Training very large scale nonlinear SVMs using Alternating Direction Method of Multipliers coupled with the Hierarchically Semi-Separable kernel approximations

S. Cipolla [*1] and J.Gondzio [† 1]

[1] The University of Edinburgh, School of Mathematics

ERGO Technical Report 21-005
August 9, 2021

Abstract

Typically, nonlinear Support Vector Machines (SVMs) produce significantly higher classification quality when compared to linear ones but, at the same time, their computational complexity is prohibitive for large-scale datasets: this drawback is essentially related to the necessity to store and manipulate large, dense and unstructured kernel matrices. Despite the fact that at the core of training a SVM there is a *simple* convex optimization problem, the presence of kernel matrices is responsible for dramatic performance reduction, making SVMs unworkably slow for large problems. Aiming to an efficient solution of large-scale nonlinear SVM problems, we propose the use of the *Alternating Direction Method of Multipliers* coupled with *Hierarchically Semi-Separable* (HSS) kernel approximations. As shown in this work, the detailed analysis of the interaction among their algorithmic components unveils a particularly efficient framework and indeed, the presented experimental results demonstrate a significant speed-up when compared to the *state-of-the-art* nonlinear SVM libraries (without significantly affecting the classification accuracy).

1 Introduction

Support vector machine (SVM) is one of the most well-known supervised classification method which has been extensively used in different fields. At its core, training nonlinear SVMs classifier boils down to a solution of a convex Quadratic Programming (QP) problem whose running time

[*]Email: scipolla@ed.ac.uk
[†]Email: j.gondzio@ed.ac.uk

heavily depends on the way the quadratic term *interacts* with the chosen optimizer. Typically, such interaction, is represented by the solution of a linear system involving the quadratic term (perhaps in some suitably modified version). However, in the nonlinear SVM case, the quadratic term involves a kernel matrix which (except for the linear kernel) is a dense and unstructured matrix. Solving (or merely storing) a linear system involving such matrices may result in unworkably slow algorithms for large scale problems. Although the use of kernel approximations in SVMs classification has been for a long time a relevant research question, see Section 1.1 for references, the existing structured approximations are not always able to capture the *essential features* of the kernel (see, once more, Section 1.1 for a detailed explanation of this statement) and, moreover, the selected structure for the kernel approximation may not be exploitable by the chosen optimizer. Aim of this work is to devise a computational framework based on the use of the *Alternating Direction Method of Multipliers* (ADMM) [5] coupled with *Hierarchically Semi-Separable* (HSS) [6] kernel approximations. Indeed, the latter choice, if on one hand allows to produce kernel approximations essentially in *a matrix-free* regime and with guaranteed accuracy [9], on the other, allows the efficient solution of (shifted) linear systems involving it. In turn, when QP problems are solved using ADMM, the solution of shifted kernel linear systems is the main expensive computational task. Such a harmonized interaction between the kernel approximation and the optimizer not only allows a fast training phase but also makes possible a fast grid search for optimal hyperparameters selection through caching the HSS approximation/factorization.

1.1 Background

Support vector machines (SVMs) [3, 11] are useful and widely used classification methods. Training a nonlinear SVM has at its core (in its dual form) the solution of the following convex quadratic optimization problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^d} \; & f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T Y K Y \mathbf{x} - \mathbf{e}^T\mathbf{x} \\ \text{s.t. } & \mathbf{y}^T\mathbf{x} = 0, \\ & x_i \in [0, C] \text{ for all } i = 1, \ldots, d, \end{aligned} \tag{1}$$

where $y_i \in \{-1, 1\}$ are target labels, $Y := diag(\mathbf{y})$, $K_{ij} := K(\mathbf{f}_i, \mathbf{f}_j)$ is a Positive Definite Kernel [21, Def. 3], $\mathbf{f}_i \in \mathbb{R}^r$ are feature vectors and $\mathbf{e}$ is the vector of all ones.

Once a solution $\bar{\mathbf{x}}$ of problem (1) has been computed, the classification function for an unlabelled data $\mathbf{f}$ can be determined by

$$\tilde{y} = sign(\sum_{i=1}^{d} y_i \bar{x}_i K(\mathbf{f}_i, \mathbf{f}) + b).$$

The bias term b is computed using the support vectors that lie on the margins, i.e., considering j s.t. $0 < \bar{x}_j < C$, the following formula is used:

$$b = \sum_{i=1}^{d} y_i \bar{x}_i K(\mathbf{f}_i, \mathbf{f}_j) - y_j. \tag{2}$$

Despite their simplicity, when compared with Neural Networks (NNs), nonlinear SVMs are still recognised by practitioners of Machine Learning and Data Science as the preferred choice for classification tasks in some situations. In particular, the community seems to widely agree on the fact that NNs are not efficient on low-dimensional input data because of their huge overparametrisation and, in this case, SVMs may represent the *state of the art* for classification. Indeed, SVMs have only two hyperparameters (say the choice of a kernel-related parameter h and the penalization constant C), so they are very easy to tune to specific problems: the parameter tuning is usually performed by a simple grid-search through the parameter space.

