Efficient Sparse Approximation of Support Vector Machines Solving a Kernel Lasso

Marcelo Aliquintuy[1], Emanuele Frandi[2], Ricardo Ñanculef[1(✉)], and Johan A.K. Suykens[2]

[1] Department of Informatics, Federico Santa María University, Valparaíso, Chile
{maliq,jnancu}@inf.utfsm.cl
[2] ESAT-STADIUS, KU Leuven, Leuven, Belgium
{efrandi,johan.suykens}@esat.kuleuven.be

Abstract. Performing predictions using a non-linear support vector machine (SVM) can be too expensive in some large-scale scenarios. In the non-linear case, the complexity of storing and using the classifier is determined by the number of support vectors, which is often a significant fraction of the training data. This is a major limitation in applications where the model needs to be evaluated many times to accomplish a task, such as those arising in computer vision and web search ranking.

We propose an efficient algorithm to compute sparse approximations of a non-linear SVM, i.e., to reduce the number of support vectors in the model. The algorithm is based on the solution of a Lasso problem in the feature space induced by the kernel. Importantly, this formulation does not require access to the entire training set, can be solved very efficiently and involves significantly less parameter tuning than alternative approaches. We present experiments on well-known datasets to demonstrate our claims and make our implementation publicly available.

Keywords: SVMs · Kernel methods · Sparse approximation · Lasso

1 Introduction

Non-linear support vector machines (SVMs) are a powerful family of classifiers. However, while in recent years one has seen considerable advancements on scaling kernel SVMs to large-scale problems [1,9], the lack of sparsity in the obtained models, i.e., the often large number of support vectors, remains an issue in contexts where the run time complexity of the classifier is a critical factor [6,13]. This is the case in applications such as object detection in images or web search ranking, which require repeated and fast evaluations of the model. As the sparsity of a non-linear kernel SVM cannot be known *a-priori*, it is crucial to devise efficient methods to impose sparsity in the model or to sparsify an existing classifier while preserving as much of its generalization capability as possible.

Recent attempts to achieve this goal include *post-processing approaches* that reduce the number of support vectors in a given SVM or change the basis used

© Springer International Publishing AG 2017
C. Beltrán-Castañón et al. (Eds.): CIARP 2016, LNCS 10125, pp. 208–216, 2017.
DOI: 10.1007/978-3-319-52277-7_26

to express the classifier [3,12,15], and *direct methods* that modify the SVM objective or introduce heuristics during the optimization to maximize sparsity [2,6,7,11,14]. In a recent breakthrough, [3] proposed a simple technique to reduce the number of support vectors in a given SVM showing that it was asymptotically optimal and outperformed many competing approaches in practice. Unfortunately, most of these techniques either depend on several parameter and heuristic choices to yield a good performance or demand significant computational resources. In this paper, we argue how these problems can be effectively circumvented by sparsifying an SVM solving a simple Lasso problem [4] in the kernel space. Interestingly, this criterion was already mentioned in [12], but was not accompanied by an efficient algorithm neither systematically assessed in practice. By exploiting recent advancements in optimization [4,5,9], we devise an algorithm that is significantly cheaper than [3] in terms of optimization and parameter selection but is competitive in terms of the accuracy/sparsity tradeoff.

2 Problem Statement and Related Work

Given data $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ with $\mathbf{x}_i \in X$ and $y_i \in \{\pm 1\}$, SVMs learn a predictor of the form $f_{\mathbf{w},b}(\mathbf{x}) = \text{sign}(\mathbf{w}^T\phi(\mathbf{x}) + b)$ where $\phi(\mathbf{x})$ is a feature vector representation of the input pattern $\mathbf{x}$ and $\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}$ are the model parameters. To allow more flexible decision boundaries, $\phi(\mathbf{x})$ often implements a non-linear mapping $\phi : X \to \mathcal{H}$ of the input space into a Hilbert space $\mathcal{H}$, related to X by means of a *kernel function* $k : X \times X \to \mathbb{R}$. The kernel allows to compute dot products in $\mathcal{H}$ directly from X, using the property $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$, $\forall\, \mathbf{x}_i, \mathbf{x}_j \in X$. The values of $\mathbf{w}, b$ are determined by solving a problem of the form

