Journal of Machine Learning Research 1 (2006) 1-11 Submitted 1/06; Revised 7/06; Published 10/00

Comments on the "Core Vector Machines: Fast SVM Training on Very Large Data Sets"

Gaëlle Loosli GAELLE@LOOSLI.FR
LITIS, INSA de Rouen
Avenue de l'Université
76801 Saint-Etienne du Rouvray
France

Stéphane Canu STEPHANE.CANU@INSA-ROUEN.FR
LITIS, INSA de Rouen
Avenue de l'Université
76801 Saint-Etienne du Rouvray
France

Editor: Nello Cristianini

Abstract

In a recently published paper in JMLR, Tsang et al. (2005) present an algorithm for SVM called Core Vector Machines (CVM) and illustrate its performances through comparisons with other SVM solvers. After reading the CVM paper we were surprised by some of the reported results. In order to clarify the matter, we decided to reproduce some of the experiments. It turns out that to some extent, our results contradict those reported. Reasons of these different behaviors are given through the analysis of the stopping criterion.

Keywords: SVM, CVM, large scale, KKT gap, stopping condition, stopping criteria.

Introduction

In a recently published paper in JMLR, Tsang et al. (2005) present an algorithm for SVM, called CVM. In this paper, some illustration of the CVM performances, compared to others solvers are shown. We have been interested in reproducing their results concerning the checkers problem because we knew that SimpleSVM (Vishwanathan et al., 2003) could handle such a problem more efficiently than reported in the paper. Tsang et al. (2005, Figure 3 page 379) show that CVM's training time is independent of the sample size and obtains similar accuracies to the other SVM solvers. We discuss those results through the analysis of the stopping criteria of the different solvers and the effects of hyper-parameters.

Indeed in SVMs, there are hyper-parameters: the slack trade-off (C) and the stopping tolerance (ϵ). For RBF kernels, the bandwidth (γ) is also an hyper-parameter and it can be estimated via the distance between points from opposite classes (Tsang et al., 2005, page 376). It can also be fixed using cross-validation. The slack trade-off is used to permit points to be misclassified. A small value for C allows many points to be misclassified by the solution while large value or infinite value forces the solution to classify well all the points (hard-margins). This hyper-parameter can be chosen with cross-validation or using knowledge on the the problem. Those two hyper-parameters are well-known and more details can be found in literature (Schölkopf and Smola, 2002). The

stopping tolerance is most often left as a default value in the solvers. It corresponds to the precision required before stopping the algorithm and is strongly linked to the implementation. The value it takes is close to zero and we point out in this paper that it is not the value by itself that is important but the stopping criteria in which it is involved (section 2).

Coming back to the comparison of solvers, our first experiment (section 1) shows how different sets of hyper-parameters produce different behaviors for both CVM and SimpleSVM. We also give results with libSVM (Chang and Lin, 2001a) for the sake of comparison since SMO (Platt, 1999) is a standard in SVM resolution. Our results indicates that the choice of the hyper-parameters for CVM does not only influence the training time but also greatly the performance (more than for the other methods).

Section 2 aims at understanding what influences the most the CVM accuracy. We show that CVM uses a stopping criterion which may induce complex tuning and unexpected results. We also explore the behavior of CVM when C varies compared to libSVM and SimpleSVM.

In section 3, our second experiment points out that the choice of the magnitude of the hyper-parameter C (which behaves as a regulator) is critical, more than the bandwidth for this problem. It is interesting to notice that each method has *modes* that are favorable and *modes* that are not. The last experiments enlightens those different modes.