On the other hand, even if the SVM training is related to a convex optimization problem for which there exist efficient solution methods, training SVMs for large scale datasets may be a computationally challenging option essentially due to the fact that, in order to be able to use the Kernel Trick, SVMs cache a value for the kernelized "distance" between any two pairs of points: for this reason an $O(d^2)$ storage requirement is to be expected. In general, without any particular *specialization*, training SVMs is unworkably slow for sets beyond, say, 10^4 datapoints.

Without any doubts, the most successful class of methods designed to handle storage difficulties, is represented by decomposition methods [15, 23, 30, 31]: unlike most optimization methods which update all the variables in each step of an iterative process, decomposition methods modify only a subset of these at every iteration leading, hence, to a small sub-problem to be solved in each iteration. A prominent example in this class is represented by [8] which delivers, somehow, a standard benchmark comparison in the SVMs training panorama. It is important to note at this stage that since only few variables are updated per iteration, for difficult/large-scale problems, decomposition methods may suffer from a slow convergence.

On the other hand, an alternative way to overcome storage issues is to approximate the kernel matrix K and, indeed, there is a rich literature concerned with the acceleration of kernel methods which are usually based on the efficient approximation of the kernel map. The most popular approach is to construct a low-rank matrix approximation of the kernel matrix reducing the arithmetic and storage cost [12, 13, 16, 18–20, 25, 26, 35, 45]. We mention explicitly Nyström-type methods [18, 24, 41] and random feature maps to approximate the kernel function directly [32] or as a preconditioner [1]. However, the numerical rank of the kernel matrix depends on parameters, which are, in turn, data-dependent: the Eckart–Young–Mirsky theorem, see [39, Sec. 2.11.1] justifies low rank approximations only when the kernel matrix is characterized by a sufficiently fast decay of the singular values. For example, the Gaussian kernel matrix, i.e., $K_{ij} = \exp^{-\frac{\|\mathbf{f}_i - \mathbf{f}_i\|^2}{2h^2}}$, is approximately low-rank only if $h > 0$ is sufficiently large (see the left panel in Figure 1 for an example) but, for classification purposes, a small value of h may be required.

Several methods were proposed to overcome the fact that K is not necessarily approximately low-rank. The main idea, in this context, relies on the initial splitting of the data into clusters,

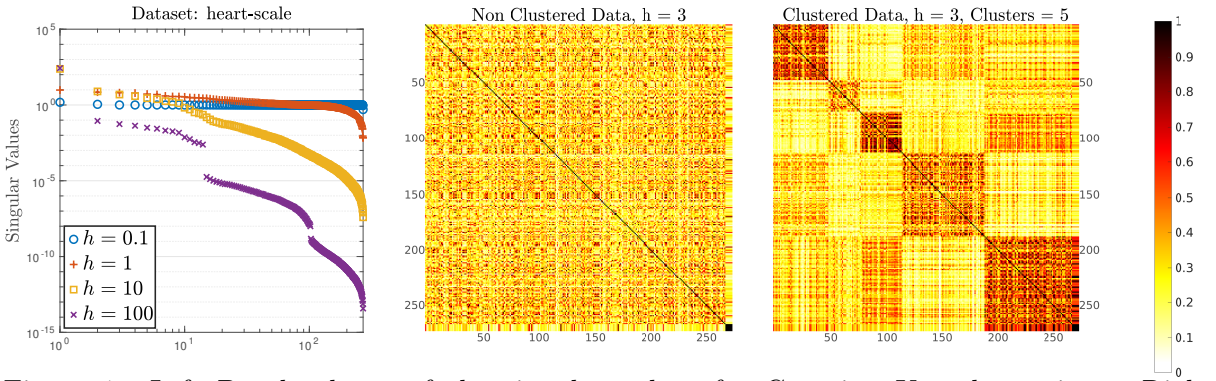

Figure 1: Left Panel: decay of the singular values for Gaussian Kernel matrices. Right Panel: Gaussian Kernel matrices obtained with/without preliminary data clustering. Dataset: `heart_scale` [8].

so that between-classes interactions in the kernel matrix may be represented/well approximated by either sparse or low-rank matrices [36, 40, 44] (see right panel in Figure 1 for a pictorial representation of this idea).

