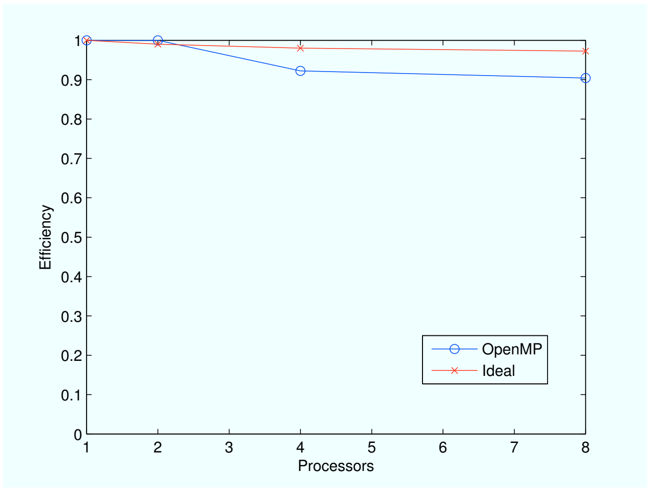
For the second experiment we have used the web dataset. There are a total of 24692 data in the training set and 300 attributes, which consist of a bag of words from web pages. Gaussian Kernel with a value γ = 7.8125 and C = 64 has been used for both LIBSVM and PSSVM.

Table II shows the resulting machine size, percentage of correct classification, run time in milliseconds for LIBSVM and both parts of the developed algorithms when 1, 2, 4 and 8 cores are used.

In this case the run time of IRWLS is 3 times smaller than LIBSVM and the machine size is again 2 orders of



(a) Speedup



(b) Efficiency

Fig. 1. Speedup and efficiency in experiment 1

magnitude smaller. The classification error is similar in both algorithms.

The speedup and efficiency of this algorithm are shown in figure 2a and 2b respectively (OpenMP in the graphs). The run time of every subtask using one core has been taken too and using it has been calculated the speedup and efficiency in an environment with no losses of efficiency launching the processes on different cores, it appears as “Ideal” in the figures, this is the maximum level that could be reached with a parallel implementation.

The efficiency in an ideal environment is still close to one because almost the entire algorithm has been parallelized, the results of the OpenMP implementation are worse than int the previous experiment. The reason is because tasks are faster and the effects of distributing processes on the cores have more influence in the results.

C. *USPS database*

The USPS data set contains 7291 handwritten digits for training and 2007 for testing, the number of attributes is 784 (each digit is a 27x28 image). In this experiment we are going to classify odd digits vs even digits. The parameters that we have chosen are γ =1/256 and C = 10.

TABLE II

RUN TIME, MACHINE SIZE AND ACCURACY OF EXPERIMENT 2

Algorithm	LibSVM	PSSVM 1 Core	PSSVM 2 Cores	PSSVM 4 Cores	PSSVM 8 Cores
SGMA(ms)		131186	69584	38828	21686
IRWLS(ms)		42307	22344	11637	6040
Run time(ms)	566994	173493	105334	57447	27726
Machine size	16781	85	85	85	85
Accuracy(%)	97.57	97.67	97.67	97.67	97.67

TABLE III

RUN TIME, MACHINE SIZE AND ACCURACY OF EXPERIMENT 3

Algorithm	LibSVM	PSSVM 1 Core	PSSVM 2 Cores	PSSVM 4 Cores	PSSVM 8 Cores
SGMA(ms)		20229	10517	5718	3706
IRWLS(ms)		84206	48114	26254	13541
Run time(ms)	9351	104436	58631	31972	17247
Machine size	684	200	200	200	200
Accuracy(%)	97.06	96.31	96.31	96.31	96.31

In table III the results (accuracy, machine size and run time) are shown for LIBSVM and both parts of the developed algorithm. In this experiment the accuracy is similar but LIBSVM have better percentage of correct classification.

Now, the run time of LIBSVM is better than the run time of PS-SVM because this problem has a resolution with few support vectors, and therefore LIBSVM is highly efficient.