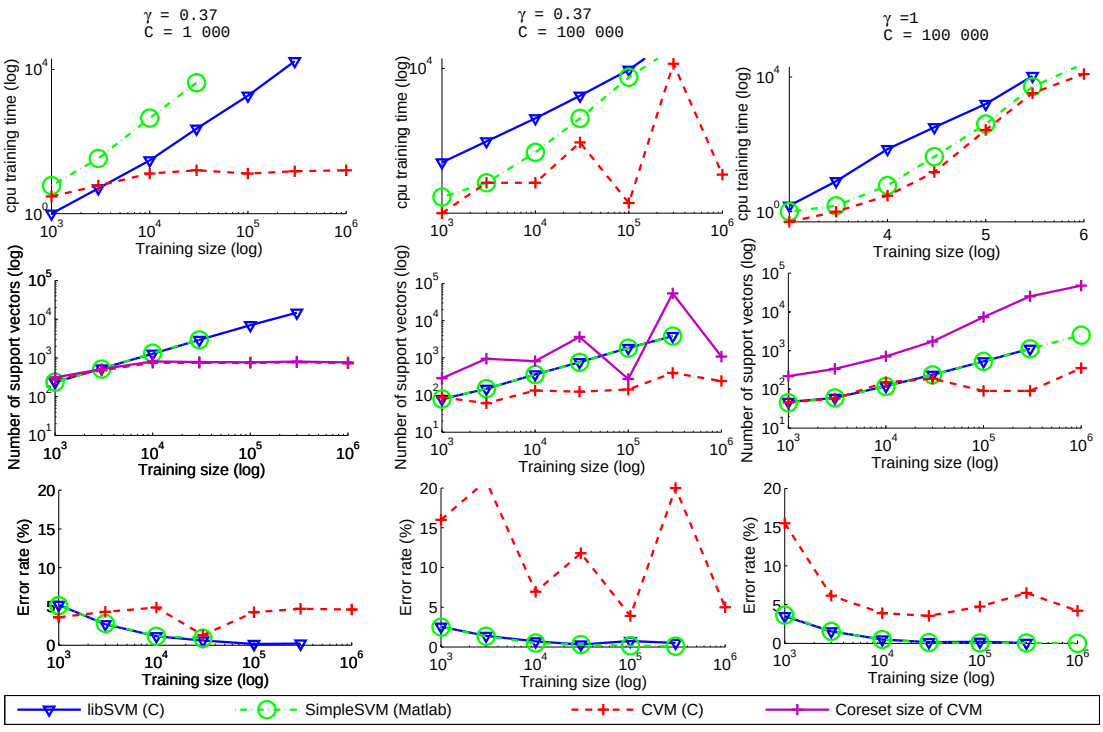

Figure 1: Comparison between CVM, libSVM and SimpleSVM on the checkers problem. For all figures, $\triangledown$ is for libSVM, $\circ$ for SimpleSVM and $+$ for CVM. The three figures on the first column show the results while trying to reproduce figure 3 (page 379) from the paper ($C = 1000$, $\gamma = 0.37$, *cf.* page 376, section 6 and page 377, section 6.1). The second column shows results with a larger C ($C = 100000$ and $\gamma = 0.37$) and the last shows results for $C = 100000$ and $\beta = 1$. Missing results are due to early termination because of not enough memory and/or the too long training time.

About the reproducibility All experiments presented here can be reproduced with codes and dataset that are all published.

- **CVM (Tsang et al., 2005)** Version 1.1 (in C) downloaded from `http://www.cs.ust.hk/~ivor/cvm.html` for Linux platform on the 3^{rd} of April 2006.

- **LibSVM (Chang and Lin, 2001a)** We used the release contained in CVM (here we should mention that a new version of libSVM exists (Fan et al., 2005), which would probably give some better results).

- **SimpleSVM (Vishwanathan et al., 2003; Loosli, 2004)** We used version 2.3 available at `http://asi.insa-rouen.fr/~gloosli/coreVSsimple.html`, written in Matlab.

- **Data** We have published the used datasets at `http://asi.insa-rouen.fr/~gloosli/coreVSsimple.html`, in both formats for libSVM and SimpleSVM.

Note that all results presented here are obtained using a stopping tolerance $\epsilon = 10^{-6}$ as proposed in the CVM paper. Moreover, from now and for the rest of the paper, SimpleSVM will refer to the specific version implemented in the toolbox, as well as SMO will refer to the libSVM implementation.

1. Alternative parameters for the checkers

This section aims at verifying that SimpleSVM can treat one million points in problems like the checkers with adapted hyper-parameters. On the way, we also see that this leads to better performance than the one presented on figure 3 page 379 of Tsang et al. (2005) and not only for SimpleSVM.

