BALK : Bandwidth Autosetting for SVM with Local Kernels Application to data on incomplete grids ⋆

Loosli Gaëlle[1], Deffuant Guillaume[1], Stéphane Canu[2]

[1] Cemagref, Laboratoire d'Ingénierie des Systèmes Complexes,
24 avenue des Landais, 63172 Aubière, France
prenom.nom@cemagref.fr

[2] LITIS, Laboratoire d'Informatique, Traitement de l'Information et Signal,
Avenue de l'Université, 76800 Saint Etienne du Rouvray, France
scanu@insa-rouen.fr

Résumé : This paper focuses on the use of Support Vector Machines (SVM) when learning data located on incomplete grids. We identify here two typical behaviours to be avoided, that we call holes. Holes are regions of the space with no training data where the decision changes. We propose a novel algorithm which aims at preventing holes to appear. It automatically selects the local kernel bandwidth during training. We provide hard-margin and soft-margin versions and several experimental results. Even though our method is designed for a specific application, it turns out that it can be applied to more general problems. **Mots-clés** : SVM, Incomplete grids, hyper-parameters

1 Introduction

This paper presents a new method to solve Support Vector Machines and select the local kernel bandwidth at the same time. We called this method BALK (Bandwidth Autosetting for Local Kernel SVM). This first version of BALK is dedicated to a particular configuration of the dataset : we are interested in solving the SVM when points are located on an incomplete grid of fixed step size ϵ. Solving this kind of problem is useful in several applications where one needs to approximate a shape in a space from points labelled "inside" or "outside" the shape. SVM are already used in those contexts (for instance Deffuant *et al.* (2007); Lagoudakis & Parr (2003); Deheeger & Lemaire (2007)). One problem of using SVM is the need for setting hyper-parameters. When learning on a grid, the number of points grows exponentially with the dimension D of the grid and using methods such as cross-validation becomes unfeasible in a reasonable time. Hence it is not rare to see applications where hyper-parameters are set based on the

⋆This research was supported by the European project PATRES (NEST-043268).

feeling of the practitioner. We propose here to provide a convenient tool that addresses this issue.

1.1 Viability kernel

We briefly describe first the problem that motivated this research direction. The viability theory aims at controlling a dynamical system such that it remains in the set of admissible states called K, the viability constraints set. In Aubin (1991), the author defines the concept of viable state, which is a state for which there exists at least one control function that allows to remain in K indefinitely. The set of all viable states is called the viability kernel (which has no relation with SVM kernels). The key point is to be able to determine this viability kernel for a given dynamic system, accurately and efficiently.

The viability theory comes with an algorithm to find the viability kernel. However, the result is an ensemble of points and not a function. In Deffuant *et al.* (2007), the authors propose to use SVM to find the separating function. Their algorithm is iterative, corresponding to time steps of the dynamical system. At the beginning, all points are positive, i.e. they are all considered as viable. Each step starts by training a SVM on the labelled points of the grid. Then, the SVM frontier is used together with the dynamics to determine wether a control can be found to stay inside K. If not, the label of the corresponding point is set to minus. Otherwise is remains positive. The iterations go on until no label is changed. The algorithm is also adapted to active learning since datasets are huge and labels expensive to compute. Doing so, they face a problem we called here holes. Because of the active learning scheme, there are large parts of the space without any training point and in those areas, some SVM frontiers may appear without any support vector to support them. Those holes give wrong controls and all following steps are wrong.

This paper aims at giving tools to set SVM hyper-parameters such that holes are avoided. By definition of the problem, a hard-margin SVM should be used since errors in the definition of the frontier at a given step of the dynamic system induces control errors at the next steps.

1.2 Organisation of the paper

The next section defines the problem we want to solve : we point out the effects of bad kernel hyper-parameters. In the following section, we focus on the hard-margin setting and propose the hard-margin BALK. The fourth section extends the results to soft-margin setting with the soft-margin BALK. In the fifth section we provide experiments on artificial datasets in 2 and 3 dimensions as well as results on viability kernels and we show the advantages of using our algorithm. This section reports preliminary results on general datasets, with examples that are not coordinates on a grid but images.

2 Learning on partial grids

As mentioned before, we are interested in learning shapes from points located on regular grids. In this kind of problem, one can access to any point on the grid. However, most of time, asking for a label is expensive. This is the reason why active learning is here a method of choice. For SVM, it is easy to implement simple heuristics to achieve active learning. The most used one is probably the one consisting in using points only if they fall inside the current margins. For illustration purposes and study of the problem only, we reproduce the effects of the active learning scheme with a very simple rule : only points that have an opposite-class neighbour at a distance less than $2\sqrt{D}\epsilon$ are eligible for training the SVM. This is the incomplete grid, obtained by filtering all points far from the frontiers. This is not meant to be real active learning, but only to illustrate the holes problem.