$$\mathbf{min}_{\mathbf{w},b}\ \tfrac{1}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 + C\sum\nolimits_{i=1}^{m} \ell\left(y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b)\right)^p, \tag{1}$$

where $p \in \{1, 2\}$ and $\ell(z) = (1 - z)_+$ is called *the hinge-loss*. It is well-known that the solution $\mathbf{w}*$ to (1) can be written as a linear combination of the training patterns in the feature space $\mathcal{H}$. This leads to the "kernelized" decision function

$$f_{\mathbf{w},b}(\mathbf{x}) = \text{sign}\left(\mathbf{w}^{*T}\phi(\mathbf{x}) + b^*\right) = \text{sign}\left(\sum\nolimits_{i=1}^{n} y_i\beta_i^* k(\mathbf{x}_i, \mathbf{x}) + b^*\right), \tag{2}$$

whose run time complexity is determined by the number n_{sv} of examples such that $\beta_i^* \neq 0$. These examples are called *the support vectors* (SVs) of the model. In contrast to the linear case ($\phi(\mathbf{x}) = \mathbf{x}$), kernel SVMs need to explicitly store and access the SVs to perform predictions. Unfortunately, it is well known that in general, n_{sv} grows as a linear function of the number of training points [6,13] (at least all the misclassified points are SVs) and therefore n_{sv} is often too large in practice, leading to classifiers expensive to store and evaluate. Since n_{sv} is the number of non-zero entries in the coefficient vector $\boldsymbol{\beta}^*$, this problem if often referred in the literature as the *lack of sparsity* of non-linear SVMs.

Methods to address this problem can be categorized in two main families: *post-processing* or *reduction methods*, which, starting with a non-sparse classifier, find a more efficient predictor preserving as much of the original predictive

accuracy as possible, and *direct methods* that modify the training criterion (1) or introduce heuristics during its optimization to promote sparsity. The first category include methods selecting a subset of the original support vectors to recompute the classifier [12,15], techniques to substitute the original support vectors by arbitrary points of the input space [10] and methods tailored to a specific class of SVM [8]. The second category includes offline [6,7,11], as well as online learning algorithms [2,14]. Unfortunately, most of these techniques either incur in a significant computational cost or depend on several heuristic choices to yield a good performance. Recently, a simple, yet asymptotically optimal reduction method named ISSVM has been presented in [3], comparing favorably with the state of the art in terms of the accuracy/sparsity tradeoff. The method is based on the observation that the hinge loss of a predictor $f_{\mathbf{w},b}$ can be approximately preserved using a number of support vectors proportional to $\|\mathbf{w}\|_{\ell_2}$ by applying sub-gradient descent to the minimization of the following objective function

$$g_{\text{ISSVM}}(\tilde{\mathbf{w}}) = \max_{i:h_i>0} \left(h_i - y_i \left(\tilde{\mathbf{w}}^T \mathbf{x} + b\right)\right) \ , \tag{3}$$

where $h_i = \max(1, y_i(\mathbf{w}^T\mathbf{x} + b))$. Using this method to sparsify an SVM $f_{\mathbf{w},b}$ guarantees a reduction of n_{sv} to at most $\mathcal{O}(\|\mathbf{w}\|_{\ell_2})$ support vectors. However, since different levels of sparsification may be required in practice, the algorithm is equipped with an additional projection step. In the course of the optimization, the approximation $\tilde{\mathbf{w}}$ is projected into the ℓ_2-ball of radius δ, where δ is a parameter controlling the level of sparsification. Unfortunately, the inclusion of this projection step and the weak convergence properties of sub-gradient descent makes the algorithm quite sensitive to parameter tuning.

3 Sparse SVM Approximations via Kernelized Lasso

Suppose we want to sparsify an SVM with parameters $\mathbf{w}_*, b_*$, kernel $k(\cdot,\cdot)$ and support set $S = \{(\mathbf{x}_{(i)}, y_{(i)})\}_{i=1}^{n_{\text{sv}}}$. Let $\phi : X \to \mathcal{H}$ be the feature map implemented by the kernel and $\phi(\mathbf{S})$ the matrix whose i-th column is given by $\phi(\mathbf{x}_{(i)})$. With this notation, $\mathbf{w}_*$ can be written as $\mathbf{w}_* = \phi(\mathbf{S})\boldsymbol{\alpha}_*$ with $\boldsymbol{\alpha}_* \in \mathbb{R}^{n_{\text{sv}}}$[1]. In this paper, we look for approximations of the form $\mathbf{u} = \phi(\mathbf{S})\boldsymbol{\alpha}$ with sparse $\mathbf{u}$. Support vectors such that $u_{(i)} = 0$ are pruned from the approximation.