1.1 Original settings

We used the parameters described in their paper ($\gamma = 0.37$ and $C = 1000$), γ being fixed to this value so the results can easily be reproduced (the setting of the paper relies on the average distance of couples of points on the training set). Note that $\gamma = 1/\beta$ corresponds to the hyper-parameters directly given to the toolboxes - $k(x_i, x_j) = \texttt{exp}\big(-\gamma(x_i - x_j)^2\big)$ and that β is the notation used in the studied paper. Results are presented on figure 1, first column.

1.2 Others settings

Our motivation is that the checkers is a separable problem. First, this should reduce the number of support vectors and second, it should be possible to have a zero error in test. Thus we used in a second time some other hyper-parameters : the couple $\gamma = 0.37$, $C = 100000$ on the one hand and $\gamma = 1$ and $C = 100000$ on the second hand (found by cross-validation). Examples of results are presented on figure 1, second and third columns [1].

1. It turns out that they are representative of the behaviors of the algorithms on this problem for most of randomly drawn training sets

1.3 Results

- For the given settings, we can see similar behaviors as the one presented in the paper concerning training size, performance and speed. In particular, due to the large number of support vectors required for this setting (graph (b), figure 1), SimpleSVM cannot be run over 30,000 data points,
- the possibility to run SimpleSVM or CVM until 1 million points depends on the settings,
- both SimpleSVM and libSVM achieve a 0 testing error (which means to us that the settings are more adapted) while CVM shows a really unstable behavior regarding testing error,
- for CVM, training time grows very fast for large values of C, due to huge intermediate number of active points (see section 3), even if it still gives a sparse solution in the end,
- finally, the training error shows that CVM does not converge towards the solution for all hyper-parameters.

2. Study of CVM

Because figure 1 shows very different behaviors of the algorithms, we focus in this part on the stopping criteria and on the role played by the different hyper-parameters. To do so we first compare CVM and SimpleSVM algorithms and then we point out that stopping criteria are different and may explain the results reported in figure 1. We illustrate our claims on figure 2 which shows that the magnitude of C is the key hyper-parameter to explore the different behaviors of the algorithms.

2.1 Algorithm

The algorithm presented in CVM uses MEB (Minimum Enclosing Balls) but can also be closely related to decomposition methods (such as SimpleSVM). Both methods consist in two alternating steps, one that defines groups of points and the other that computes the $\boldsymbol{\alpha}$ (the Lagrangian multipliers involved in the dual of the SVM, see Schölkopf and Smola (2002)). In order to make clear that CVM and SimpleSVM are similar, we briefly explain those steps for each. Doing so we also enlighten their differences. Then we give the two algorithms and specify where stopping conditions apply.

2.1.1 DEFINING THE GROUPS

CVM: divides the database between *useless* and *useful* points. The useful points are designated as the *coreset* and correspond to all the points that are candidates to be support vectors. This set can only grow since no removing step is done. Among the points of the coreset, not all are support vectors in the end.

SimpleSVM: divides the database into three groups: I_w for the non bounded support vectors ($0 < \boldsymbol{\alpha}_w < C$), I_C for the bounded points - misclassified or in the margins ($\boldsymbol{\alpha}_C = C$) and I_0 which contains the non support vectors ($\boldsymbol{\alpha}_0 = 0$).

2.1.2 FINDING THE $\boldsymbol{\alpha}$

CVM: solves the QP on the coreset only using SMO and thus obtains the $\boldsymbol{\alpha}$.

SimpleSVM: solves an optimization problem such that the optimality condition leads to a linear system. This linear system has $\boldsymbol{\alpha}_w$ as unknown.

2.1.3 ALGORITHMS

CVM: uses the following steps:

Algorithme 1 CVM algorithm

1: initialize ▷ two points in the coreset, the first center of the ball, the first radius (see the original paper for details on the MEB)
2: **if** no point in the remaining points falls outside the ϵ-ball **then**
3: stop (with ϵ-tolerance) ▷ outer loop
4: **end if**
5: take the furthest point from the center of the current ball, add it to the core set
6: solve the QP problem on the points contained in the coreset (using SMO with warm start with ϵ-tolerance) ▷ inner loop
7: update the ball (center and radius)
8: go to second step

Two things require computational efforts there: the distance computation involved in steps 2 and 5, and solving the QP in step 6. For the distance computation, the authors propose a heuristics that consists of sampling 59 points instead of checking all the points, thus reducing greatly the time required. Doing so they ensure at 95% that the furthest point among those 59 points is part of the 5% of points the furthest from the center. For step 6, they use warm start feature of the SMO algorithm, doing a rank-one update of the QP.