2.1 SVM and kernels

We briefly recall here the SVM decision function. For further reading we propose Burges (1998). Let $x_i,\ i \in [1, \ell]$ be the training points and $y_i,\ i \in [1, \ell]$ the associated labels. The SVM function is expressed as the linear combination of elements of the kernel : $f(.) = \sum_i \alpha_i y_i k(x_i, .)$ and the decision function is $D(.) = sign(f(.) + b)$ where b is the bias. We now look at the kernel. Since data is located on a grid, the euclidean distance is a relevant information and this is the reason why we work on local kernels such as the Gaussian kernel For this kernel, the only hyper-parameter is the bandwidth σ.

$$\text{Gaussian} \quad k(x, y) = \exp \frac{-\|x - y\|^2}{2\sigma^2}$$

2.2 When do holes appear ?

Holes can be separated in two categories, depending on their nature. We describe and explain both in the next paragraphs.

2.2.1 Lack of decision

The first one, which is also the easiest to deal with, is due to a lack of decision is some regions of the space. For radial kernels, it is a fact that far from the support vectors, decision is based on constant b since :

$$\lim_{\|x_i - x\|^2 \to \infty} k(x_i, x) = 0 \quad \Leftrightarrow \quad \lim_{\|x_i - x\|^2 \to \infty} f(x) = b \tag{1}$$

Increasing σ increases the distance between support vectors and bias decision. Hence a good way to prevent holes is to maximise σ. It is possible to compute the differential :

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{\ell} \frac{\alpha_i y_i}{\sigma} \frac{\|x - x_i\|}{\sigma} k(x_i, x) \frac{(x_i - x)}{\|x - x_i\|} \tag{2}$$

From this expression, one can evaluate how many support vectors $\mathbf{x}_i$ significantly influence a given point x (see Bengio *et al.* (2005), section 6.2 for details). They are those for which $\frac{\|x-x_i\|}{\sigma} k(x_i, x)$ is less than 2. This is useful to define, after learning, an area where the decision can be trusted regarding this first category of holes.

2.2.2 Geometry of the problem

The second category of holes is closely linked to the geometry of the problem. Hence it is hard to know in advance if there can be some. However, we can characterise the mechanism that plays here. In a problem where the geometry requires a lot of flexibility but the kernel bandwidth is large, some α_i become so large that the influence of the support vector goes much further than the others. This influence is totally compensated by several support vectors of the opposite label but further, it is not constrained by the data. We illustrate this on a very artificial problem on figure 1. We see that each positive (cross) support vector is compensated by at least 2 negative (circle) support vectors so that the decision is locally good even though the kernel bandwidth is large. For some regions that are further away, the positive support vectors are more influencing than the closest training points and we observe that the decision changes with no point to support this change. This creates a hole.

For α_i with smaller values this doesn't happen because the influences remain local. The values of α_i is related to the bandwidth value through the kernel. The higher σ, the higher α_i. So to avoid holes, we need reduce σ so that α_i remain in a reasonable range of values. Remark that this problem doesn't occur when using soft-margin with small C. For this category of holes, we want to limit α_i, which can only be obtained by minimising σ in the hard-margin setting.

2.2.3 Objective

Our point in this article is to provide a practical solution to avoid both holes. It is obvious that choosing a good kernel bandwidth is the key. However we wish to propose a method that does not require to test several values in order to choose the best one, retraining the SVM many times. In the specific problems we are interested in, holes errors are far more important than errors near the real frontier. Indeed, the precision of the results are given by the precision ϵ of the grid. Hence we propose to evaluate the quality of the solutions based on the absence of holes first and then on the generalisation error. We consider all solutions without holes. Among all, the best one will be the most regular one (which is likely to be the sparsest). Therefore, we want to select the largest bandwidth with no holes.

The most basic method to automatically set the kernel bandwidth consists in running several SVM with different bandwidths and selecting the best one. Differences between methods lie in the choice of the range of value and the criteria to evaluate the results. Some more sophisticated methods use gradient to find the next hyper-parameter value Cristianini *et al.* (1999); Keerthi *et al.* (2007). This is fine when it is reasonable to run several instances. However it is not always the case, in particular for large datasets like grids in high dimensions. Recently, Wang *et al.* (2007) proposed a regularisation