1.2 Contribution

Aim of this work is to propose and analyse the use of the *Hierarchically Semi-Separable* (HSS) matrix representation [6] for the solution of large scale kernel SVMs. Indeed, the use of HSS approximations of kernel matrices has been already investigated in [9, 33] for the solution of large scale Kernel Regression problems. The main reason for the choice of the HSS structure in this context can be summarised as follows:

- using the STRUctured Matrix PACKage (`STRUMPACK`) [34] it is possible to obtain HSS approximations of the kernel matrices without the need to store/compute explicitly the whole matrix K. Indeed, for kernel matrix approximations, `STRUMPACK` uses a partially matrix-free strategy (see [9]) essentially based on an adaptive randomized sampling which requires only a black-box matrix-times-vector multiplication routine and the access to selected elements from the kernel matrix;

- in the preprocessing step employed in `STRUMPACK`, clustering algorithms are employed to find groups of points with large inter-group distances and small intra-group distances. This feature permits to fully exploit the underlying data geometry to obtain valuable algebraic approximations of the kernel matrix;

- the resulting approximations allow fast approximate kernel matrix computations with linear scalability for the computation of matrix-vector products and solution of linear systems.

In particular, we trace the main contribution of this work in unveiling a particularly efficient interaction between the HSS structure and ADMM [5] in the SVMs case, see Section 2. When problem (1) is suitably reformulated in a form exploitable by ADMM, just the solution of one linear system involving the (shifted) kernel matrix is required per ADMM iteration: kernel matrices approximated using the HSS structure allow highly efficient solutions of such linear systems. Indeed, in this framework, approximating the kernel matrix with an HSS structure (h fixed) results in a highly efficient optimization phase for a fixed value of C (see Section 3.3). It is important to note, moreover, that the computational footprint related to the kernel matrix approximation phase is fully justified by the fact that the same approximation can be *reused* for training the model with different values of C; this feature makes our proposal particularly attractive when a fine grid is used for the tuning of the penalization parameter C. It is important to note, at this stage, that also the works [22, 43] analyse the use of ADMM for SVMs: in [43] ADMM has been used to solve linear SVMs with feature selection whereas in [22] a hardware-efficient nonlinear SVM training algorithm has been presented in which the Nyström approximation is exploited to reduce the dimension of the kernel matrices. Both works represent and use, somehow, different frameworks and techniques from those presented here.

2 The computational framework

Problem (1) can be written as follows:

$$\begin{aligned} \min_{\mathbf{x},\, \mathbf{z}\in\mathbb{R}^d} \ & \frac{1}{2}\mathbf{x}^T Y K Y \mathbf{x} - \mathbf{e}^T\mathbf{x} + I_{\mathbf{y}^T\mathbf{x}=0}(\mathbf{x}) + I_{[0,C]}(\mathbf{z}) \\ \text{s.t. } & \mathbf{x} - \mathbf{z} = 0, \end{aligned} \tag{3}$$

where, for a given subset $S \subset \mathbb{R}^d$, $I_S(\mathbf{x})$ is the indicator function of the set S, defined as

$$I_S(\mathbf{x}) := \begin{cases} 0 & \text{if } \mathbf{x} \in S \\ +\infty & \text{if } \mathbf{x} \notin S. \end{cases}$$

The Augmented Lagrangian corresponding to (3) reads as

$$\mathcal{L}_\beta(\mathbf{x}, \mathbf{z}, \boldsymbol{\mu}) = \frac{1}{2}\mathbf{x}^T Y K Y \mathbf{x} - \mathbf{e}^T\mathbf{x} + I_{\mathbf{y}^T\mathbf{x}=0}(\mathbf{x}) + I_{[0,C]}(\mathbf{z}) - \boldsymbol{\mu}^T(\mathbf{x} - \mathbf{z}) + \frac{\beta}{2}\|\mathbf{x} - \mathbf{z}\|^2. \tag{4}$$

Reformulation (3) with an extra copy of variable x makes it easier to exploit partial separability and facilitates a direct application of ADMM to solve it. Indeed, ADMM [5] is our choice of an (efficient) solution technique for problem (3). In Algorithm 1 we summarize its main steps:

1 **for** $k = 0, 1, \dots$ **do**
2 $\quad \mathbf{x}^{k+1} = \min_{\mathbf{x}\in\mathbf{R}^d} \mathcal{L}_\beta(\mathbf{x}, \mathbf{z}^k, \boldsymbol{\mu}^k)$; `/* x minimization */`
3 $\quad \mathbf{z}^{k+1} = \min_{\mathbf{z}\in\mathbf{R}^d} \mathcal{L}_\beta(\mathbf{x}^{k+1}, \mathbf{z}, \boldsymbol{\mu}^k)$; `/* z minimization */`
4 $\quad \boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k - \beta(\mathbf{x}^{k+1} - \mathbf{z}^{k+1})$; `/* Multiplier Update */`
5 **end**

Algorithm 1: ADMM

2.1 ADMM details

Let us observe that the solution of the problem in Line 2 of Algorithm 1 is equivalent to the solution of the problem

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^d} \ & \frac{1}{2}\mathbf{x}^T Y \underbrace{(K + \beta I)}_{=:K_\beta} Y\mathbf{x} - \underbrace{(\mathbf{e} + \boldsymbol{\mu}^k + \beta\mathbf{z}^k)^T}_{=:\mathbf{q}^k}\mathbf{x} \\ \text{s.t. } & \mathbf{y}^T\mathbf{x} = 0. \end{aligned} \tag{5}$$

Writing the KKT conditions of problem (5), i.e.,

$$\begin{bmatrix} YK_\beta Y & -\mathbf{y} \\ -\mathbf{y}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{e} + \boldsymbol{\mu}^k + \beta\mathbf{z}^k \\ 0 \end{bmatrix},$$

and eliminating the variable λ, it is possible to write its solution in a closed form:

$$\mathbf{x}^{k+1} = YK_\beta^{-1}Y\mathbf{q}^k - \frac{\mathbf{e}^T K_\beta^{-1} Y\mathbf{q}^k}{\mathbf{e}^T K_\beta^{-1}\mathbf{e}} YK_\beta^{-1}\mathbf{e},$$

where we used the fact that $Y\mathbf{y} = \mathbf{e}$. Moreover, the problem at Line 3 of Algorithm 1 can be written alternatively as

$$\arg\min_{\mathbf{z}\in[0,C]} \ g(\mathbf{z}) := \frac{\beta}{2}\mathbf{z}^T\mathbf{z} - \beta\mathbf{z}^T\mathbf{x}^{k+1} + \mathbf{z}^T\boldsymbol{\mu}^k,$$

which also has a closed-form solution (see [2, Example 2.2.1]):

$$\mathbf{z}^{k+1} = \Pi_{[0,C]}(\mathbf{x}^{k+1} - \frac{1}{\beta}\boldsymbol{\mu}^k), \tag{6}$$

where $\Pi_{[0,C]}$ is the component-wise projection onto the interval $[0, C]$. Summarizing the observations carried out in this section, we observe that Algorithm 1 can be written in closed form as in Algorithm 2:

for $k = 0, 1, \ldots$ **do**

$\mathbf{x}^{k+1} = YK_\beta^{-1}Y\mathbf{q}^k - \frac{\mathbf{e}^T K_\beta^{-1} Y \mathbf{q}^k}{\mathbf{e}^T K_\beta^{-1}\mathbf{e}} YK_\beta^{-1}\mathbf{e}$; /* x minimization */

$\mathbf{z}^{k+1} = \Pi_{[0,C]}(\mathbf{x}^{k+1} - \frac{1}{\beta}\boldsymbol{\mu}^k)$; /* z minimization */

$\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k - \beta(\mathbf{x}^{k+1} - \mathbf{z}^{k+1})$; /* Multiplier Update */

end

Algorithm 2: Closed form ADMM for problem (3)

2.1.1 Computational Cost and Convergence

Algorithm 2 requires the solution of a linear system involving the matrix K_β at every iteration (the vector $YK_\beta^{-1}\mathbf{e}$ can be precomputed) plus a series of operations of linear complexity. Moreover, since Algorithm 2 is a particular instance of ADMM, it is convergent, see [5].

3 Experiments

3.1 Hierarchically Semi-Separable matrix representation

As already pointed out previously, one of the main computational issues associated with problem (1) relates to the fact that the matrix K is usually dense and of large dimension: the cubic computational complexity of application and the quadratic storage requirements for kernel matrices limit the applicability of kernel methods for SVM in large scale applications. To overcome this problem many different approaches have been proposed in literature, see the discussion in Section 1. The one we decide to take into account here, is the Hierarchically Semi-Separable (HSS) approximation of the kernel matrix in the form proposed in [9]. In general, the HSS approximation of a given matrix uses a hierarchical block 2×2 partitioning of the matrix where all off-diagonal blocks are compressed, or approximated, using a low-rank product [6]. The accurate description of such techniques in the case of kernel matrices is out of the scope of this work and, for this reason, we refer the reader to [9, Sec. II.B – II.C] for the full details. Instead, for our purposes, we mention explicitly the peculiarities of the approach prosed in [9] (HSS-ANN) which have driven our choice:

- it uses a randomized sampling [28] which requires just a partially matrix-free method: the construction process involves only the exploitation of (multiple) matrix-times-vector products and access to individual elements of K;
- the matrix-times-vector operation is further approximated using a column sampling based on a Approximate Nearest Neighbours (ANN) of the data points [27, 42] which uses the similarity between them to identify the dominating entries of the kernel matrix, see [9, Sec. II.B];

- the overall complexity of the HSS-ANN construction (excluding the preprocessing phase on the data) is $O(r^2 d)$ where r is the maximum HSS rank, i.e., the maximum rank over all off-diagonal blocks in the HSS hierarchy, see [9, Sec. II.C and Alg. 3]. The storage complexity of HSS-ANN is $O(dr)$;
- after the construction, the (shifted) HSS kernel matrix approximation can be factorized into a ULV form [7], where L is lower triangular and U and V are orthogonal. This factored form, computed just once for fixed h in our approach, can be used to solve linear systems involving the (shifted) kernel matrix.

3.2 Implementation details

In Algorithm 3 we summarise the pseudo-code of our implementation. It is based on `STRUMPACK` library (`Version 5.1.0`) [17, 37], which provides efficient routines for the approximation $\tilde{K}$ of a kernel matrix K (see Line 1 of Algorithm 3). Moreover, once $\tilde{K}$ is obtained, it provides efficient routines for the solution of linear systems of the form $\tilde{K}_\beta \mathbf{x} = \mathbf{b}$ through the exploitation of a ULV factorization (see Line 3 of Algorithm 3). It is worth noting that for a fixed kernel value h the approximation $\tilde{K}$ and the factorization ULV of $\tilde{K}_\beta$ are computed just once and then *reused* for all the values C in the grid search.

It is also important to note that in practice the bias b is obtained averaging over all the support vectors that lie on the margin instead of using equation (2). Indeed, defining $M := \{j \mid 0 < \bar{x}_j < C\}$ and $\bar{e}_j = 1$ if $j \in M$ or $\bar{e}_j = 0$ otherwise, the bias b is often computed using

$$b = \frac{1}{|M|} \sum_{j \in M} (\sum_{i=1}^{d} y_i \bar{x}_i K(\mathbf{f}_i, \mathbf{f}_j) - y_j) = \frac{1}{|M|} (\bar{\mathbf{x}}_{\mathbf{y}}^T K \bar{\mathbf{e}} - \sum_{j \in M} y_j), \tag{7}$$

where $(\bar{\mathbf{x}}_{\mathbf{y}})_j := y_j \bar{x}_j$. If the full kernel matrix K is not available, computing (7) may be time consuming for large datasets since it requires a series of kernel evaluations. On the other hand, the right-hand side of equation (7) suggests that if an approximation $\tilde{K}$ of K is available for which matrix vector products can be *inexpensively* evaluated, the bias computation requires exactly just one matrix vector product and one scalar product. This is indeed the case when an HSS approximation of the kernel matrix is available and we exploit this property in our implementation, see Line 17 in Algorithm 3.

Finally, to conclude this section, we address briefly the problem of relating the solution $\tilde{\mathbf{x}}$ of the approximated SVM problem

$$\begin{aligned} \min_{\mathbf{x} \in \mathbb{R}^d} \ & \tilde{f}(\mathbf{x}) := \frac{1}{2} \mathbf{x}^T Y \tilde{K} Y \mathbf{x} - \mathbf{e}^T \mathbf{x} \\ \text{s.t. } & \mathbf{y}^T \mathbf{x} = 0, \\ & x_i \in [0, C] \text{ for all } i = 1, \ldots, d, \end{aligned} \tag{8}$$

Input: K kernel function, h kernel parameter, β ADMM parameter, $F_{train} \in \mathbb{R}^{r\times d}$, $\mathbf{y}_{train} \in \mathbb{R}^d$, $F_{test} \in \mathbb{R}^{r\times m}$, $\mathbf{y}_{test} \in \mathbb{R}^m$ – training and testing data.

