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

Gaëlle Loosli, Stéphane Canu

June 30, 2006

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.

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 independant 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 bandwidth of the kernel (γ), the slack trade-off (C) and the stopping tolerance (ϵ). For RBF kernels, the bandwidth 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 choosen with cross-validation or using knowledge on the the problem. Those two hyper-parameters are well-known and more details can be found in litterature [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, 2001] 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 has two stopping criteria that are playing different roles (and sometimes antagonists) and this may induces 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 enlights those different modes.

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, 2001]** 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.

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.

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 - $k(x_i, x_j) = exp\big(-\gamma(x_i - x_j)^2\big)$ and that β is the notation used in the studied paper. Results are presented on figure 1, first column.

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].

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.

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

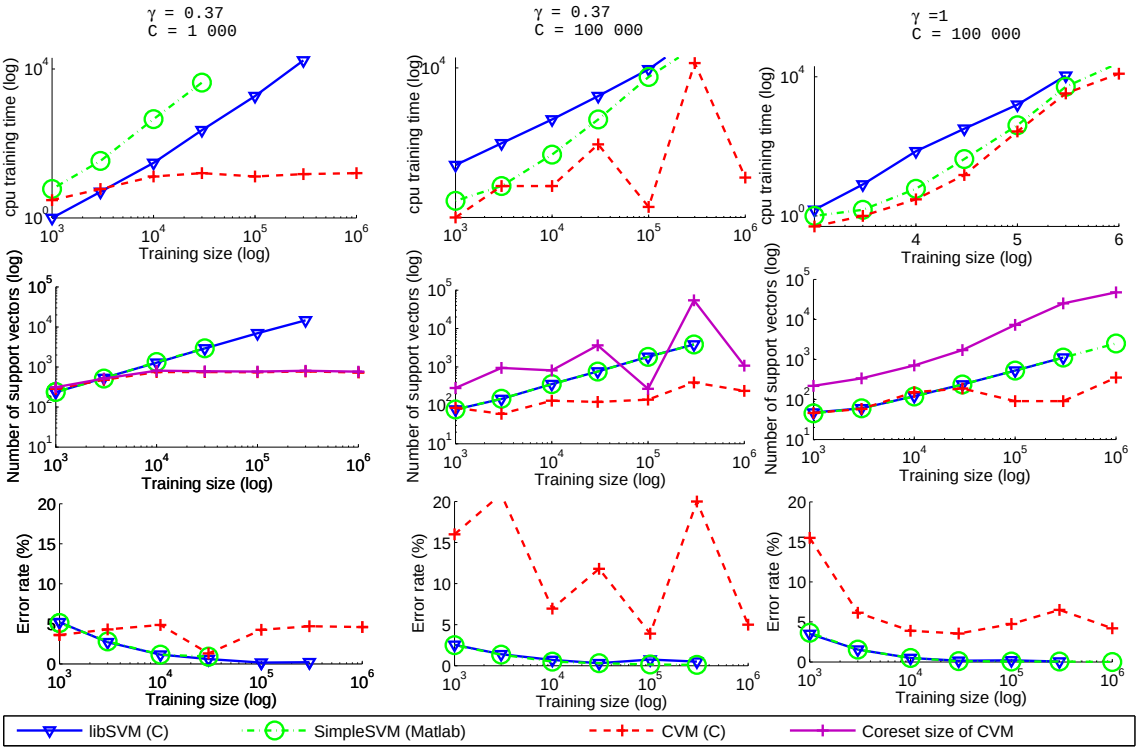

Figure 1: Comparison between CVM, libSVM and SimpleSVM on the checkers problem. For all figures, ∇ is for libSVM, o for SimpleSVM and + for CVM. The three figures on the first column show the results while trying to reproduce figure 3 (page 379) from the paper ($C = 1000$, $\gamma = 0.37$, *cf.* page 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.

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 enlight 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 gets the $\boldsymbol{\alpha}_w$ by solving a system of linear equations.

