Parallel Semiparametric Support Vector Machines [1]

Roberto Díaz-Morales, Harold Y. Molina-Bulla
and Ángel Navia-Vázquez
{rdiazm,h.molina,navia}@tsc.uc3m.es

University Carlos III de Madrid
Deparment of Signal Theory and Comunications

August, 2011

[1] This work has been supported by CICYT grant TEC2008/02473.

Contenido

Support Vector Machines

- Kernel methods have become very popular in the machine learning.
- Support Vector Machines (SVMs)are considered the "state-of-art" to solve classification problems:
  - Performance working with high dimensional data.
  - Ability to adjust the machine size once its hyperparameters are set.
  - Classifier size usually results very high $->$ Computational cost.
- Some methods propose to reduce the machine size growing up a semiparametric model.
  - The complexity is kept under control.
  - Good ratio of complexity and performance.

Parallelization

- In recent years the number of cores in computers has increased considerably.
- New programming interfaces, as OpenMP, have emerged that supports multiplatform shared memory multiprocessing programming.
- New research lines to adapt classical techniques of machine learning to a parallel scenario.
- In this work we derive a new method to train semiparametric SVMs based on SGMA and Iterated Re-Weighted Least Squares (IRWLS).

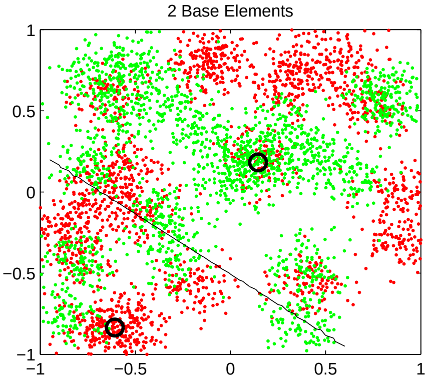
SGMA

- A kernel evaluation $k(x_i, .)$ can be approximated as a linear combination of other kernels $k(x_1, .)$,...,$k(x_n, .)$ with n base elements.
- SGMA identifies the elements of the training set $\{x_1,...,x_m\}$ whose projections represent accurately the support vectors.
- The approximation error once the weights α_{ij} are chosen is then: $Err(\alpha) = trK - \sum_{i=1}^{m} \sum_{i=1}^{n} \alpha_{i,j} K_{i,j}$
- Adding a new base element c_{n+1} the new error can be exoressed as a function of the previous error:[2] $Err(\alpha^{m,n+1}) = Err(\alpha^{m,n}) - \underline{\eta^{-1} ||\mathbf{K^{m,n}z} - \mathbf{k_{Sm}}||^2}$
- This algorithm choose iteratively a new base element comparing de error descendant of a group of candidates.

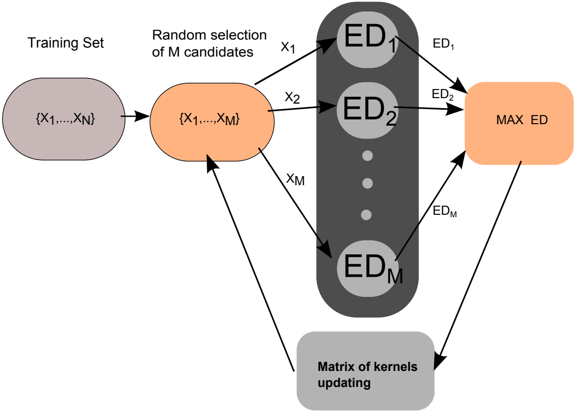
[2] $\mathbf{z} = \mathbf{K_C^{-1}} \cdot \mathbf{k_{mC}}$ and $\eta = 1 - \mathbf{z^T K_{mC} z}$

SGMA-Simulation

Boundary regions in function of the number of base elements.



SGMA - Parallelization



SGMA - Parallelization

- Products of matrices and vector, easily parallelizable distributing the rows of the matrix result among the number of cores.