$\tilde{K} = \texttt{HSScompression}(K(F_{train}, F_{train}), h)$;
$\tilde{K}_\beta = \tilde{K} + \beta I$;
$[U, L, V] = \texttt{ULVfactorization}(\tilde{K}_\beta)$;
$\mathbf{w} = (ULV)^{-1}\mathbf{e}$;
$w_1 = \mathbf{e}^T\mathbf{w}$;
$\mathbf{w} = Y_{train}\mathbf{w}$;
for $C \in \{C_1, \ldots, C_{max}\}$ **do**
  Initialize $\mathbf{x}_0, \mathbf{z}_0, \boldsymbol{\mu}_0$;
  for $k = 0, 1, \ldots, MaxIt$ **do**
    $w_2 = \mathbf{w}^T\mathbf{x}^k$;
    $\mathbf{x}^{k+1} = Y(ULV)^{-1}Y\mathbf{x}^k - \frac{w2}{w1}\mathbf{w}$; /* x minimization */
    $\mathbf{z}^{k+1} = \Pi_{[0,C]}(\mathbf{x}^{k+1} - \frac{1}{\beta}\boldsymbol{\mu}^k)$; /* z minimization */
    $\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k - \beta(\mathbf{x}^{k+1} - \mathbf{z}^{k+1})$; /* Multiplier Update */
  end
  Define $\mathbf{z_y} = Y_{train}\mathbf{z}^{MaxIt}$; /* Computing Bias */
  Define $\bar{\mathbf{e}}_j = 1$ if $0 < (\mathbf{z}_{MaxIt})_j < C$ or $\bar{\mathbf{e}}_j = 0$ otherwise ;
  $b = \frac{1}{\|\bar{\mathbf{e}}\|_1}(\mathbf{z_y}^T\tilde{K}\bar{\mathbf{e}} - \sum_{j:\bar{\mathbf{e}}_j\neq 0}(\mathbf{y}_{train})_j)$;
  for $j = 1, \ldots, m$ **do**
    $(\tilde{\mathbf{y}}_{test})_j = sign(\sum_{i=1}^d (\mathbf{z_y})_i K((\mathbf{f}_{train})_i, (\mathbf{f}_{test})_j) + b)$; /* Label Assignement */
  end
end

Algorithm 3: SVM training/testing using Strumpack and ADMM

to the solution $\bar{\mathbf{x}}$ of the original problem (1). Indeed, using a similar technique to the one presented in [16, Sec. 4.1.], for any unitary invariant form we obtain

$$\begin{aligned} |f(\bar{\mathbf{x}}) - \tilde{f}(\tilde{\mathbf{x}})| &\leq \max\{\frac{1}{2}|\tilde{\mathbf{x}}^T Y(\tilde{K} - K)Y\tilde{\mathbf{x}}|, \frac{1}{2}|\bar{\mathbf{x}}^T Y(K - \tilde{K})Y\bar{\mathbf{x}}|\} \\ &\leq \frac{1}{2}\max\{\|\tilde{\mathbf{x}}\|^2, \|\bar{\mathbf{x}}\|^2\}\|\tilde{K} - K\|. \end{aligned} \tag{9}$$

Using the boundedness of $\frac{1}{2}\max\{\|\tilde{\mathbf{x}}\|^2, \|\bar{\mathbf{x}}\|^2\}$, we obtain that for $\tilde{K} \to K$ it holds $\tilde{f}(\tilde{\mathbf{x}}) \to f(\bar{\mathbf{x}})$. Equation (9) suggests that for increasingly accurate approximations $\tilde{K}$ of K, the accuracy classification performance of the *approximate* SVM classifier (8) matches increasingly closely the accuracy classification performance of the *exact* SVM classifier (1). Nonetheless, we will show experimentally, that this may be also true when quite poor approximations are used, see

Table 4 in the following section.

3.3 Numerical results

Our code is written in `C++` and the numerical experiments are performed on a Dell PowerEdge R920 machine running Scientific Linux 7 and equipped with Four Intel Xeon E7-4830 v2 2.2GHz, 20M Cache, 7.2 GT/s QPI, Turbo (4x10Cores) 256 GB RAM.

Dataset	Features	Training Set Dim.	$\|Train_+\|$	Test Set Dim.	$\|Test_+\|$
a8a	122	22696	5506	9865	2335
w7a	300	24692	740	25057	739
rcv1.binary	47236	20242	10491	135480	71326
a9a	122	32561	7841	16281	3846
w8a	300	49749	1479	14951	454
ijcnn1	22	49990	4853	91701	8712
cod.rna	8	59535	19845	271617	90539
skin.nonskin*	3	171540	135986	73517	58212
webspam.uni*	254	245000	148717	105000	63472
susy*	18	3500000	1601659	1500000	686168

Table 1: Problem Set Details. * = Test Set obtained using Random 30% of the original Training Set.