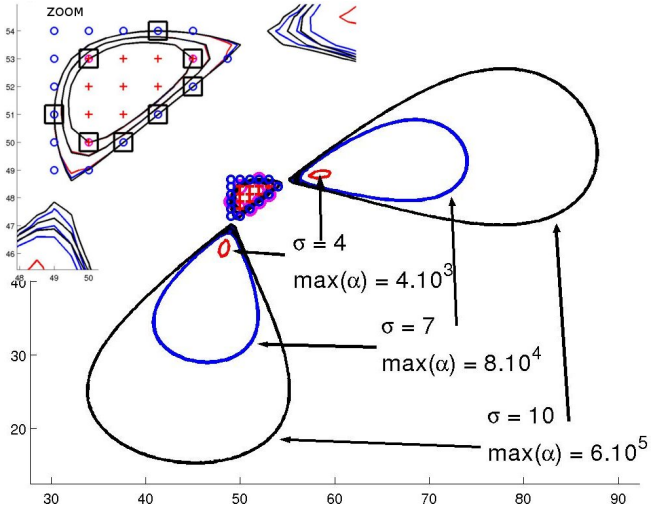

FIG. 1 – We draw HM-SVM frontiers for different bandwidths. Holes appear for σ larger than 4 and we see that $\max(\alpha_i)$ are larger and larger when σ increases. This very simple case is constructed to illustrate the problem.

path for the kernel bandwidth. Others methods are using multiple kernels (Rakotomamonjy *et al.* (2007)). An other approach (Burbidge (2002)) proposes to select the kernel bandwidth during training, similarly to what is presented here. The bandwidth is estimated from the median minimal distance between positive and negative support vectors. Hence the method consists in running the SVM and changing the bandwidth every h steps according to the current support vectors. This method is not applicable on a grid since the estimated bandwidth is likely to be ϵ at each step.

3 Hard-margin BALK

3.1 Formalisation

Small bandwidth σ leads to bias holes and high number of support vectors. Large bandwidth σ leads to less flexibility and implies large values of α_i obtained during optimisation. In soft-margin setting, the lack of flexibility for large bandwidths is compensated by the regularisation term C, which limits possible values of α_i. Our idea is to apply this limitation while keeping the hard-margin setting. If we impose hard-margin and a bound on α_i, the only way to ensure a feasible solution is to make σ a free variable of the problem. However, this variable is embedded in the kernel and it is hard to optimise directly on it. We want to optimise over σ : given C, find the largest σ leading to a zero training error. This way, we benefit from the C bounding while keeping the hard-margin restriction. This is expressed in system 3 and its dual form 4.

$$\left\{\begin{array}{ll} \max\limits_{\sigma} & \sigma \\ \text{s.t.} & \sum\limits_{i=1}^{\ell} \widetilde{\xi}_i = 0 \\ \text{with} & \widetilde{\xi} = \underset{f,b,\xi}{\text{argmin}} \quad \frac{1}{2}\|f_\sigma\|^2 + C\sum\limits_{i=1}^{\ell}\xi_i \\ & \left\{\begin{array}{ll} \text{s.t.} & y_i(f_\sigma(\mathbf{x}_i)+b) \geq 1-\xi_i; \\ \text{and} & \xi_i \geq 0;\ i=1,\ell \end{array}\right. \end{array}\right. \tag{3}$$

The dual form is :

$$\left\{\begin{array}{ll} \max\limits_{\sigma} & \sigma \\ \text{s.t.} & \widetilde{\alpha}_i < C\ ;\ i=1,\ell \\ \text{and} & \widetilde{\alpha} = \underset{\alpha}{\text{argmax}}\ -\frac{1}{2}\alpha^\top G_\sigma \alpha + \alpha^\top \mathbf{1} \\ & \left\{\begin{array}{ll} \text{s.t.} & \mathbf{y}^\top \alpha = 0 \\ \text{and} & 0 \leq \alpha_i \leq C;\ i=1,\ell \end{array}\right. \end{array}\right. \tag{4}$$

with $G_\sigma(i,j) = y_i y_j k_\sigma(x_i, x_j)$

3.2 The algorithm

There are several ways to solve system (4). The most straight forward way simply consists in alternating steps, starting from a small σ (which can be set easily as much smaller than the grid precision) and increasing it as long as no α_i reaches C. However this does not bring anything from a complexity point of view and it is basically a line search. Moreover, this process starts from the largest solution in terms of support vectors and cannot be applied to large datasets. We propose to change of point of view. Instead of increasing σ as long as the solution match the constraints, we decrease it as long as the constraints are violated, starting from a large σ. By formulating the problem this way, we face two questions : 1/ how to initialise σ_0 so that $\sigma_0 > \sigma_{optimal}$? 2/ how to discard σ efficiently if it is too large ?

3.2.1 How to initialise σ_0 ?

The first question can be addressed when points are located on a grid. Indeed, we know the distance between points and we can use it to evaluate the locality of a kernel for a given bandwidth. The kernel value between two points represents their proximity. In the SVM function, the kernel values are computed between a support vector and an other point. There, this value can be interpreted as the proportion of the α_i allowed to influence the second point. We consider here the "allowed influence" of a support vector on its direct neighbours which are at most at a distance of $\sqrt{D}\epsilon$ where D is the dimension of the space containing the grid.