SimpleSVM: uses the following steps:

Algorithme 2 SimpleSVM algorithm

1: initialize ▷ two points in I_w, first values of $\boldsymbol{\alpha}_w$
2: **if** no point in the remaining points is misclassified by the current solution **then**
3: stop (with ϵ-tolerance)
4: **end if**
5: take the furthest misclassified point from the frontier, add it to I_w
6: update the linear system and retrieve $\boldsymbol{\alpha}_w$
7: **if** all $0 < \boldsymbol{\alpha}_w < C$ **then**
8: go to second step
9: **else**
10: remove violating point from I_w (put it in I_0 or I_c depending on the constraint it violates) and go to step 6
11: **end if**

Again, two things require computational efforts here: the classification involved in steps 2 and 5, and solving the linear system in step 6. For the classification, a heuristic is used that consists of taking the first violator found (yet all the points are eventually checked hence insuring an exact resolution). For step 6, a rank-one update is performed which is $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3)$ for full resolution.

2.2 Stopping criteria

We now consider the stopping criteria used in those algorithms. Indeed we have noted in the previous algorithms steps where a stopping criteria is involved (the ϵ-tolerance).

For the sake of clarity, we will also give details about the ν-SVM. Indeed, we will see that the CVM stopping criterion is analog to the ν-SVM one.

Notations and definitions Let $\mathbf{x}$ be our data labeled with $\mathbf{y}$. Let $k(.,.)$ be the kernel function. We will denote $\boldsymbol{\alpha}$ the vector of Lagrangian multipliers involved in the dual problem of SVMs.

K is the kernel, such that $K_{ij} = y_i y_j k(x_i, x_j)$. As defined in Tsang et al. (2005, eq. 17), $\tilde{K}$ is the modified kernel, such that $\tilde{K}_{ij} = y_i y_j k(x_i, x_j) + y_i y_j + \frac{\delta_{ij}}{C}$ ($\delta_{ij} = 1$ if $i = j$ and 0 otherwise). $\tilde{K}$ is used for the reformulation of the SVM as a MEB problem.

As defined in Tsang et al. (2005, page 370), $\tilde{\kappa} = \tilde{K}_{ii} = \kappa + 1 + \frac{1}{C}$ with $\kappa = K_{ii}$ (which is equal to 1 for the RBF kernel).

$\mathbf{1}$ stands for vectors of ones, $\mathbf{0}$ for vectors of zeros. All vectors are columns unless followed by a prime.

C-SVM type (SimpleSVM and SMO) The stopping criterion commonly used in classical SVM (C-SVM) is

$$\min(K\boldsymbol{\alpha}_1 - \mathbf{1}) \geq -\epsilon_1 \tag{1}$$

The two implementations SimpleSVM and libSVM (the C-SVM version) use similar stopping criteria, up to the biased term. This corresponds to the constraint of good classification. Here the magnitude of $\boldsymbol{\alpha}$, influenced by C, slightly modifies the *tightness* of the stopping condition. For SMO, a very strict stopping criteria causes a large number of iterations and slows down the training speed. This is the reason why it is usually admitted that SMO should not be run with large C values (and this idea is often extended to SVM independently of the solver, which is a wrong shortcut). On the other hand, SimpleSVM's training speed is linked to the number of support vectors (there is no direction search with a first order gradient to slow it down). For separable problems, large C produces sparser solutions, so SimpleSVM is faster with large C.