Our approximation criterion is based on two observations. The first is that the objective function (3) can be bounded by a differentiable function which is more convenient for optimization. Importantly, this function also bounds the expected loss of accuracy incurred by the approximation. Indeed, the following result (whose proof we omit for space constraints) holds:

[1] Note that we have just re-indexed the support vectors in (2) to make the model independent of the entire training set and defined $\boldsymbol{\alpha}_j = y_j\beta_j$ for notational convenience.

Proposition 1. *Consider an SVM implementing the decision function* $f_{\mathbf{w},b}(\mathbf{x}) = sign(\mathbf{w}^T\phi(\mathbf{x}) + b)$ *and an alternative decision function* $f_{\mathbf{u},b}(\mathbf{x}) = sign(\mathbf{u}^T\phi(\mathbf{x})+b)$, *with* $\mathbf{u} \in \mathcal{H}$. *Let* $\ell(z)$ *be the hinge loss. Then,* $\exists M > 0$ *such that*

$$(i)\ g_{ISSVM}(\mathbf{u}) \leq M \left\|\mathbf{u} - \mathbf{w}\right\|_{\mathcal{H}}, \quad (ii)\ E\left(\ell\left(yf_{\mathbf{u}}(\mathbf{x})\right) - \ell\left(yf_{\mathbf{w}}(\mathbf{x})\right)\right) \leq M \left\|\mathbf{u} - \mathbf{w}\right\|_{\mathcal{H}}.$$

The result above suggests that we can substitute $\mathbf{w} \in \mathcal{H}$ in the original SVM by some $\mathbf{u} \in \mathcal{H}$ such that $\|\mathbf{u} - \mathbf{w}_*\|^2$ is small. However, the obtained surrogate does need to be sparse. Indeed, minimizing $\|\mathbf{u} - \mathbf{w}_*\|^2$ in $\mathcal{H}$ trivially yields the original predictor $\mathbf{w}_*$ which is generally dense. We thus need to restrict the search to a family of sparser models. Our second observation is that a well-known, computationally attractive and principled way to induce sparsity is ℓ_1-norm regularization, i.e., constraining $\mathbf{u}$ to lie in a ball around 0 with respect to the norm $\|\mathbf{u}\|_{\ell_1} = \sum_i |u_i|$. Thus, we approach the task of sparsifying the SVM by solving a problem of the form

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^{n_{\text{SV}}}} \tfrac{1}{2}\|\phi(\mathbf{S})\boldsymbol{\alpha} - \mathbf{w}_*\|^2 \text{ s.t. } \|\boldsymbol{\alpha}\|_{\ell_1} \leq \delta, \tag{4}$$

where δ is a regularization parameter controlling the level of sparsification. The obtained problem can be easily recognized as a kernelized Lasso with response variable $\mathbf{w}_*$ and design matrix $\phi(\mathbf{S})$. By observing that

$$\begin{aligned} \|\mathbf{w}_* - \phi(\mathbf{S})\boldsymbol{\alpha}\|^2 &= \mathbf{w}_*^T\mathbf{w}_* - 2\boldsymbol{\alpha}_*^T\phi(\mathbf{S})^T\phi(\mathbf{S})\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\phi(\mathbf{S})^T\phi(\mathbf{S})\boldsymbol{\alpha} \\ &= \mathbf{w}_*^T\mathbf{w}_* - 2\boldsymbol{\alpha}_*^T\mathbf{K}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} = \mathbf{w}_*^T\mathbf{w}_* - 2\mathbf{c}^T\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha}, \end{aligned} \tag{5}$$

where $\mathbf{c} = \mathbf{K}\boldsymbol{\alpha}_*$, it is easy to see that solving (4) only requires access to the kernel matrix (or the kernel function):