- For the matrix updating $\mathbf{K_C^{-1}}$, the block matrix inversion is used:

$$\mathbf{K_{C+1}} = \begin{pmatrix} \mathbf{K_C} & \mathbf{k_{mC}} \\ \mathbf{k_{mC}^T} & 1 \end{pmatrix}$$

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

$$k = 1 - \mathbf{k_{mC}^T} \mathbf{K_C^{-1}} \mathbf{k_{mC}}$$

IRWLS-Algoritmo

- This algorithm calculate the optimal weights of the semiparametric SVM.
- It consists of formulating the SVM training problem as a Weighted Least Squares one and repeating iteratively until the convergence the weights updating and LS solving.
- We have P trining data and R base elements selected using SGMA.
- Step 0: Initialization
  $a_i = 1, \forall$ 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
  $\mathbf{1} = [1, ..., 1]^T$
  $\mathbf{y} = [y_1, ..., y_p]^T$

IRWLS-Algoritmo

- Step 1: Obtain optimal weights and bias

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

$$\mathbf{K_2} = \begin{pmatrix} \mathbf{K_{SC}^T D_a y} \\ \mathbf{1 D_a y} \end{pmatrix}$$

$$\begin{pmatrix} \beta \\ b \end{pmatrix} = (K_1^{-1} K_2)$$

- Step 2: Compute errors
  $\mathbf{o}(x_i) = \sum_{r=1}^{R} \beta_i k(x_i, c_r)$
  $e_i = y_i - o(x_i)$

IRWLS-Algoritmo

- Step 3: Update Weighting values

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

- Step 4: Evaluate convergence

$$\begin{cases} ||\beta(\mathbf{k+1}) - \beta(\mathbf{k})||_2 + ||\mathbf{b}(\mathbf{k+1}) - \mathbf{b}(\mathbf{k})||_2 & <= 10^{-3} \quad \textit{Stop} \\ ||\beta(\mathbf{k+1}) - \beta(\mathbf{k})||_2 + ||\mathbf{b}(\mathbf{k+1}) - \mathbf{b}(\mathbf{k})||_2 & > 10^{-3} \quad \textit{Go to step } 1 \end{cases}$$

IRWLS-Parallelization

- Step 1: This step has the highest computational cost $O(R^2P)$, $P >> R$. To obtain $\mathbf{K_1}$ and $\mathbf{K_2}$ the different rows to be obtained were divided among the cores.
  The parallel inversion of $\mathbf{K_1}$ has been done with the block matrix pseudoinversion:

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

  Computational cost of matrix inversion is R^3, each subtask do 5 operations of cost $(R/2)^3$, that represents 5/8 of the complete inversion. This efficiency loss has been partially solved using LU inversion and back substitution, it is possible to implement operations like $A^{-1}B$ with the same computational cost than A^{-1}.

- Steps 2 y 3: These steps have also been parallelized dividing the training data set among the cores to evaluate their errors and weighting values.

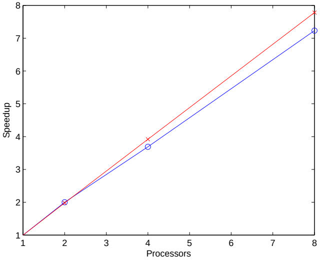
Experiments

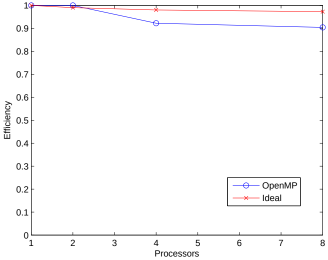
- This algorithm has been implemented in C using the programming interface OpenMP to parallelize its execution.
- It has been evaluated against the unreduced machines obtained with the library LIBSVM 2.91.
- Both algorithms are executed on a Sun X4150 server with eight cores.
- To evaluate the parallelization quality two criteria have been used:
  $\text{Speedup} = \frac{Serial\ Run\ Time}{Parallel\ Run\ Time}$
  $\text{Efficiency} = \frac{Speedup}{Number\ of\ cores}$

Experiment 1

- UCI Adult data set: 32561 patterns with 123 binary attributes. Gaussian kernel with $\gamma = 0{,}5$ and $C = 100$.