ν-SVM type In the ν-SVM setting (see Chen et al. (2005) for further details), the stopping criterion is

$$\min(K\boldsymbol{\alpha}_2 - \rho_2\mathbf{1}) \geq -\epsilon_2 \tag{2}$$

where ρ_2 corresponds to the margin (see Chang and Lin (2001b) for derivations). In order to compare this criterion to the previous one, it is possible to exhibit links between the two problems. Let say that for a given problem and a given ϵ_2, the ν-SVM optimal solution is $\boldsymbol{\alpha}_2$ and ρ_2. To obtain the same solution using the C-SVM, one can use the following relations: $C = 1/\rho_2$ (see Schölkopf and Smola (2002, p209, prop. 7.6)) and $\epsilon_1 = \epsilon_2/\rho_2$. The solution $\boldsymbol{\alpha}_1$ obtained by the C-SVM can also be compared to $\boldsymbol{\alpha}_2$ since we know that $0 \leq \boldsymbol{\alpha}_2 \leq 1$ and $0 \leq \boldsymbol{\alpha}_1 \leq C$. Thus there is $\boldsymbol{\alpha}_2 = \boldsymbol{\alpha}_1/C$. Thanks to those relations it is possible to express the C-SVM stopping criterion with the ν-SVM terms by setting

$$\epsilon_1 = \epsilon_2 C = \frac{\epsilon_2}{\rho_2}$$

With a small ϵ_2, a small margin found by the ν-SVM leads to a large C (for instance $1/\epsilon_2$). The equivalent C-SVM should run with $\epsilon_1 = 1$ which is very loose. Because of this scaling effect, comparison should be careful.

CVM CVM uses two overlapping loops. The outer one for the coreset selection and the inner one to solve the QP problem on the coreset. Thus CVM require two stopping criteria, one per loop.

Let us first consider the inner loop, *i.e.* points that are constituting the coreset. The related multipliers $\boldsymbol{\alpha}_3$ are found using SMO. However, as mentioned in Tsang et al. (2005, equation 5), the solved QP is similar to a ν-SVM problem. Hence the stopping condition is similar to the ν-SVM stopping condition. The major difference between a ν-SVM and the inner loop of CVM is the use of a modified kernel $\tilde{K}$ instead of K. The leading stopping criterion is the one of the outer loop.

The stopping criterion of the outer loop (equivalent to eq. 25 of the paper) is:

$$\min\big((\tilde{K}\boldsymbol{\alpha}_3)_{\mathcal{R}} - \rho_3\mathbf{1}\big) \geq -\epsilon_3\tilde{\mathrm{K}} \tag{3}$$

where $\mathcal{R}$ indicates points that are outside the coreset. Note that if $\boldsymbol{\alpha}_3$ is the CVM solution for a given ϵ_3, since $\mathbf{1}'\boldsymbol{\alpha}_3 = 1$, the equivalent solution $\boldsymbol{\alpha}_1$ given by a C-SVM is linked as follows : $\boldsymbol{\alpha}_3 = \boldsymbol{\alpha}_1/(\mathbf{1}'\boldsymbol{\alpha}_1)$. Moreover, setting $\rho_3 = 1/(\mathbf{1}'\boldsymbol{\alpha}_1)$, using those relations and expanding the kernel expression give:

$$\min\Big(\frac{\big((K + \mathbf{y}\mathbf{y}' + \frac{1}{C}Id)\boldsymbol{\alpha}_1\big)_{\mathcal{R}} - \mathbf{1}}{\mathbf{1}'\boldsymbol{\alpha}_1}\Big) \geq -\epsilon_3(\mathrm{K} + 1 + \frac{1}{C}) \tag{4}$$

Here again thanks to those relations it is possible to express the C-SVM stopping criterion with the outer CVM terms by taking

$$\epsilon_1 = \frac{\epsilon_3(\mathrm{K} + 1 + \frac{1}{C}) + \big((\mathbf{y}\mathbf{y}' + \frac{1}{C}Id)\boldsymbol{\alpha}_3\big)_\ell}{\rho_3} \tag{5}$$

where ℓ denotes the indice of the point minimizing the left-hand side part of (4). We can observe a similar scale effect to the one happening for ν-SVM.

Apart from the possibility to compare fairly to other solvers, let us study the behavior of CVM depending on C or on the training set size.

- The modified kernel hides a non-negligible regularizing effect. Indeed $1/C$ is added to the diagonal. Thus for small C, the kernel becomes well conditioned. Hence we can expect CVM to give better results with small C.
- From its ν-SVM like form, it also gets stricter condition for small C. This strict condition increases the number of steps and, as a side effect, the probability to check more points outside the coreset when sampling the 59 points.

Overall, for a small C, CVM should be slower than SMO (since it repeats SMO) and give accurate results.