The speedup and efficiency are shown in figure 3a and 3b respectively. The reason why the efficiency here is smaller than in others experiments, is because this is the smallest data set that we have used and the tasks to divide are smaller.

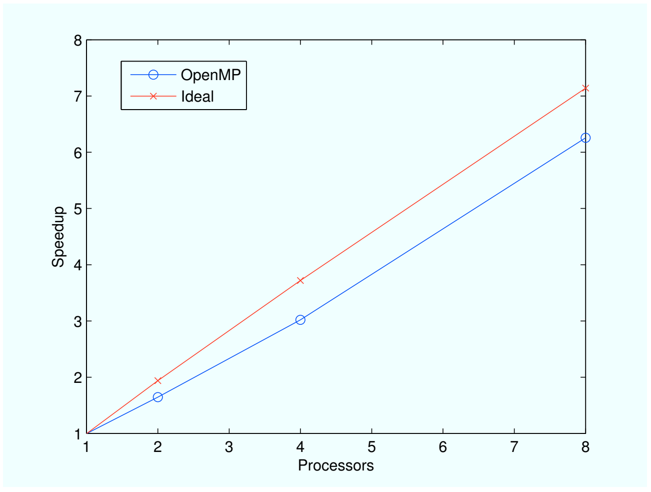
Figure 4 shows the efficiency that can be reached using 1-32 cores in an ideal environment. As can be seen, its possible a performance improvement of the algorithm increasing the number of cores.

D. *MNIST database*

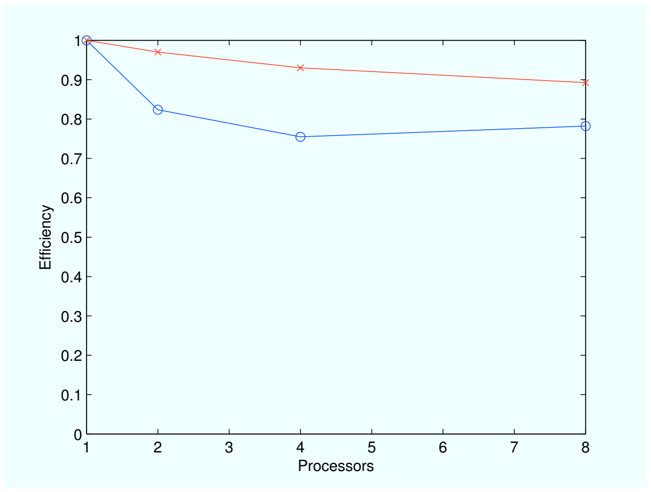
For the last experiment we have used the MNIST hand-written data set. It ss composed of 60000 samples for training and 10000 for testing. The number of features is 576. We have chosen Gaussian Kernel and the parameters are γ = 0.125 and C = 10. This is a ten-class problem, so we have evaluated ten SVM classifiers ”one versus all”.

Table IV shows the accuracy and machine size for every classifier using LIBSVM and PS-SVM. The accuracy is comparable. The machine size in the case of PS-SVM is two orders of magnitude lower.

Table V shows the run time of LIBSVM and both parts of the algorithm for each classifier. Here PS-SVM is faster and has the advantage that the base elements can be reused for all the classifiers, and hence need to be computed only once.