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^{n_{\text{SV}}}} g(\boldsymbol{\alpha}) = \tfrac{1}{2}\boldsymbol{\alpha}^T\mathbf{K}\boldsymbol{\alpha} - \mathbf{c}^T\boldsymbol{\alpha} \text{ s.t. } \|\boldsymbol{\alpha}\|_{\ell_1} \leq r. \tag{6}$$

This type of approach has been considered, up to some minor differences, by Schölkopf *et al.* in [12]. However, to the best of our knowledge, it has been largely left out of the recent literature on sparse approximation of kernel models. One possible reason for this is the fact that the original proposal had a high computational cost, making it unattractive for large models. We reconsider this technique arguing that recent advancements in Lasso optimization make it possible to solve the problem efficiently using high-performance algorithms with strong theoretical guarantees [4]. Importantly, we show in Sect. 5 that this efficiency is not obtained at the expense of accuracy, and indeed the method can match or even surpass the performance of the current state-of-the-art methods.

Algorithm. To solve the kernelized Lasso problem, we adopt a variant of the Frank-Wolfe (FW) method [5], an iterative greedy algorithm to minimize a convex differentiable function $g(\boldsymbol{\alpha})$ over a closed convex set Σ, specially tailored to handle large-scale instances of (6). This method does not require to compute the matrix $\mathbf{K}$ beforehand, is very efficient in practice and enjoys important convergence guarantees [5,9], some of which are summarized in Theorem 1. Given an

Algorithm 1. SASSO: Sparsification of SVMs via Kernel Lasso.

$\boldsymbol{\alpha}^{(0)} \leftarrow \mathbf{0}$, $\mathbf{g}^{(0)} = \mathbf{c}$.
for $k = 1, 2, \ldots$ **do**
  Find a descent direction: $j_*^{(k)} \leftarrow \arg\max_{j \in J} |g_j^{(k)}|$, $t_*^{(k)} \leftarrow t \operatorname{sign}\left(g_{j_*}^{(k)}\right)$.
  Choose a step-size $\lambda^{(k)}$, e.g. by a line-search between $\boldsymbol{\alpha}^{(k)}$ and $\mathbf{u}^{(k)}$.
  Update the solution: $\boldsymbol{\alpha}^{(k+1)} \leftarrow (1 - \lambda^{(k)})\boldsymbol{\alpha}^{(k)} + \lambda^{(k)} t_*^{(k)} \mathbf{e}_{j_*^{(k)}}$.
  Update the gradient: $g_j^{(k+1)} \leftarrow (1 - \lambda^{(k)}) g_j^{(k+1)} + \lambda^{(k)} t_*^{(k)} K_{j j_*^{(k)}}$ $\forall j \in [n_{\text{sv}}]$.
end

iterate $\boldsymbol{\alpha}^{(k)}$, a step of FW consists in finding a descent direction as

$$\mathbf{u}^{(k)} \in \underset{\mathbf{u} \in \Sigma}{\operatorname{argmin}} (\mathbf{u} - \boldsymbol{\alpha}^{(k)})^T \nabla g(\boldsymbol{\alpha}^{(k)}), \tag{7}$$

and updating the current iterate as $\boldsymbol{\alpha}^{(k+1)} = (1 - \lambda^{(k)})\boldsymbol{\alpha}^{(k)} + \lambda^{(k)}\mathbf{u}^{(k)}$. The step-size $\lambda^{(k)}$ can be determined by an exact line-search (which can be done analytically for quadratic objectives) or setting it as $\lambda^{(k)} = 1/(k+2)$ as in [5].

In the case of problem (6), where Σ corresponds to the ℓ_1-ball of radius t in $\mathbb{R}^{n_{\text{sv}}}$ (with vertices $\mathcal{V} = \{\pm t\mathbf{e}_i : i = 1, 2, \ldots, n_{\text{sv}}\}$) and the gradient is $\nabla g(\boldsymbol{\alpha}) = \mathbf{K}\boldsymbol{\alpha} - \mathbf{c}$, it is easy to see that the solution of (7) is equivalent to

$$j* = \arg\max_{j \in [n_{\text{sv}}]} \left|\phi(\mathbf{s}_j)^T \phi(\mathbf{S})\boldsymbol{\alpha} + c_j\right| = \arg\max_{j \in [n_{\text{sv}}]} \left|\sum\nolimits_{i:\alpha_i \neq 0} \alpha_i K_{ij} + c_j\right|. \tag{8}$$

The adaptation of the FW algorithm to problem (6) is summarized in Algorithm 1 and is referred to as SASSO in the rest of this paper.