- A large C, on the contrary, does not influence the kernel conditioning anymore
- From the form of the stopping criterion, the condition is looser
- The inner stopping condition is loose too. The looser it is, the fewer support vectors it takes.

Overall, we can expect less accurate results for large C.

As for the training set size, it plays on the scaling effect. The more points there are, the more important this effect is. As a consequence, the previous tendencies depending on C are more visible for large datasets. Finally, we would like to underline the fact that the stopping criterion for the inner loop is not clearly defined. This analysis is simplified by admitting that the inner stopping criterion is equivalent to the outer one.

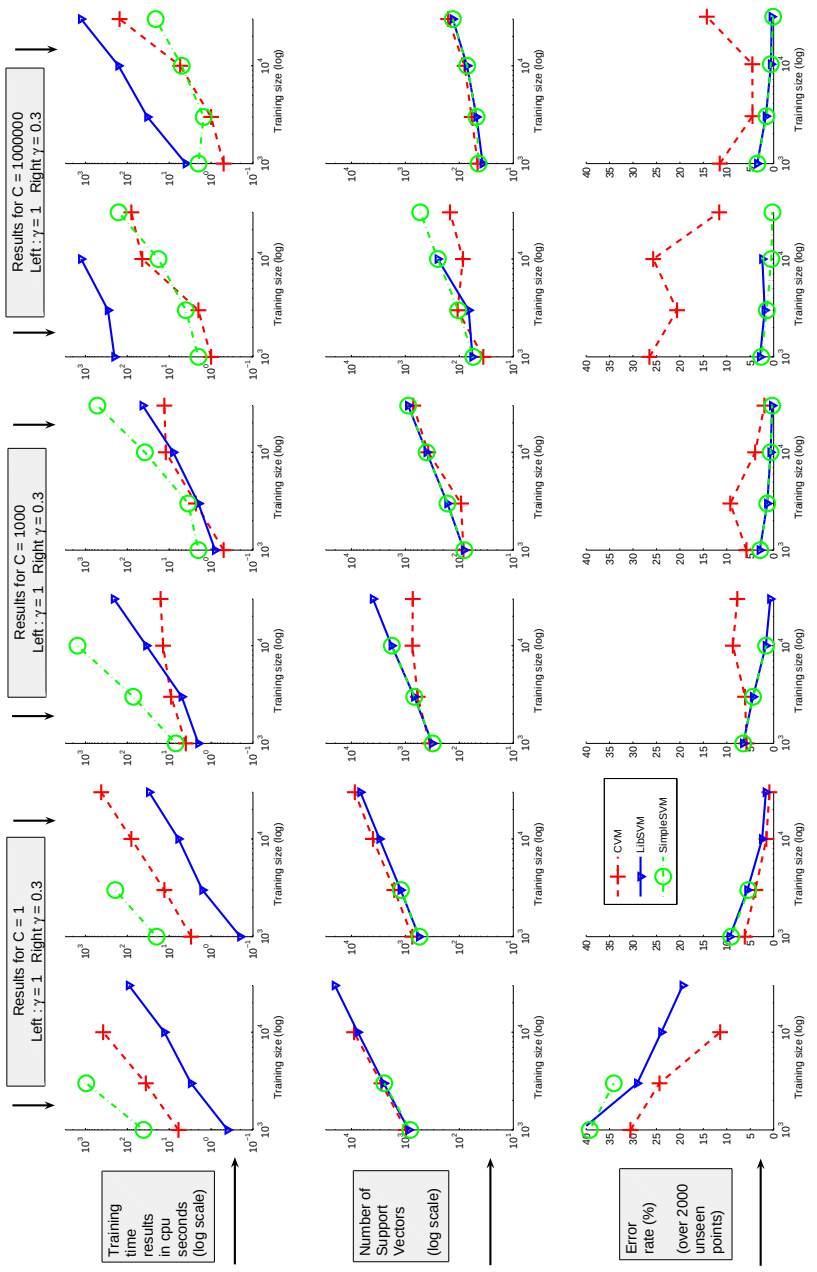

Figure 2: Experiments to compare results for different sets of hyper-parameters. Each column shows results for a couple of settings γ and C. For each column, the first row gives the training time for each algorithm. The second row shows the number of support vectors while the third gives the error rate. All reported results are obtained with a KKT tolerance $\epsilon = 10^{-6}$. Note that those experiments are run up to 30,000 points only. Missing results are due to early termination because of not enough memory and/or the too long training time.