2.1.3 Algorithms

CVM uses the following steps:

1 – initialize (two points in the coreset, the first center of the ball, the first radius [2])
2 – if no point in the remaining points falls outside the ϵ-ball, stop (with ϵ_1-tolerance[3])
3 – take the furthest point from the center of the current ball, add it to the core set
4 – solve the QP problem on the points contained in the coreset (using SMO with warm start with ϵ_2-tolerance[4])
5 – update the ball (center and radius)
6 – go to second step

Two things require computational efforts there: the distance computation involved in steps 2 and 3, and solving the QP in step 4. For the distance

[2] see the original paper for details on the MEB
[3] See subsection 2.2 for notation
[4] See subsection 2.2 for notation

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 4, they use warm start feature of the SMO algorithm, doing a rank-one update of the QP.

SimpleSVM uses the following steps:

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, stop (with ϵ_3-tolerance[5])
3 – take the furthest misclassified point from the frontier, add it to I_w
4 – update the linear system and retrieve $\boldsymbol{\alpha}_w$
5 – if all $0 < \boldsymbol{\alpha}_w < C$ go to second step
6 – else remove violating point from I_w (put it in I_0 or I_c depending on the constraint it violates) and go to step 4.

Again, two things require computational efforts here: the classification involved in steps 2 and 3, and solving the linear system in step 4. 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 4, 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).

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.
ϵ_1, ϵ_2 and ϵ_3 are the stopping tolerances, respectively for the CVM coreset, SMO involved in CVM and SimpleSVM stopping criterion.

[5] See subsection 2.2 for notation

SimpleSVM and SMO The stopping criterion commonly used in SVM is

$$\min(K\boldsymbol{\alpha} - \mathbf{1}) \geq \epsilon = \epsilon_2 = \epsilon_3 \tag{1}$$

It is used both in SimpleSVM and libSVM, so ϵ_2 and ϵ_3 are equals. This corresponds to the constraint of good classification. Here the magnitude of $\boldsymbol{\alpha}$, influenced by C, slightly modifies the *strictness* 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.

CVM Let's first consider points that are constituing the coreset. The related multipliers $\boldsymbol{\alpha}_{coreset}$ are found using SMO. Hence the stopping criterion used is the one given previously (equation 1). Let's now consider points that are not part of the coreset yet. Then the stopping criterion (equivalent to eq. 25 of the paper) is:

$$\min\big((\tilde{K}\boldsymbol{\alpha})_\ell(1 - \mathbf{1}'\boldsymbol{\alpha}_\ell)\big) \geq -\epsilon\tilde{\kappa} = -{\epsilon_1}^6 \tag{2}$$

ℓ indicates points that are outside the coreset. See annexe A for details. Taking into account that the C slack trade-off influences both the values of $\tilde{\kappa}$ and $\mathbf{1}'\boldsymbol{\alpha}_\ell$ (recall that C is a bound on each α), we can analyse this stopping criterion as follows:

- For RBF kernel, $\tilde{\kappa} = 2 + \frac{1}{C}$. If C is large, $\tilde{\kappa}$ is almost 2 and can be ignored. If $0 < C \leq 1$ then $\tilde{\kappa} \geq 3$ and may loosen the stopping condition.
- This stopping criteria is sensitive to the magnitude of $\mathbf{1}'\boldsymbol{\alpha}$ (large value loosens the stopping condition). This value is influenced by two settings:
  - The amount of data, since $\boldsymbol{\alpha} \geq \mathbf{0}$, the sum grows with the number of support vectors [7],
  - C, since large C permits large values of $\boldsymbol{\alpha}$ and small C limits them.

2.2.1 Summary

For large C

CVM stopping condition ϵ_1 is loose and ϵ_2 is very strict. It causes higher error rate (loose selection of coreset) and slow resolution (long intermediate SMO steps)

[6] Remark here that ϵ is given to the algorithm and ϵ_1 is the actual stopping value

[7] remark that this is illustrated on fig 1, the test error is growing for CVM when the training size increases. This is due to the looser and looser stopping condition.