Let $kmax$ be the maximal acceptable kernel value for two neighbours. It is necessarily between 0 and 1 since the studied local kernels lie between 0 and 1.

$$\|x_1 - x_2\|^2 = \epsilon^2 D \tag{5}$$

$$k(x_1, x_2) \leq kmax \tag{6}$$

For Gaussian kernel :

$$\sigma \leq \sqrt{\frac{-\epsilon^2 D}{2\log(kmax)}} \tag{7}$$

influence k	σ Gauss.
0.5	$0.8493\sqrt{D}\epsilon$
0.75	$1.3183\sqrt{D}\epsilon$
0.95	$3.1222\sqrt{D}\epsilon$
0.99	$7.0533\sqrt{D}\epsilon$

Even though it's not obvious to fix $kmax$ *a priori*, we observe that the magnitude of σ lies between $0.8\sqrt{D}\epsilon$ and $7\sqrt{D}\epsilon$. Expressing the kernel bandwidth this way presents the advantage to automatically taking into account the dimension and the precision of the grid. We fix $\sigma_0 = 7\sqrt{D}\epsilon$.

3.2.2 How to discard σ efficiently ?

We claim here that we don't need to solve entirely each SVM before discarding it. Indeed, as soon as a support vector is bounded, we know the hard-margin constraint is violated and we can stop this optimisation and decrease σ. We propose the HM-BALK (Hard-Margin Bandwidth Autosetting SVM for Local Kernels), based on the SimpleSVM solver (see appendix 6). Since the SimpleSVM makes a full optimisation of the current set of support vectors at each step, bounded support vector are quickly found. Moreover, the worst classified points are added first to the set of support vectors, hence potentially bounded support vectors are treated first.

3.2.3 Implementation details

From the original algorithm, the projection step is modified. In the SimpleSVM projection step, the current solution is projected inside the admissible set if constraints ($0 \leq \alpha_i \leq C$) are violated and the violating point is removed from the working group I_w. In BALK, we still project the solution when $\alpha_i < 0$ but not if $\alpha_i > C$: any time a support vector gets a α_i larger than C, the bandwidth is reduced and the violating point remains in the working group (see algorithm 1). To do so we define a simple decrease rule : $\sigma = max(\sigma - \delta * \sigma, 10^{-3})$ with $0 < \delta < 1$. One can imagine other rules here but this simple one already turns out to give good results.

Algorithm 1 HM-BALK

Input : data x, y, C
Initialise $I_w \leftarrow i, j$ with $y_i \neq y_j$.
repeat
  Compute α_{I_w}
  if $\exists \alpha_i \leq 0$ **then**
    Project solution in the admissible set, remove i from I_w
  else if $\exists \alpha_i \geq C$ **then**
    Reduce bandwidth $\sigma = max(\sigma - \delta * \sigma, 10^{-3})$
  else
    Select next point to add to I_w
  end if
until No point can be added to I_w

4 Soft-margin BALK

In practice, labelling errors can occur and for this reason, we need to provide a soft-margin version of BALK. Another motivation is that we plan to extend BALK to non grid datasets (see section 5.3) which are often noisy. Generarely, extending SVM from hard-margin to soft-margin consists in tolerating errors for training points, which is equivalent to putting a bound on the Lagrange multipliers associated to the support vectors. The problem here is that we already put such a bound on α_i to reduce the bandwidth. We designed the SM-BALK (Soft-Margin BALK) such that α_i are double bounded. In a first step, they are bounded by C_2 in order to autoset σ and in a second step, they are bounded by $C_1 \leq C_2$ corresponding to tolerated errors.

$$\begin{cases} \min\limits_{f,\xi,b} & \frac{1}{2}\|f_{\sigma^\star}\|^2 + C_1 \sum\limits_{i=1}^{\ell} \xi_i \\ \text{s.t.} & y_i(f_{\sigma^\star}(\mathbf{x}_i) + b) \geq 1 - \xi_i; \quad i = 1, \ell \\ \text{and} & \xi_i \geq 0; \quad i = 1, \ell \\ \text{with} & \sigma^\star = \underset{\sigma}{\text{argmax}}\, \text{HM-BALK}(C_2) \\ \text{and} & C_1 \leq C_2 \end{cases} \tag{8}$$

Values for C_1 and C_2 can be significantly different since they are not linked to the same aspect of the learning task. C_2, for bandwidth adjusting, can be large while C_1, for regularisation, can be relatively small, similarly to what is usually done.

4.1 The algorithm

The idea is to combine two behaviours in order to solve only one SVM. We monitor α_i values. Anytime one reaches C_2, the bandwidth is decreased. If all are under C_2, we check if one reaches C_1. If so, the point is bounded. Doing so, we combine the two objectives (autosetting the bandwidth and tolerating training errors) at the same time (see algorithm 2).