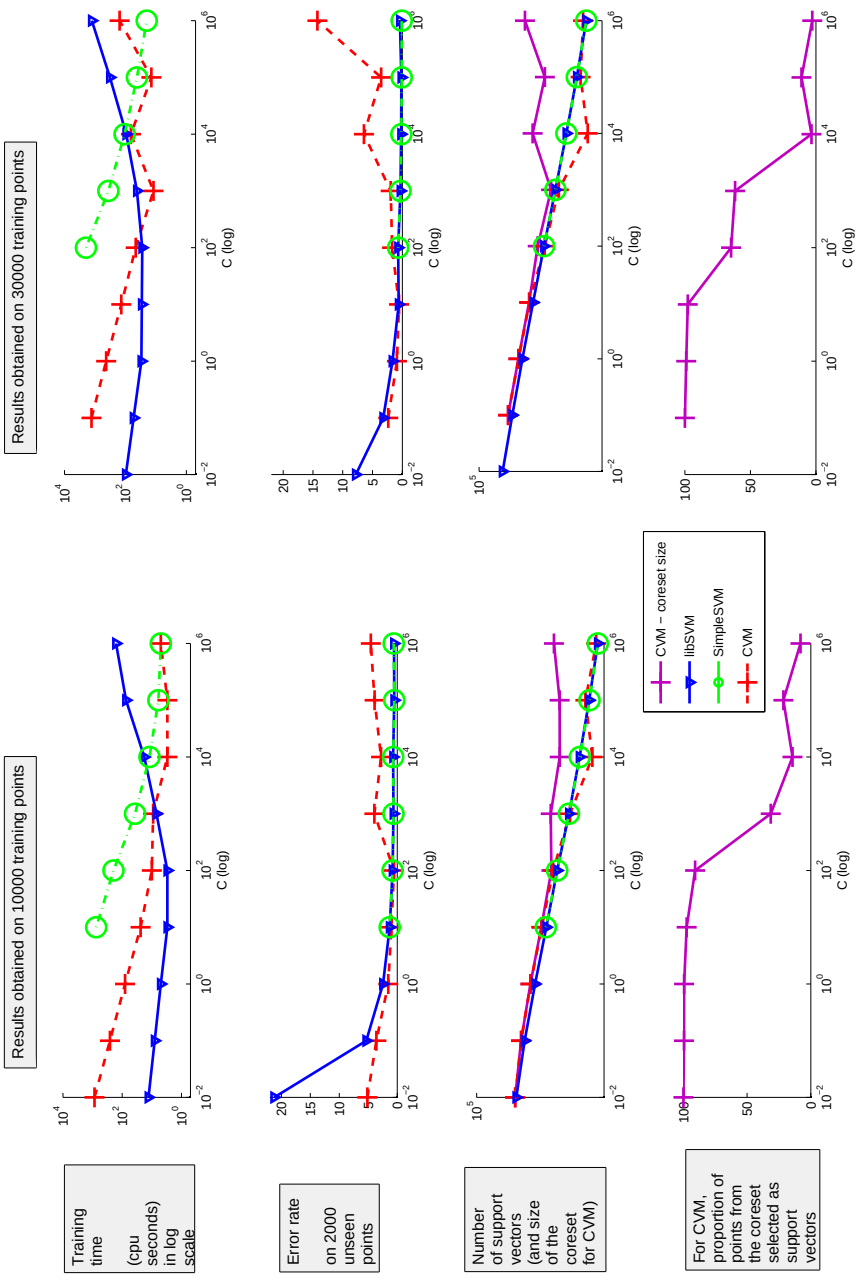

Figure 3: Experiments to illustrate the influence of hyper-parameter C for the different algorithms. First column reports experiments with 10,000 training points and the second uses 30,000 points. For each column, the top row gives the training time (log scale) depending on C while the second row shows the error rate achieved on 2000 unseen points. The third row gives the number of support vector as well as the size of the coreset for CVM. The bottom row reports the proportion of support vectors selected in the coreset for CVM. All experiments are done with $\gamma = 1$ and $\epsilon = 10^-6$.