Theorem 1. *Consider problem (6) with* $r \in (0, \|\boldsymbol{\alpha}^*\|_{\ell_1})$. *Algorithm 1 is monotone and globally convergent. In addition, there exists* $C > 0$ *such that*

$$\|\mathbf{w}_* - \phi(\mathbf{S})\boldsymbol{\alpha}^{(k)}\|^2 - \|\mathbf{w}_* - \phi(\mathbf{S})\boldsymbol{\alpha}^{(k+1)}\|^2 \leq C/(k+2). \tag{9}$$

Tuning of b. We have assumed above that the bias b of the SVM can be preserved in the approximation. A slight boost in accuracy can be obtained by computing a value of b which accounts for the change in the composition of the support set. For the sake of simplicity, we adopt here a method based on a validation set, i.e., we define a range of possible values for b and then choose the value minimizing the misclassification loss on that set. It can be shown that it is safe (in terms of accuracy) to restrict the search to the interval $[b_{\min}, b_{\max}]$ where

$$b_{\min} = \inf_{\mathbf{x} \in S : \mathbf{w}^T\mathbf{x} > 0} -\mathbf{w}^T\mathbf{x}\,, \qquad b_{\max} = \sup_{\mathbf{x} \in S : \mathbf{w}^T\mathbf{x} < 0} -\mathbf{w}^T\mathbf{x}.$$

4 Experimental Results

We present experiments on four datasets recently used in [2,3] to assess SVM sparsification methods: Adult (a8a), IJCNN, TIMIT and MNIST. Table 1 summarizes the number of training points m and test points t for each dataset. The SVMs to sparsify were trained using SMO with a RBF kernel and parameters set up as in [2,3]. As discussed in Sect. 2, we compare the performance of our algorithm with that of the ISSVM algorithm, which has a publicly available C++ implementation [3]. Our algorithms have been also coded in C++. We executed the experiments on a 2 GHz Intel Xeon E5405 CPU with 20 GB of main memory running CentOS, without exploiting multithreading or parallelism in computations. The code, the data and instructions to reproduce the experiments of this paper are publicly available at https://github.com/maliq/FW-SASSO.

We test two versions of our method, the standard one in Algorithm 1, and an aggressive variant employing a *fully corrective* FW solver (where an internal optimization over the current active set is carried out at each iteration, see