SimpleSVM the small number of support vectors causes a fast training.

SMO the very strict stopping criteria causes a large number of iteration before convergence, hence a slow training (penalized by the first order descent on an almost flat surface).

For small C

CVM if C is small but greater than 1, stopping condition ϵ_1 is strict and leads to slow training (many iterations). Error rate is thus very low (we think that a high number of iteration increases the probability of visiting every training point even with the sampling trick). If C is less than 1, then the stopping condition becomes looser (for instance, $C = 0.001$ multiplies the given tolerance by 1000).

SimpleSVM the large number of support vectors induces a slow training time and the memory may be too short.

SMO the stopping condition is loose hence SMO converges in a relatively small number of iterations, hence there is a fast training.

3 Experimental results

We now propose experiments illustrating the previous analysis. We first make the training size varie and then C. For all the experiments, we use $\epsilon = 10^{-6}$. We reason why we do so is that there is no way to get identical stopping criteria for all methods since CVM uses this value for two different things. Having said that, the comparison is not necesseraly unfair, since on the one hand, libSVM and SimpleSVM uses the same stopping criteria and on the other hand, CVM uses also the very same stopping criteria for the QP solver. The difference only applies to the coreset selection.

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 (the coreset selection is based on a strict criterion while the others have a loose criterion).

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), $C \in [10^{-2}, 10^{6}]$. As expected, we can analyze those curves by parts, the changing point being between $C = 100$ and $C = 1000$ depending on the training size:

small C SimpleSVM fails because of not enough memory, libSVM is the fastest and CVM the most accurate. The coreset size corresponds almost exactly to the number of support vectors

large C SimpleSVM is faster than libSVM and both are as accurate. CVM get large error rate and we see that the less the coreset set is related to the support vectors, the less accurate the method is and the slowest it is. The lack of accuracy is certainly related to the loose stopping condition on the coreset selection. The training time of CVM is easily explained since it has a large coreset. Indeed the internal SMO step is slow due to C (so a strict stopping condition) and is applied to many points, even tough there are only a few percent of support vectors in the end. So each CVM iteration is slow. 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, on the contrary, that far from it, *CVM is not as accurate as other implementations* and furthermore that it may not converge towards the solution, while libSVM and SimpleSVM do. Hence the reader may have to trade-off between *fast and approximate solution* - CVM - and *slower (yet feasible) and exact solution* - SimpleSVM or libSVM.

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.

As mentioned by **?**, conclusion, who are studying the νSVM (which has a similar stopping condition to the CVM, *i.e.* very sensitive to the $\boldsymbol{\alpha}$ magnitude), we need here an adaptive ϵ in order to avoid unstable behaviors. On top of that, one who uses SVM should be aware that each method has favorable and non favorable ranges of hyper-parameters.

Finally, we also acknowledge that a fast heuristic which achieves loose yet still acceptable accuracy can be 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 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 (version 2.3), 2001. URL `citeseer.nj.nec.com/chang01libsvm.html`.

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. *Journal of Machine Learning Research*, 6:363–392, 2005. ISSN 1533-7928.

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

A CVM Stopping Criterion

In Tsang et al. [2005, eq 25], the authors give the stopping condition to include new points in the coreset.

$$y_\ell\big(\mathbf{w}'\phi(\mathbf{x}_\ell)+b\big)-\rho \geq -(\epsilon+\frac{\epsilon^2}{2})\tilde{\mathrm{K}} \tag{3}$$

ℓ designate the points that are not part of the coreset.
ϕ is a mapping of $\mathbf{x}$.
ρ is a variable of the problem.
$\mathbf{w}$ is the vector of coefficients defining the separating hyperplane.
b is the offset of the hyperplane.