Algorithm 2 SM-BALK

Input : data x, y, C_1, C_2
Initialise $I_w \leftarrow i, j$ with $y_i \neq y_j$.
repeat
  Compute α_{I_w}
  if $\exists \alpha_i \leq 0$ **then**
    Project solution in the admissible set, remove i from I_w and put it to I_0
  else if $\exists \alpha_i \geq C_2$ **then**
    Reduce bandwidth $\sigma = max(\sigma - \delta * \sigma, 10^{-3})$
  else if $\exists \alpha_i \geq C_1$ **then**
    Project solution, move i to I_C and remove it from I_w
  else
    Select next point to add to I_w
  end if
until No point can be added to I_w

4.2 Comments on the complexity of BALK

Our method enables to set the SVM bandwidth during learning. Compared to cross validation or other external loop based method, BALK has one level of complexity less. However, this has obviously a cost on the SVM. The main over-cost is the need for re-computing kernel elements when changing the bandwidth. The overall complexity of BALK is somewhere between the classic SVM without kernel caching and SVM with kernel caching. Experimentally, we observed that the overcost is relatively low, as shown on figure 2 for a series of experiments for datasets from 600 to 10000 points (generated as described in the next section). On this figure we report HM-BALK training time and the equivalent HM-SVM trained with the BALK selected σ.

5 Experiments

We tested both versions of BALK on artificial and real datasets. We present here results in 2 and 3 dimensions for visualisation purposes and up to 6D grids.

5.1 Artificial datasets

Artificial shapes are generated based on simple rules. We first create a complete grid. Then we pick a few random points in the space. Then, for each point of the grid, we compute the sum of the distance to its closest neighbours among the random points. If the results is above a threshold, the label is positive, otherwise it is negative. This way we can generate arbitrarily smooth or complex shapes in any dimensions, depending on the number of random points and neighbours, and arbitrarily large or small shapes depending on the threshold. Keeping the random points in memory let us test any point of the space if needed. The incomplete grid is obtained by removing from the database all the points that don't have an opposite neighbour at a distance of at most $2\sqrt{D}\epsilon$.

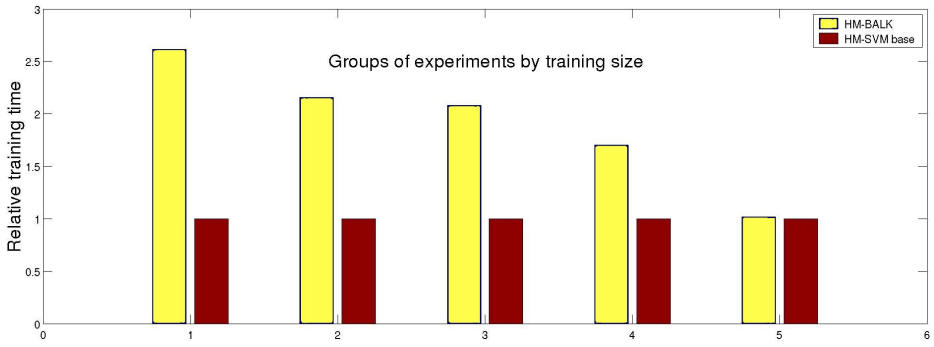

FIG. 2 – Training time comparison between BALK and SVM trained with BALK selected σ, both in hard margin versions. We give average training time for each dataset size on 20 runs. The variance on the results is negligible. Results are shown relatively to a baseline of 1 sec. of training for SVM corresponding to x sec. for BALK. Datasets are generated as described in section 5 in 2 dimensions. In the first group, dataset contain 441 points. The others have respectively 676, 1156, 2601 and 10201 training points. We see that the relative overcost decreases when the database increases.

5.1.1 Holes

Figure 3 shows the effect of a too large bandwidth and what we obtain by using the HM-BALK. The complete grid in 3D contains 1331 points and it remains around 867 after filtering. The cost of the first result would be 1 and the second would be 0.

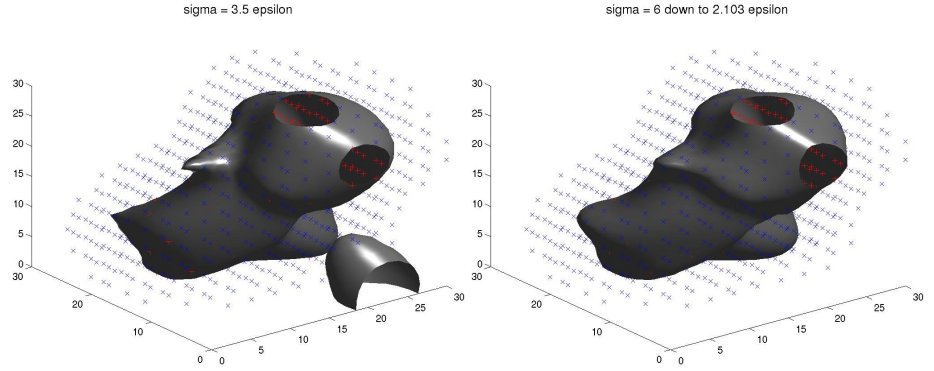

FIG. 3 – Results of hard-margin SVM with a too large bandwidth (left) and the autoset bandwidth (right). A "large α hole" is present on the left, not on the right.