Table 1 summarizes the details for the chosen dataset. In Tables 4 and 5 we report the results obtained using our proposal for different parameters related to the accuracy of the HSS-ANN approximation (increasing accuracy) where all the other non specified HSS-ANN parameters have to be considered the default ones. In our experiments we choose, in Algorithm 3, $MaxIt = 10$ and the Gaussian Kernel function $K(\mathbf{f}_i, \mathbf{f}_j) = \exp^{-\frac{\|\mathbf{f}_i - \mathbf{f}_j\|^2}{2h^2}}$. Indeed, it is important to observe that the choice of making a prescribed number of ADMM iterations instead of using a standard stopping criterion, is motivated by the fact that for machine learning applications going for accurate optimal solution does not necessarily have to deliver the best classification accuracy. On the other hand, the fact that one choice of the ADMM parameter $MaxIt$ permits to obtain satisfactory classification accuracy for all the problems in our dataset confirms the robustness of our approach (also if we should mention the experimental observation concerning the fact that for particular problems, a different choice of this parameter may led to better classification performance). Finally, concerning the choice of the ADMM parameter β, we observed that for larger problems an increasing value of β is recommended: we chose $\beta = 10^2$ if the training size $d \in [10^4, 10^5]$, $\beta = 10^3$ if $d \in [10^5, 10^6]$ and $\beta = 10^4$ if $d \geq 10^6$.

In Table 2 we report the results obtained using `LIBSVM Version 3.25` [8], which implements specialized algorithms to address the SVM problem (LIBSVM uses a Sequential Minimal Op-

timization type decomposition method [4, 14, 31]). In Table 3 we report the results obtained using RACQP [29] (where a multi-block generalization of ADMM is employed, see also [10, 38] for related theoretical analysis).

In particular, the kernel parameter h and the penalization term C were estimated by running a grid-check when instances are solved using our proposal (the HSS-ANN accuracy parameters used are those specified in Table 5 since our proposal achieves (generally) the best classification accuracy in this case). Those pairs were then used to solve the instances with LIBSVM and RACQP. The pairs were chosen from a relatively coarse grid, $h,\ C \in \{0.1, 1, 10\}$ because the goal of this experiment is to demonstrate that although our approach uses kernel *approximations*, it can still achieve comparable classification accuracy but with a reduced runtime when compared with other algorithms for the solution of SVM problems which use the *true* kernel matrices.

The first important observation concerning Tables 4 and 5 is that, unexpectedly (see equation (9)), increasing the HSS accuracy parameters (generally) does not lead to a significant increase of classification accuracy: we obtain quite good classification accuracy despite using very rough approximations (see Table 4). The problem which benefited most an improved kernel approximation is `webspam.uni`. Indeed, the classification accuracy has increased by nearly 1% in this case. At the same time, increasing the HSS accuracy parameters adversely affects the `Compression` and `Factorization` time. It is important to note also that the `ADMM Time` needed to train the model is completely negligible when compared to the time needed to produce the HSS-ANN approximations. As was already pointed out, this feature allows for a very fast grid-search on the parameter C (for the largest considered problem it takes roughly $10s$ to train the model for a fixed C). Indeed, the choice of the parameter C may greatly affect the performance of the classification accuracy (see Figure 2 for some examples).

Concerning the comparison of our approach with LIBSVM and RACQP (compare Tables 4 and 5 with Tables 2 and 3, respectively) several remarks are in order. The first one concerns the coherence of the HSS-ANN approximations with the classification accuracy: the accuracy results obtained for the grid-selected h and C are always comparable to those obtained using LIBSVM and generally better than those obtained using RACQP (both approaches use, in different ways, the true kernel matrices). The second one concerns the computational time: for smallest problems or problems with high dimensional features, our proposal may not be the best performer (see, e.g., the problems `w7a`, `rcv1.binary` and `w8a`) but, on the contrary, when the dimension of the training set increases and the number of features is *small*, the approach proposed in this paper becomes a clear winner (see problems `ijcnn`, `cod.rna`, `webspam.uni` and `susy`): the goodness and advantages of our approach are further underpinned observing that the *total training time* needed for the grid search on the parameter C (h fixed) can be roughly obtained multiplying the values in the column `ADMM Time` by the number of grid values selected for C (in our case 3). This is not true for LIBSVM and RACQP where the training phase is *restarted from scratch* for all the values C (considering also in this case h fixed).