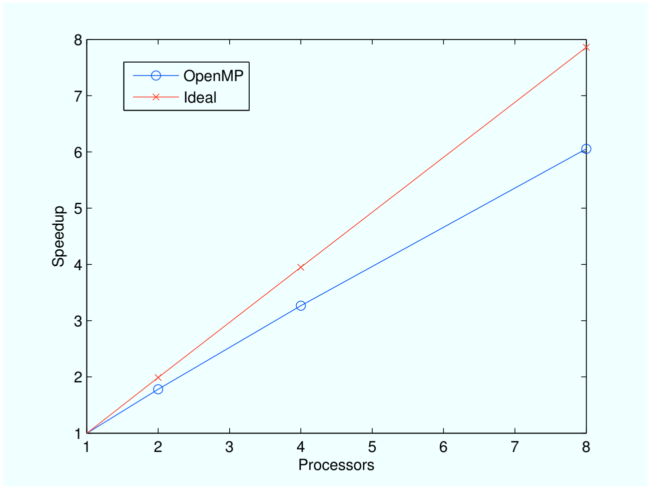


(a) Speedup

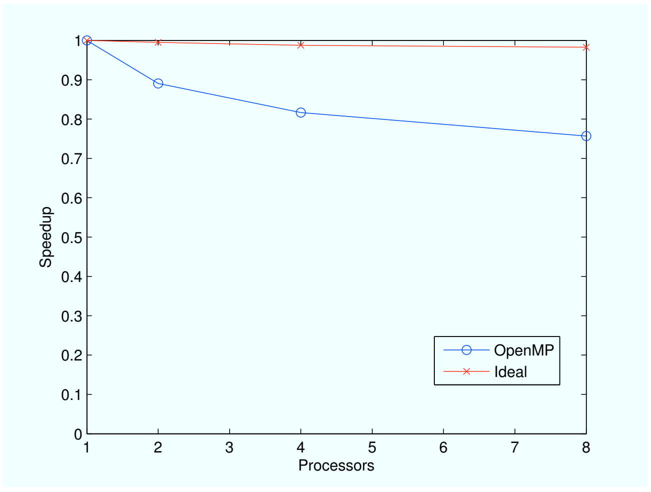


(b) Efficiency

Fig. 2. Speedup and efficiency in experiment 2



(a) Speedup



(b) Efficiency

Fig. 3. Speedup and efficiency in experiment 3

Figure 5a and 5b show, the average speedup and efficiency of the classifiers ($\pm 2\sigma$ lines are also plotted, σ being the standard deviation).

IV. Conclusions

This paper proposes a parallel training algorithm for semiparametric support vector machines (PS-SVM), the main contribution is the use of quadtrees for the parallelization of matrix inversion, dividing the tasks among different processors.

It has been shown the efficiency of using multiple cores on a machine because it allows increasing the speedup with a proportion close to the number of used cores.

It's possible to see how the efficiency decreases as the number of cores increases because with smaller subtasks, more time is wasted launching them so the system is less efficient.

Amdhal law says what the speedup is equal to 1/((1-P)+P/N), where P is the proportion of a program that can be made parallel and N the number of cores. The upper