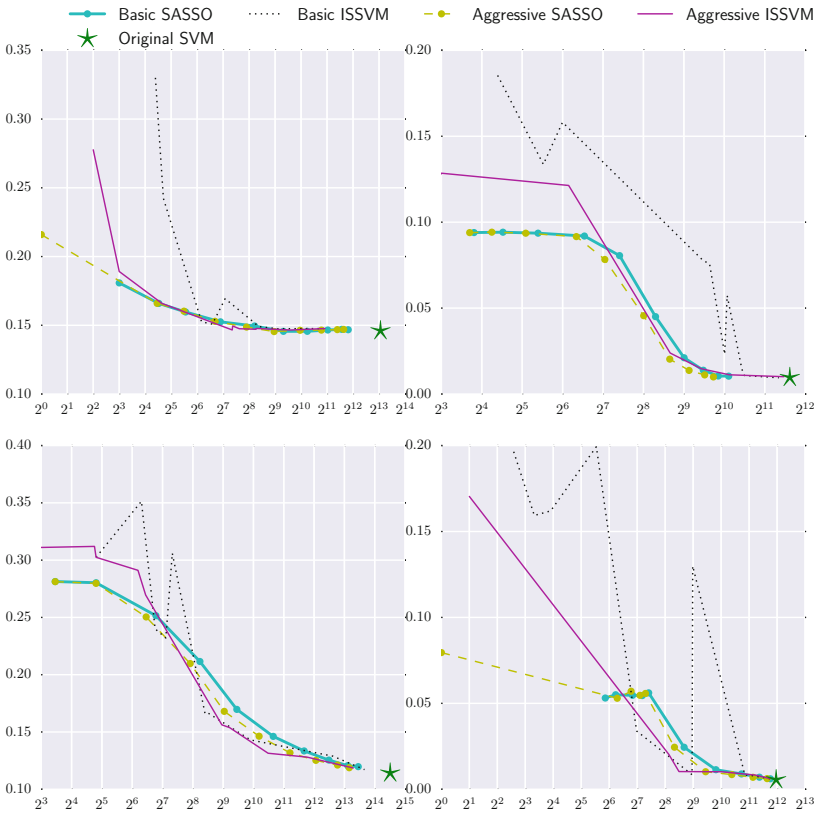

Fig. 1. Test accuracy (y axis) versus sparsity (number of support vectors, x axis). From top-left to bottom-right: Adult, IJCNN, TIMIT and MNIST datasets.

Table 1. Time required to build the sparsity/accuracy path. We report the total time incurred in parameter selection (training with different parameters and evaluation in the validation set) and the average training time to build a path.

Dataset	Method	Average train time (secs)	Total time (secs) train & val.
Adult	SASSO BASIC	2.08E+02	2.66E+02
$m = 22696$	SASSO AGG.	1.99E+02	2.51E+02
$t = 9865$	ISSVM BASIC	2.78E+03	1.97E+04
	ISSVM AGG.	1.33E+03	4.71E+04
IJCNN	SASSO BASIC	8.20E+01	2.74E+02
$m = 35000$	SASSO AGG.	4.34E+01	1.57E+02
$t = 91701$	ISSVM BASIC	4.23E+03	3.47E+04
	ISSVM AGG.	5.35E+03	1.98E+05
TIMIT	SASSO BASIC	1.57E+02	6.22E+02
$m = 66831$	SASSO AGG.	1.46E+02	5.24E+02
$t = 22257$	ISSVM BASIC	1.02E+04	7.68E+04
	ISSVM AGG.	6.99E+03	2.49E+05
MNIST	SASSO BASIC	4.03E+01	4.16E+02
$m = 60000$	SASSO AGG.	3.96E+01	3.93E+02
$t = 10000$	ISSVM BASIC	3.53E+04	3.83E+04
	ISSVM AGG.	2.77E+04	9.74E+05

e.g. [5]). The baseline comes also in two versions. The "basic" version has two parameters, namely ℓ_2-norm and η: the first controls the level of sparsity, and the second is the learning rate used for training. The "aggressive" version has an additional tolerance parameter ϵ (see [3] for details). To choose values for these parameters, we reproduced the methodology employed in [3], i.e., for the learning rate we tried values $\eta = 4^{-4}, \ldots, 4^2$ and for ϵ (in the aggressive variant) we tried values $\epsilon = 2^{-4}, \ldots, 1$. For each level of sparsity, we choose a value based on a validation set. This procedure was repeated over 10 test/validation splits.

Following previous work [2,3], we assess the algorithms on the entire sparsity/accuracy path, i.e., we produce solutions with decreasing levels of sparsity (increasing number of support vectors) and evaluate their performance on the test set. For ISSVM, this is achieved using different values of the ℓ_2-norm parameter. During the execution of these experiments, we observed that it is quite difficult to determine an appropriate range of values for this parameter. Our criterion was to set this range manually till obtaining the range of sparsities reported in the figures of [3]. For SASSO, the level of sparsity is controlled by parameter δ in (4). The maximum value for δ is easily determined as the ℓ_1-norm of the input SVM and the minimum value as 10^{-4} times the former. To make comparison fair, we compute 10 points of the path for all the methods.

Results in Fig. 1 show that the sparsity/accuracy tradeoff path obtained by SASSO matches that of the (theoretically optimal) ISSVM method [3], and often tends to outperform it on the sparsest section of the path. However as it can be seen from Table 1, our method enjoys a considerable computational advantage over ISSVM: on average, it is faster by 1–2 orders of magnitude, and the overhead due to parameter selection is marginal compared to the case of ISSVM, where the total time is one order of magnitude larger than the single model training time. We also note that the aggressive variant of SASSO enjoys a small but consistent advantage on all the considered datasets. Both versions of our method exhibit a very stable and predictable performance, while ISSVM needs the more aggressive variant of the algorithm to produce a regular path. However, this version requires considerable parameter tuning to achieve a behavior similar to that observed for SASSO, which translates into a considerably longer running time.