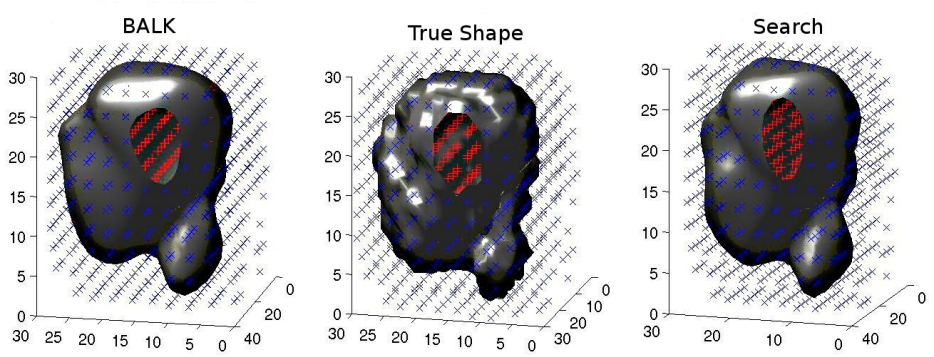

FIG. 4 – Comparing results between automatic setting (left figure), setting chosen based on the test of a range of values (right figure) and the real shape (middle one).

5.1.2 Shape approximation

Figure 4 compares the obtained results between the HM-BALK and an external loop for evaluation. We also provide a visualisation for the real generated shape. We observe that both give similar results.

5.1.3 When the grid dimension increases

In this series of experiments, the still use the shape generator to create databases. We run systematically the HM-BALK on datasets of increasing dimensions, from 2D to 6D. For each dimension, we generate 20 shapes to learn. Databases contain from 1000 to 4000 training points. On figure 5 we show the average results for 3 learning schemes. The first bars are the results obtained by HM-BALK with $C = 20000/D$ and $\sigma_0 = 7\sqrt{D}\epsilon$. The second bars are the results obtained by a range search (from $\sigma = 0.8$ to $7\sqrt{D}\epsilon$) with hard-margin SVM, taking each time the σ giving the lowest error rate on a test set. The third bars are the results obtained as the second ones but selecting the largest σ for which no hole is detected on a test set. Error rates and holes detection for all experiments are obtained by testing points of a regular grid of smallest precision $\epsilon_t = 0.8\epsilon$.

From these experiment, we observe that BALK is able to provide stable results, as good as a line search can do. Moreover, solution sizes stay is reasonable proportion of the training size. However, we cannot guaranty that no hole of the second category appears and it happens for some configurations. We can only say that the smallest C, the fewer holes. Having said that, it turns out that for those experiments, holes are avoided in more than 99% of the cases. We also see that expressing σ depending on the grid dimension and precision effectively permits to set σ independently of the grid.

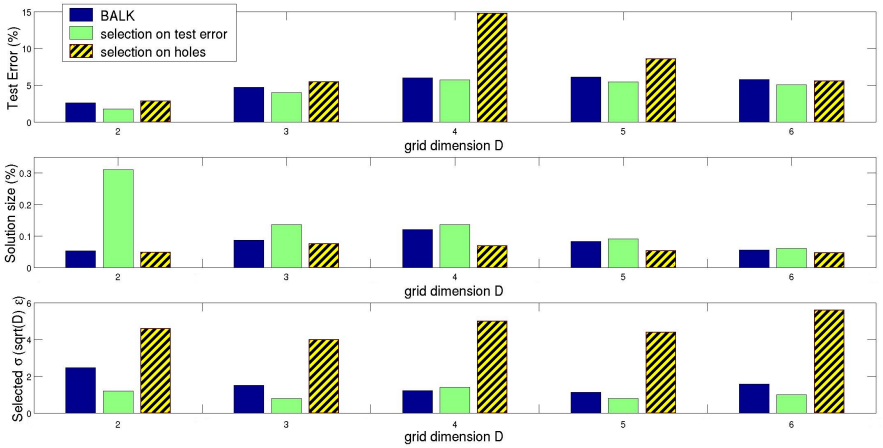

FIG. 5 – This figure shows results for grid dimensions from 2 to 6, and training sizes from 1000 to 4000 points. The top bars are the average error rates. The middle ones are the average proportion of support vectors. The bottom ones are the average selected σ. For each, the first bars report results of HM-BALK with $C = 20000/D$ and $\sigma_0 = 7\sqrt{D}\epsilon$, the second bars are given by selecting σ with the lowest test error rate and the third bars are obtained by selecting the largest σ for which no hole is detected on a test set. For second and third bars, we chose σ in $[0.8 : 0.6 : 7]$.