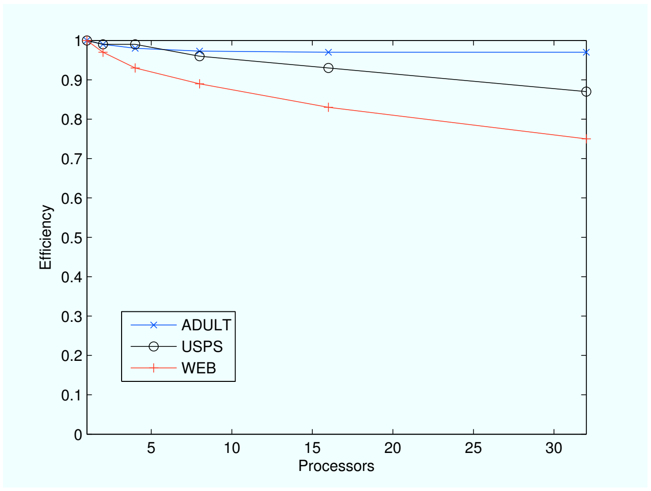
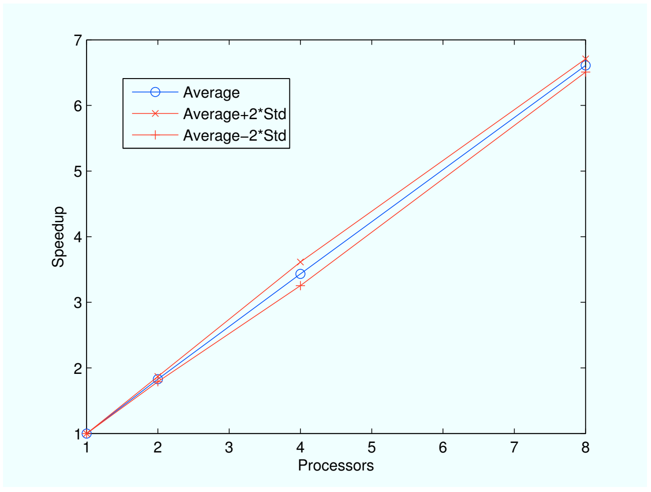
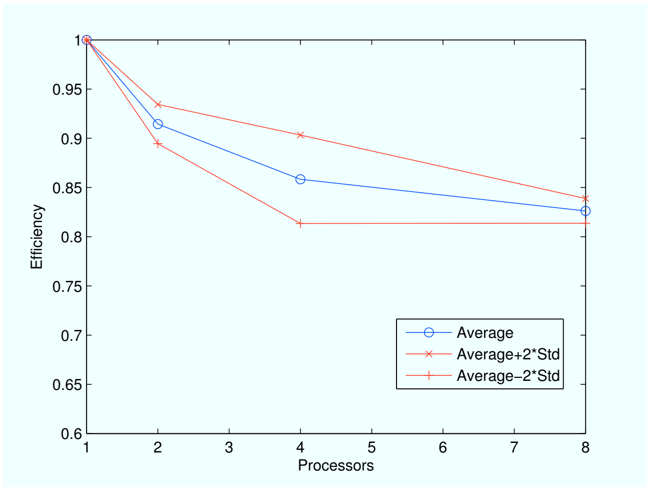


Fig. 4. Efficiency in an ideal environment



(a) Speedup



(b) Efficiency

Fig. 5. Speedup, efficiency and its standard deviations in experiment 4

TABLE IV

MACHINE SIZE AND ACCURACY OF EXPERIMENT 4

CLASS	LIBSVM Size	PSSVM Size	LIBSVM Accuracy(%)	PSSVM Accuracy(%)
0	40361	378	97.01	97.40
1	35282	378	99.74	99.49
2	40890	378	94.47	94.59
3	41818	378	96.20	96.55
4	41816	378	97.12	97.21
5	41292	378	96.04	94.29
6	40130	378	97.28	96.31
7	41236	378	97.98	97.54
8	42186	378	95.12	95.33
9	42672	378	97.80	95.82
Average	40768	378	96.88	96.46

TABLE V

RUN TIME IN SECONDS OF EXPERIMENT 4

CLASS	LibSVM	PSSVM (1 Core)	PSSVM (2 Cores)	PSSVM (4 Cores)	PSSVM (8 Cores)
0	17551	10239	5637	2813	1553
1	8532	10234	5620	2958	1559
2	16820	10342	5641	2970	1565
3	16450	10244	5655	2978	1549
4	17398	10212	5648	2940	1566
5	17756	10346	5525	3031	1549
6	15760	10332	5595	3077	1554
7	16166	10242	5640	3070	1561
8	17121	10361	5687	3090	1558
9	17225	10361	5625	3068	1557
Average	16078	10291	5627	2999	1557

boundary when the number of cores tends to infinity is 1/(1-P), this effect can be observed on the results of the experiments, the slope of the speedup decreases slightly by increasing the number of processors.

As future research lines, we propose to apply these parallelization techniques to other machine learning algorithms based on kernels such as Gaussian Processes, which represent a bigger scale challenge because they don't naturally lead to sparse solutions as SVMs.

ACKNOWLEDGMENT

This work has been funded by the Spain government under grant TEC2008/02473.

REFERENCES

[1] A. Asuncion and D. Newman. UCI machine learning repository, 2007. http://archive.ics.uci.edu/ml/

[2] C.J.C Burges & B. Scholkopf, "Improving the accuracy and speed of support vector learning machines", In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pp. 375381. MIT Press, Cambridge, CA, 1997.