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

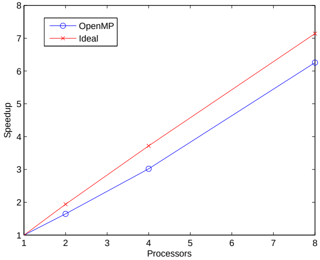


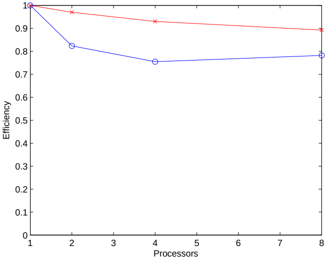


Experiment 2

- Web data set: 24692 patterns with 300 attributes. Gaussian kernel with $\gamma = 7{,}8125$ and $C = 64$.

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

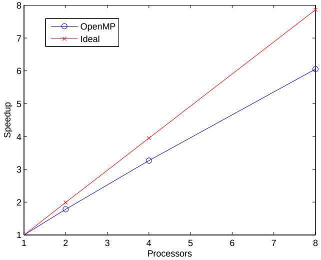


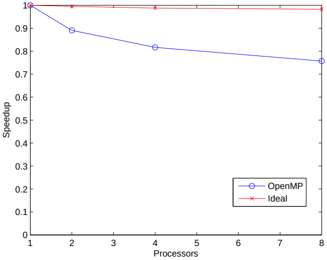


Experiment 3

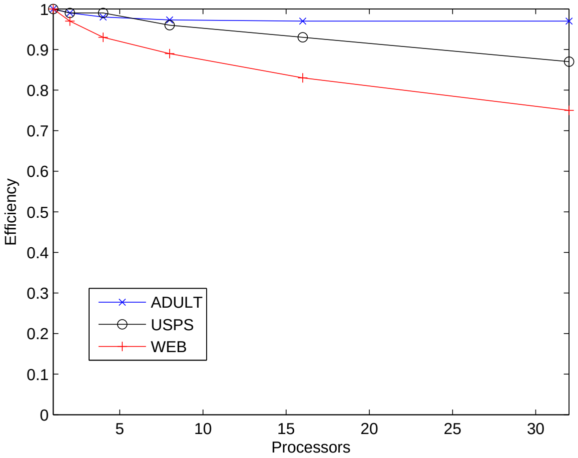
- USPS data set: 7291 handwritten digits for training and 2007 for testing with 784 attributes(each digit is a 27x28 image). In this experiment we have classified odd digits vs even digits. Gaussian kernel with γ =1/256 and C = 10.

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





Ideal environment 1-32 processors



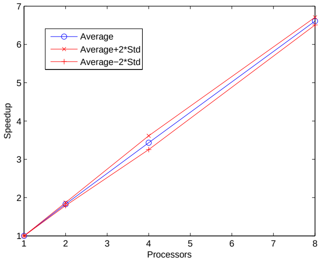
Experiment 4

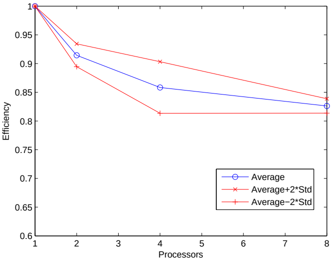
- MNIST data set: 6000 patterns for training and 1000 for testing with 576 attributes. Gaussian kernel with $\gamma = 0{,}125$ and $C = 10$. Ten classifiers one-versus-all.

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

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

Experiment 4





Conclusions

- A parallel training algorithm for semiparametric support vector machines (PSSVM) has been proposed. Using quadtrees for the parallelization of matrix inversion, dividing the tasks among different processors.
- The efficiency of using muttiples cores on a machine because it allows a speedup close to the number of cores.
- Amdhal law says that 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. This effect can be observed on the results, the slope of speedup decreases increasing the number of processors.
- As future research lines we propose to apply these parallelization techniques to other machine learning algorithms based on kernel such as Gaussian Processes, which represent a bigger scale schallenge because they don't naturally lead to sparse solutions as SVMs.

References

B. Schólkopf and A. Smola
Learning with kernels
Cambridge, MA: MIT Press, 2002.

V. Vapnik
The Nature of Statistical Learning Theory.
New York: Springer-Verlag, 1995.

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.

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

- Thank you for your attention.