3. Experimental results

We now propose experiments illustrating the previous analysis. We first make the training size varies and then C. For all the experiments, we use $\epsilon = 10^{-6}$.

Behavior of the algorithms when size is growing Figure 2 shows the results for experiments where the training size is growing for various sets of hyper-parameters. We observe :

- for a given C, results are similar in terms of training time, regardless of the kernel bandwidth,
- the best configuration for CVM is medium C regarding training time and small C regarding accuracy,
- for small C, CVM is the most accurate method (due to a regularization on the kernel),
- for large C, CVM accuracy tends to decrease.

It is clear from this experiment that we face different behaviors depending on the hyper-parameters. Each method has a favorable *mode* and those are not compatible. Since the bandwidth does not influence the trends of the obtained curves, we will fix it and focus now on variations of C.

Behavior of the algorithms when C is varying Figure 3 shows how the training changes depending on the value of C for each studied algorithm. The same experiment is conducted for two training sizes (10,000 points for the first column, 30,000 for the second), in order to point out that the training size amplifies sensitivity of CVM to its hyper-parameters. Each experiment reports the training time (top figure), the error rate (second figure), the number of support vectors and the size of the coreset (third figure) and the proportion of support vectors in the coreset for CVM (last figure), $C \in [10^{-2}, 10^{6}]$.

small C SimpleSVM fails because of not enough memory, libSVM is the fastest (loose stopping condition) and CVM the most accurate (regularized kernel). The coreset size corresponds almost exactly to the number of support vectors, which means that the selection criterion is accurate.

large C SimpleSVM is faster than libSVM (penalized by strict stopping condition) and both are as accurate. CVM gets large error rate and we see that the less the coreset set is related to the support vectors, the less accurate the method is. The point that is not so clear is the reason why the coreset grows that much and this is still to be explored.

Conclusion

In their conclusion, the authors claim that *"experimentally, it [CVM] is as accurate as existing SVM implementations"*. We show that is not always true and furthermore that it may not converge towards the solution.

Moreover we have shown that comparisons between CVM and usual SVM solvers should be careful since the stopping criteria are different. We have illustrated this point as well as the effect of magnitude of C on the training time. We have explained partly the behavior of CVM and shown that their results may be misleading, yet it still requires further exploration (particularly regarding the actual effects of the random sub-sampling).

As mentioned by Chang and Lin (2001b, conclusion), who are studying the ν-SVM (which has a similar stopping condition to the CVM, *i.e.* very sensitive to the α magnitude), we would need

here an adaptive ϵ in order to avoid unstable behaviors but it can not be done easily. Furthermore, one who uses SVM should be aware that each method has favorable and non favorable ranges of hyper-parameters.

Finally, we acknowledge that a fast heuristic which achieves loose yet still acceptable accuracy is interesting and useful for very large databases (by itself or as a starting point for more accurate methods) but to use it with confidence, one has to know when the method is indeed accurate enough. Hence CVM may benefit from being presented along with its limits.

Acknowledgment

We would like to thank very much the anonymous reviewers for helpful comments on the stopping conditions. This work was supported in part by the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778. This publication only reflects the authors' views.

References

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines, 2001a. URL `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chih-Chung Chang and Chih-Jen Lin. Training ν-support vector classifiers: Theory and algorithms. *Neural Computation*, (9):2119–2147, 2001b.

P. H. Chen, C. J. Lin, and B. Schölkopf. A tutorial on v-support vector machines. *Applied Stochastic Models in Business and Industry*, 21(2):111–136, 2005.

Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 2005.

Gaëlle Loosli. Fast svm toolbox in Matlab based on SimpleSVM algorithm, 2004. `http://asi.insa-rouen.fr/~gloosli/simpleSVM.html`.

John Platt. Fast training of support vector machines using sequential minimal optimization. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advanced in Kernel Methods - Support Vector Learning*, pages 185–208. MIT Press, 1999.

Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, 2002.

Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *J. Mach. Learn. Res.*, 6:363–392, 2005.

S. V. N Vishwanathan, Alexander J. Smola, and M. Narasimha Murty. SimpleSVM. *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.