For the sake of clarity we simplify this expression without changing the analysis of the stopping criteria effect.

$$y_\ell\big(\mathbf{w}'\phi(\mathbf{x}_\ell)+b\big)-\rho \geq -\epsilon\tilde{\mathrm{K}} \tag{4}$$

Now let's rewrite this for all the points. As shown in Tsang et al. [2005, pages 372] for points lying outside the coreset,

$$(\tilde{K}\boldsymbol{\alpha})_\ell = y_\ell\big(\mathbf{w}'\phi(\mathbf{x}_\ell)+b\big) \tag{5}$$

Hence,

$$\min\big(\tilde{K}\boldsymbol{\alpha}-\rho\mathbf{1}\big) \geq -\epsilon\tilde{\mathrm{K}} \tag{6}$$

From Tsang et al. [2005, eq 15] we know that $\rho = \boldsymbol{\alpha}'\tilde{K}\boldsymbol{\alpha}$, hence

$$\min\big(\tilde{K}\boldsymbol{\alpha}-\boldsymbol{\alpha}'\tilde{K}\boldsymbol{\alpha}\mathbf{1}\big) \geq -\epsilon\tilde{\mathrm{K}} \tag{7}$$

$$\min\big((Id-\mathbf{1}\boldsymbol{\alpha}')\tilde{K}\boldsymbol{\alpha}\big) \geq -\epsilon\tilde{\mathrm{K}} \tag{8}$$

where Id is the identity matrix. Let denote $v = \tilde{K}\boldsymbol{\alpha}$ and $M = Id - \mathbf{1}\boldsymbol{\alpha}'$. Note that the general term of matrix M is:

$$\begin{cases} M_{ii} & = 1-\alpha_i \\ M_{ij|i\neq j} & = -\alpha_i \end{cases} \tag{9}$$

Equation 8 is equal to:

$$-\epsilon\tilde{\mathrm{K}} \leq \min\big(Mv\big) \tag{10}$$

with

$$\big(Mv\big)_i = (1-\alpha_i)v_i + \sum_{j\neq i} -\alpha_j v_j = v_i\Big(1-\sum_j \alpha_j\frac{v_j}{v_i}\Big) \tag{11}$$

Let denote $v_m = \min_i v_i$. Since $v_m > 0$ (see 5), from 10 and 11 we have

$$-\epsilon\tilde{\mathrm{K}} \leq \min_i\Big[v_i\Big(1-\sum_j \alpha_j\frac{v_j}{v_i}\Big)\Big] = v_m\Big(1-\sum_j \alpha_j\frac{v_j}{v_m}\Big) \tag{12}$$

Since $\frac{v_j}{v_m} \geq 1$, $\big(1-\sum_j \alpha_j\frac{v_j}{v_m}\big) \leq \big(1-\sum_j \alpha_j\big)$. This leads to

$$-\epsilon\tilde{\mathrm{K}} \leq v_m\Big(1-\sum_j \alpha_j\Big) = min\big(\tilde{K}\boldsymbol{\alpha}(1-\boldsymbol{\alpha}'\mathbf{1})\big)\ \square \tag{13}$$

Moreover, we can also recall here that, $\tilde{\mathrm{K}} = \mathrm{K} + 1 + \frac{1}{C}$ and $\tilde{K} = K + \mathbf{y}\mathbf{y}' + \frac{1}{C}Id$ thus

$$\min\big((1-\boldsymbol{\alpha}'\mathbf{1})(K+\mathbf{y}\mathbf{y}'+\frac{1}{C}Id)\boldsymbol{\alpha}\big) \geq -\epsilon\big(\mathrm{K}+1+\frac{1}{C}\big) \tag{14}$$