5 Conclusions

We presented an efficient method to compute sparse approximations of non-linear SVMs, i.e. to reduce the number of support vectors in the model. The algorithm enjoys strong convergence guarantees and it is easy to implement in practice. Further algorithmic improvements could also be obtained by implementing the stochastic acceleration studied in [4]. Our experiments showed that the proposed method is competitive with the state of the art in terms of accuracy, with a small but systematic advantage when sparser models are required. In computational terms, our approach is significantly more efficient due to the properties of the optimization algorithm and the avoidance of cumbersome parameter tuning.

Acknowledgments. E. Frandi and J.A.K. Suykens acknowledge support from ERC AdG A-DATADRIVE-B (290923), CoE PFV/10/002 (OPTEC), FWO G.0377.12, G.088114N and IUAP P7/19 DYSCO. M Aliquintuy and R. Ñanculef acknowledge support from CONICYT Chile through FONDECYT Project 1130122 and DGIP-UTFSM 24.14.84.

References

1. Bottou, L., Lin, C.J.: Support vector machine solvers. In: Bottou, L., Chapelle, O., DeCoste, D., Weston, J. (eds.) Large Scale Kernel Machines. MIT Press (2007)
2. Cotter, A., Shalev-Shwartz, S., Srebro, N.: The kernelized stochastic batch perceptron. In: Proceedings of the 29th ICML, pp. 943–950. ACM (2012)
3. Cotter, A., Shalev-shwartz, S., Srebro, N.: Learning optimally sparse support vector machines. In: Proceedings of the 30th ICML, pp. 266–274. ACM (2013)
4. Frandi, E., Nanculef, R., Lodi, S., Sartori, C., Suykens, J.A.K.: Fast and scalable Lasso via stochastic Frank-Wolfe methods with a convergence guarantee. Mach. Learn. **104**(2), 195–221 (2016)
5. Jaggi, M.: Revisiting Frank-Wolfe: projection-free sparse convex optimization. In: Proceedings of the 30th ICML, pp. 427–435. ACM (2013)

6. Joachims, T., Yu, C.N.J.: Sparse kernel SVMs via cutting-plane training. Mach. Learn. **76**(2–3), 179–193 (2009)
7. Keerthi, S.S., Chapelle, O., DeCoste, D.: Building support vector machines with reduced classifier complexity. J. Mach. Learn. Res. **7**, 1493–1515 (2006)
8. Mall, R., Suykens, J.A.K.: Very sparse LSSVM reductions for large scale data. IEEE Trans. Neural Netw. Learn. Syst. **26**(5), 1086–1097 (2015)
9. Ñanculef, R., Frandi, E., Sartori, C., Allende, H.: A novel Frank-Wolfe algorithm. Analysis and applications to large-scale SVM training. Inf. Sci. **285**, 66–99 (2014)
10. Nguyen, D., Ho, T.: An efficient method for simplifying support vector machines. In: Proceedings of the 22nd ICML, pp. 617–624. ACM (2005)
11. Nguyen, D.D., Matsumoto, K., Takishima, Y., Hashimoto, K.: Condensed vector machines: learning fast machine for large data. IEEE Trans. Neural Netw. **21**(12), 1903–1914 (2010)
12. Schölkopf, B., Mika, S., Burges, C.J., Knirsch, P., Müller, K.R., Rätsch, G., Smola, A.J.: Input space versus feature space in kernel-based methods. IEEE Trans. Neural Netw. **10**(5), 1000–1017 (1999)
13. Steinwart, I.: Sparseness of support vector machines. J. Mach. Learn. Res. **4**, 1071–1105 (2003)
14. Wang, Z., Crammer, K., Vucetic, S.: Breaking the curse of kernelization: budgeted stochastic gradient descent for large-scale SVM training. J. Mach. Learn. Res. **13**(1), 3103–3131 (2012)
15. Zhan, Y., Shen, D.: Design efficient support vector machine for fast classification. Pattern Recogn. **38**(1), 157–161 (2005)