5.2 Viability kernel datasets

Datasets used here are generated from the problem called *car on the hill*. It is a capture basin problem, *i.e.* the goal is to reach a target while staying in the admissible constraints and it can be seen as an extension of viability problems (see Chapel & Deffuant (2007) for more details). The two dimensions of the grid are the position and the velocity of the car. The question is "what are the starting points in the space of the grid for which the car can reach the top of the hill (the target) in a given time t ?" The general algorithm is roughly as follows :

- start from the target and put all points inside the target as positive examples, others are negative
- learn the SVM function $f(x)$
- make a control step based on $f(x)$ which determines which negative points can reach the positive target in a given time step
- start again with the current obtained labels until the accumulated time step makes t

At each iteration of the algorithm, the quality of the learned SVM conditions the quality of the control and all errors are cumulative. We show on figures 6 how we can improve the quality of the results by using better setting of the kernel. For this experiment, we asked the practitioner to provide the obtained database for each step of the dynamic resolution of the car on the hill, with 10000 points on the 2D grid. We also

asked for the SVM used setting in order to be able to reproduce the results. Soft-margin SVM with large C are used in this application in order to smooth results since there may be some labelling errors. Then we applied SM-BALK on each database (representing each step) to compare the frontiers. On figure 6, we draw only the obtained frontier of each step and the bounded support vectors. The top figure shows the practitioner SVM ($C = 30000$, $\sigma = 15.8\sqrt{D}\epsilon$) and the bottom one the SM-BALK results ($C_2 = 30000$, $C_1 = 5000$). This example is particularly interesting for illustration purposes. Indeed, we see that the practitioner guess leads to solutions with many bounded support vectors. However we know that for those viability problems, the precision of the frontiers are decisive for the labelling of the next step and for the control. Hence using BALK in this context would greatly improve the precision of the dynamic process. Let us note here that we could not test the impact of BALK over several steps since the provided software for this problem is not yet able to use different bandwidth at each time step.

5.3 Application to non grid datasets

In this section we present some results obtained on images dataset : the USPS dataset. This dataset contains 7291 training examples in dimension 256 (images of 16×16 pixels, grey levels) and 2007 testing examples. Those data is not located on grids hence we face the problem of setting the first σ value. We use a simple heuristic which consists in computing the average euclidean distance d between examples of opposite classes, selected randomly. We set the initial σ to $\sigma_0 = 2d$. With the soft-margin BALK, we easily obtained error rates close to those reported in literature for Gaussian kernels on the non modified database, down to 4.2%. Therefore we also studied the effect of the choice on the C_2 value. It turns out that this method doesn't seem to be too much sensitive to it (the error rates goes from 4.2% to 5% when varies from 50 to 5000). However we still need to make this choice more reasonable and this point is under current research.

6 Conclusion

We have presented a novel algorithm that autosets the local kernel bandwidth of the SVM. This algorithm called BALK is dedicated to problems where data is coordinates on an incomplete grid. We have given various examples of applications in 2 and 3 dimensions for grids, approximating shapes and viability kernels. There remains the setting of C to investigate. The influence is limited for our task since we are not trying to obtain the lowest test error but only to avoid holes while keeping the solution sparse enough. From our observations, it appears that preventing α_i from being larger that 5000 is enough. However, a more prudent policy could set smaller C. Moreover, preliminary results indicate that it could be applied to general datasets as well. This last assumption requires more tests and justifications and is the subject of current research. A lousy argument in favour of this generalisation is that $\max_\sigma \sigma$ is equivalent to $\max_\sigma \|f_\sigma\|^2$.