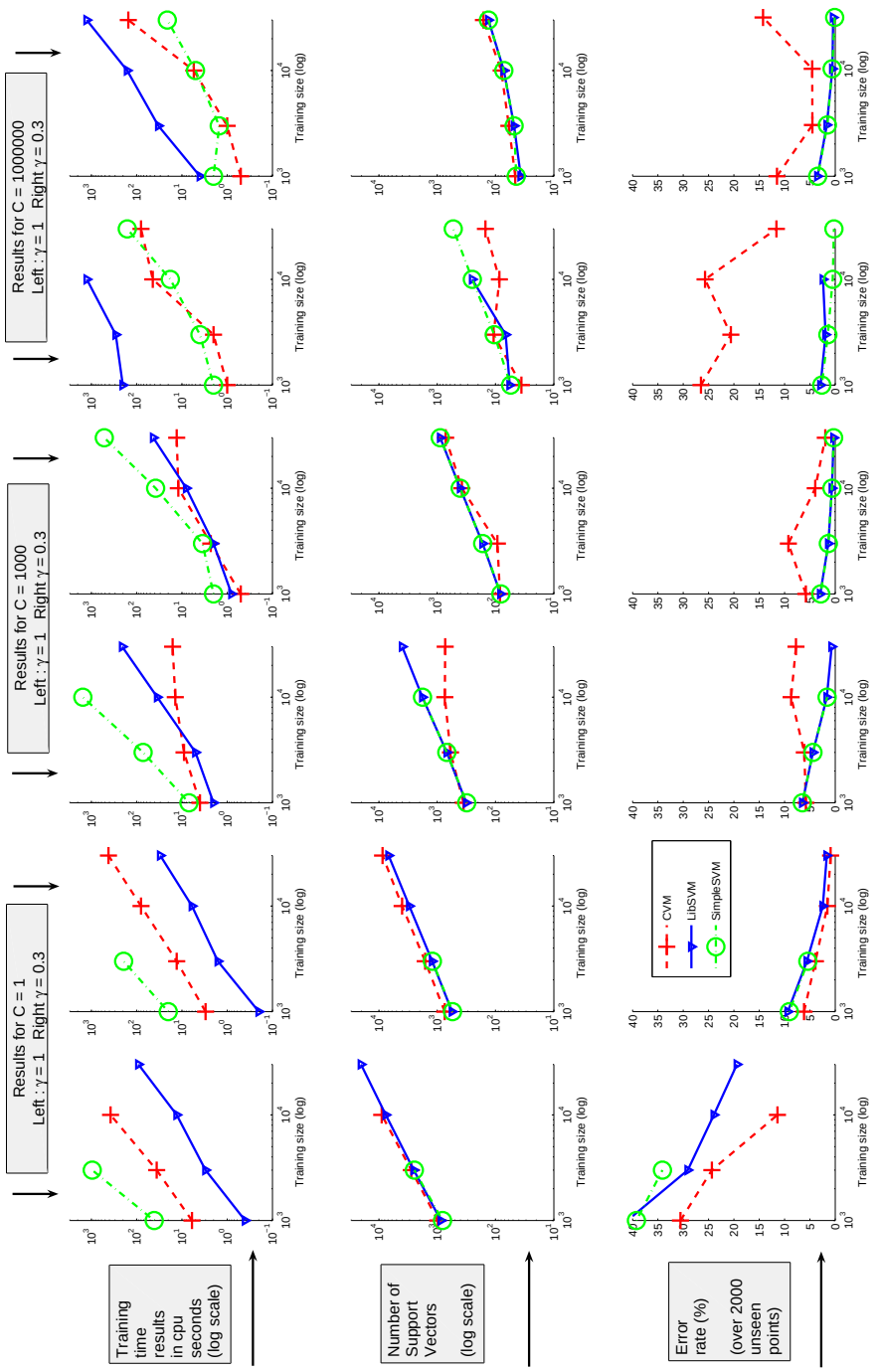

Figure 2: Experiments to compare results for different sets of hyper-parameters. Each column shows results for a couple of settings γ and C. For each column, the first row gives the training time for each algorithm. The second row shows the number of support vectors while the third gives the error rate. 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.