[3] L.J. Cao, SS Keerthi, C.J. Ong, JQ Zhang, U. Periyathamby, X.J. Fu & HP Lee, "Parallel sequential minimal optimization for the training of support vector machines", *IEEE Trans.Neural Networks*, 2006, 17, 4, 1039

[4] R. Collobert, S. Bengio & Y. Bengio, "A parallel mixture of SVMs for very large scale problems", *Neural Comput.*, 2002, 14, 5, 1105-1114, MIT Press

[5] J. X. Dong, A. Krzyzak, C. Y. Suen, "A fast Parallel Optimization for Training Support Vector Machine", *Proc. of 3rd Int. Conf. Machine Learning and Data Mining*, P. Perner and A. Rosenfeld (Eds.) Springer Lecture Notes in Artificial Intelligence (LNAI 2734), pp. 96-105, Leipzig, Germany, July 5-7, 2003

[6] S. Herrero-Lopez, J.R. Williams & A. Sanchez. "Parallel multiclass classification using SVMs on GPUs." 2010, 2-11, ACM

[7] J.T. Kwok, I.W. Tsang, "The pre-image problem in kernel methods", *IEEE Trans. Neural Networks, Vol. 15*, No 6, pp. 1517-1525, 2004.

[8] A. Navia-Vázquez, F. Pérez-Cruz, and A. Artés-Rodriguez, "Weighted least squares training of support vector classifiers leading to compact and adaptive schemes", *IEEE Trans. Neural Netw., vol. 12*, no. 5, pp. 1047 - 1059, 2001.

[9] A. Navia-Vázquez, (2007). "Support vector perception", *Neurcomputing, Vol. 70*, pp. 1089-1095, 2007.

[10] A. Navia-Vázquez, "Compact Multiclass Support Vector Machines", *Neurocomputing, Vol. 71*, No 1-3, pp. 400-405, 2007.

[11] A. Navia-Vázquez, D. Gutierrez-Gonzalez, E. Parrado-Hernandez & JJ. Navarro-Abellan. "Distributed support vector machines". *IEEE Trans.Neural Networks*, 2006, 17, 4, 1091-1097

[12] OpenMP C and C++ application program interface. Available from http://www.openmp.org

[13] E. Osuna & F. Girosi, "Reducing the run-time complexity in support vector regression". In B. Schólkopf, C.J.C. Burges and A.J. Smola, eds., *Advances in Kernel Methods Support Vector Learning*, pp. 271-284, MIT Press, Cambridge MA, 1999.

[14] E. Parrado-Hernández, J. Arenas-García, I. Mora-Jimenez, A.R. Figueiras-Vidal, and A. Navia-Vázquez, "Growing Support Vector Classifiers with Controlled Complexity". *Pattern Recognition, Vol. 36*, no. 7, pp. 1479-1488,2003.

[15] F. Pérez-Cruz, P. Alarcón-Diana, A. Návia-Vázquez, A. Artés-Rodriguez, "Fast training of support vector classifiers", *Advances in Neural Information Processing Systems, vol 13*, 2000, pp. 734-740.

[16] PRESS, W.H., TEUKOLSKY, S.A., VETTERING, W.T. & FLANNERY, B.P.: *Numerical recipes in C*. (Cambridge University Press, Cambridge, 1994, 2nd edn., pp. 48-49)

[17] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski & C. Kozyrakis. "Evaluating MapReduce for multi-core and multiprocessor systems". *In HPCA 07: Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, February 2007.

[18] B. Schólkopf, P. Knirsch, A. Smola & C. Burges, "Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces", *Neural Computation, vol. 10*, pp. 1299-1319,1998.

[19] B. Schólkopf & A. Smola, *"Learning with kernels"*. Cambridge, MA: MIT Press, 2002.

[20] A.J. Smola & B. Schólkopf, "Sparse Greedy matrix approximation for machine learning". In P. Langley, ed., *Proc. of the 17th International Conference on Machine Learning*, pp. 911-918, Morgan Kaufman, San Francisco CA, 2000.

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