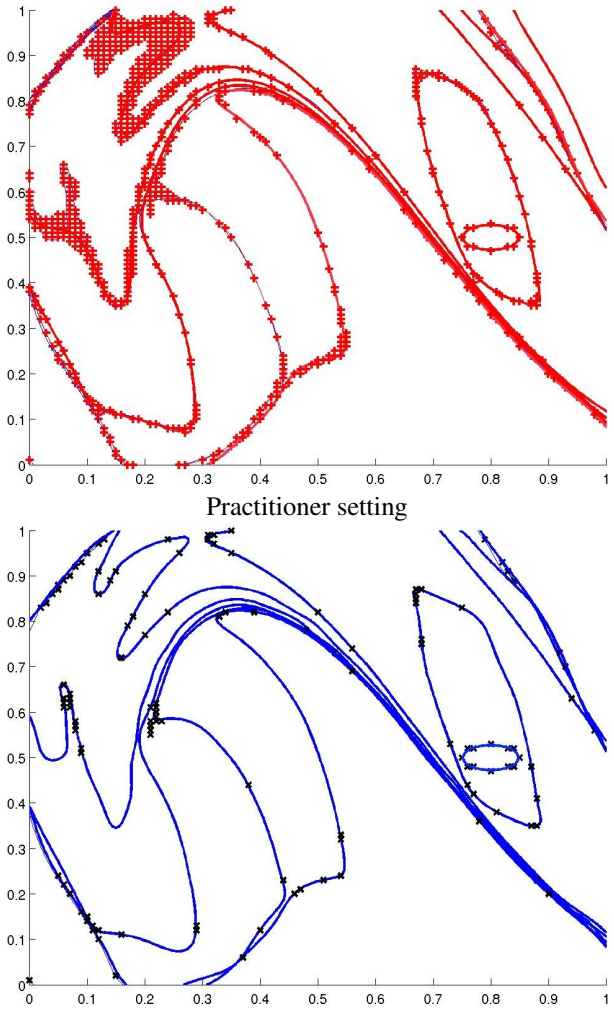

FIG. 6 – Comparing results between practitioner guess setting (top figure) and automatic setting (bottom figure) for several steps of a viability kernel approximation problem. The represented points are only the bounded support vectors. Recall that this problem is theoretically separable and should not need such points. However the control step is submitted to imprecision problems and may induce wrong labels. The first figure shows many bounded support vectors, hence many potentially misclassified points. The effect is that frontier may be far away from the real one. Such errors imply control errors at the next step. We see that the BALK leads to far better results.

Références

AUBIN J.-P. (1991). *Viability theory*. Cambridge, MA, USA : Birkhauser Boston Inc.

BENGIO Y., DELALLEAU O. & LE ROUX N. (2005). *The curse of dimentionality for lacal kernels machines*. Rapport interne.

BURBIDGE R. (2002). Adaptive kernels for support vector classification. In *MSRI-Workshop on non-linear estimation and classification, Lecture notes in Computer Science* : Springer-Verlag.

BURGES C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, **2**(2), 121–167.

CHAPEL L. & DEFFUANT G. (2007). SVM viability controller active learning : application to bike control. *proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*.

CRISTIANINI N., CAMPBELL C. & SHAWE-TAYLOR J. (1999). Dynamically adapting kernels in support vector machines. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, p. 204–210, Cambridge, MA, USA : MIT Press.

DEFFUANT G., CHAPEL L. & MARTIN S. (2007). Approximating viability kernels with support vector machines. *IEEE Transactions on automatic control*, (52), 933–937.

DEHEEGER F. & LEMAIRE M. (2007). Support vector machine for efficient subset simulations : ^{2}smart method. In *Proc. 10th Int. Conf. on Applications of Stat. and Prob. In Civil Engineering,ICASP 10*.

KEERTHI S. S., SINDHWANI V. & CHAPELLE O. (2007). An efficient method for gradient-based adaptation of hyperparameters in svm models. In B. SCHÖLKOPF, J. PLATT & T. HOFFMAN, Eds., *Advances in Neural Information Processing Systems 19*, p. 673–680. Cambridge, MA : MIT Press.

LAGOUDAKIS M. G. & PARR R. (2003). Reinforcement learning as classification : Leveraging modern classifiers. *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*.

LOOSLI G. (2004). Fast svm toolbox in Matlab and Octave based on SimpleSVM algorithm. `asi.insa-rouen.fr/~gloosli/simpleSVM.html`.

RAKOTOMAMONJY A., BACH F., CANU S. & GRANDVALET Y. (2007). More efficiency in multiple kernel learning. In *ICML '07 : Proceedings of the 24th international conference on Machine learning*, p. 775–782, New York, NY, USA : ACM.

VISHWANATHAN S., SMOLA A. J. & MURTY M. N. (2003). SimpleSVM. In *ICML*, p. 760–767.

WANG G., YEUNG D.-Y. & LOCHOVSKY F. H. (2007). A kernel path algorithm for support vector machines. In *ICML*, p. 951–958.

Appendix

SimpleSVM

We briefly recall here the dual problem and the SimpleSVM solver Vishwanathan *et al.* (2003); Loosli (2004). The dual problem of the SVM is as follows :

$$\left\{ \begin{array}{ll} \max\limits_{\boldsymbol{\alpha}\in\mathbb{R}^n} & -\frac{1}{2}\boldsymbol{\alpha}^\top G\boldsymbol{\alpha} + \mathbf{1}^\top\boldsymbol{\alpha} \\ & \mathbf{y}^\top\boldsymbol{\alpha} = 0 \\ & 0 \leq \alpha_i \leq C \qquad i \in [1, ..., n] \end{array} \right. \tag{9}$$

where G is the kernel matrix such that $G_{ij} = y_i k(\mathbf{x}_i, \mathbf{x}_j) y_j$. The SimpleSVM solver uses groups of points :

- I_0 for non support vectors : $\alpha_i = 0$
- I_w for non bounded support vectors : $0 < \alpha_i < C$
- I_C for bounded support vectors : $\alpha_i = C$

Each step of the SimpleSVM consists in defining the groups and solving a linear system to compute α_i in group I_w only.