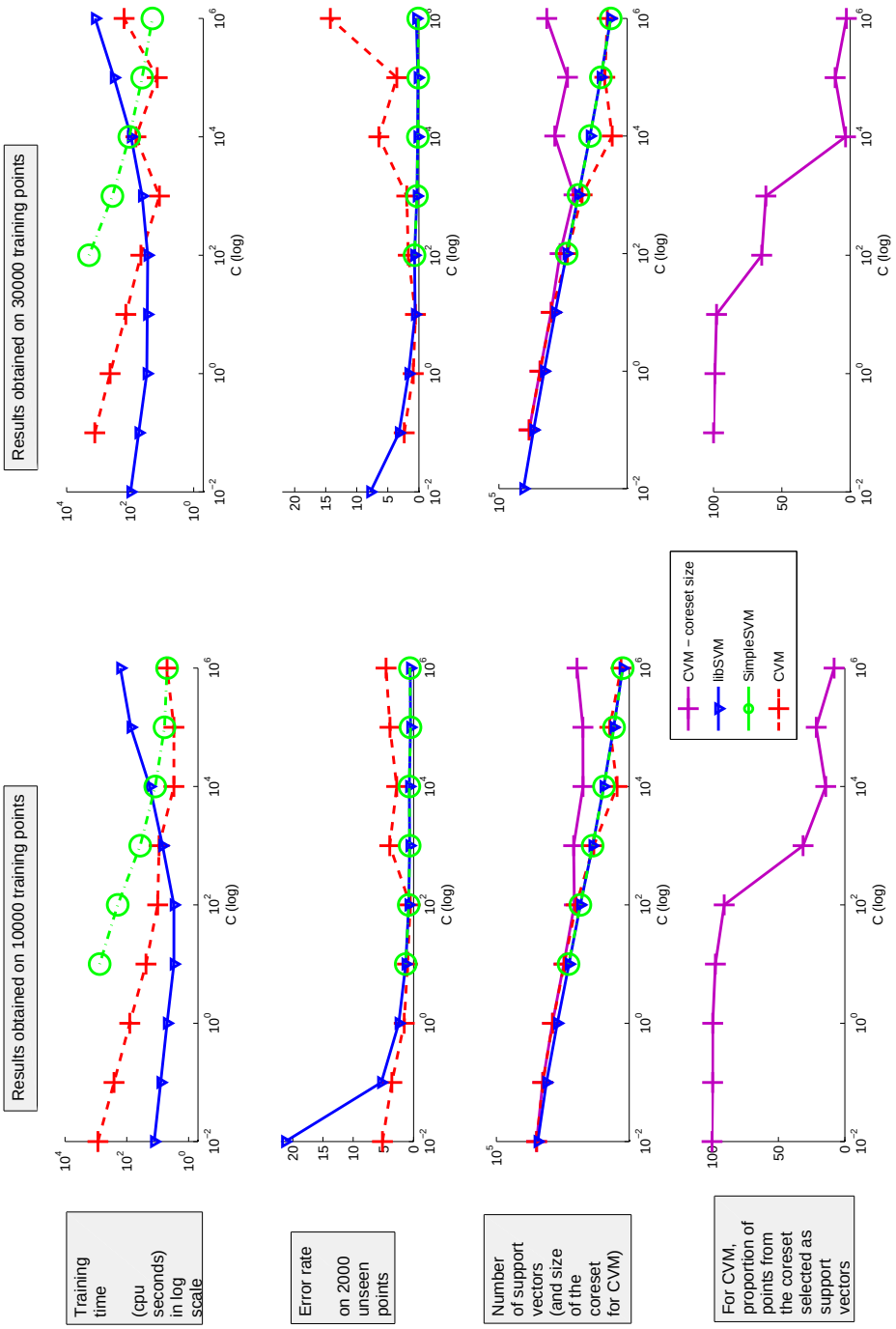

Figure 3: Experiments to illustrate influence of hyper-parameter C for the different algorithms. First column reports experiments with 10,000 training points and the second uses 30,000 points. For each column, the top row gives the training time (log scale) depending on C while the second row shows the 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 $\epsilon = 10^-6$.