Finally, for the sake of fairness, concerning the comparison of running times of our proposal with those from RACQP, we should mention the fact that RACQP is implemented in Matlab,

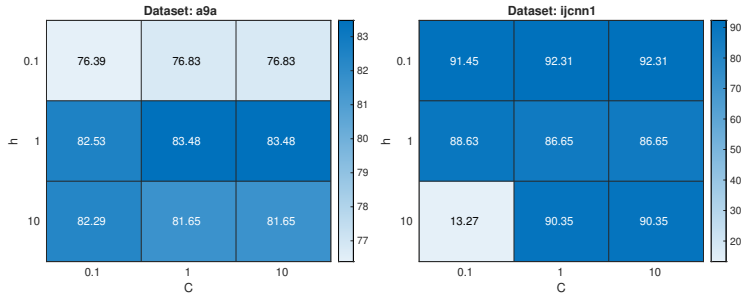

Figure 2: Heatmap of the classification accuracy for the datasets `a9a` and `ijcnn1`.

Dataset	Runtime [s]	Accuracy [%]
a8a	123.308	83.953
w7a	148.110	97.904
rcv1.binary	261.399	93.247
a9a	305.913	82.697
w8a	508.232	99.444
ijcnn1	345.805	96.007
cod.rna	110.997	90.374
skin.nonskin	344.938	99.960
webspam.uni	13354.384	99.081
susy	††	

Table 2: LIBSVM. †† = stopped after 10h.

Dataset	Runtime [s]	Accuracy [%]
a8a	98.269	79.757
w7a	82.838	97.050
rcv1.binary	67.830	71.987
a9a	206.527	82.237
w8a	348.122	97.806
ijcnn1	427.551	91.460
cod.rna	531.787	33.333
skin.nonskin	4689.815	97.649
webspam.uni	21669.329	92.830
susy	††	

Table 3: RACQP. †† = stopped after 10h.

which is presumably slower than a compiled language such as `C++`.

4 Conclusions

In this work we proposed an ADMM-based scheme (see Algorithm 3) which employs HSS-ANN approximations (see [9] and Section 3) to train SVMs. Numerical experiments obtained using `STRUMPACK` [37] in a sequential architecture, show that our proposal compares favourably with `LIBSVM` [8] and `RACQP` [29] in terms of computational time and classification accuracy when the dimension of the training set increases. Indeed, both `LIBSVM` and `RACQP` use different decomposition methods for the *exact* kernel matrix, which may be slow for large scale problems. Our proposal, instead, resorting on an *all-at-once optimal* exploitation of structured approximations of the kernel matrices, is less prone to the *curse of dimensionality* allowing us to train datasets of larger dimensions.

Acknowledgements: This work was supported by Oracle Labs.

Dataset	HSS Construction			ADMM Time [s]	Best Parameters		Accuracy [%]
	Compression [s]	Factorization [s]	Memory [MB]		h	C	
a8a	135.923	6.181	112.968	0.300	1	1,10	83.314
w7a	2161.920	14.442	99.345	0.486	1	1,10	97.465
rcv1.binary	6319.780	1.665	58.839	0.173	10	1,10	89.940
a9a	256.032	8.162	179.192	0.471	1	1,10	83.477
w8a	10476.200	107.71	273.1	1.498	1	1,10	97.679
ijcnn	9.772	1.980	153.586	0.470	0.1	1,10	92.403
cod.rna	2.900	2.863	181.47	0.444	10	0.1	89.305
skin.nonskin	1127.79	11.078	538.349	1.219	10	0.1,1,10	99.846
webspam.uni	5809.6	3.228	757.969	0.909	0.1	0.1,1,10	95.551
susy	3938.68	25.614	13599.4	9.471	1	0.1,1,10	72.338

Table 4: Strumpack&ADMM. Strumpack parameters: `hss_rel_tol`= 1, `hss_abs_tol`= 0.1, `hss_max_rank`= 200, `hss_approximate_neighbors`= 64.

Dataset	HSS Construction			ADMM Time [s]	Best Parameters		Accuracy [%]
	Compression [s]	Factorization [s]	Memory [MB]		h	C	
a8a	795.597	16.276	218.673	0.588	1	1,10	83.476
w7a	2311.330	15.229	107.393	0.621	1	1,10	97.465
rcv1.binary	14211.0	1.425	58.84	0.210	10	1,10	87.921
a9a	1176.99	21.3909	379.852	0.986	1	1,10	83.643
w8a	10774.900	124.076	296.472	1.738	1	1,10	97.672
ijcnn	21.393	2.041	168.007	0.298	0.1	1,10	92.314
cod.rna	23.242	2.377	182.424	0.280	10	1,10	89.308
skin.nonskin	1232.730	7.560	544.544	0.972	10	0.1,1,10	99.855
webspam.uni	7003.52	5.640	861.542	1.297	0.1	0.1,1,10	96.123
susy	14495.9	159.972	18264.2	15.889	1	0.1,1,10	72.047

Table 5: Strumpack&ADMM. Strumpack parameters: `hss_rel_tol`= 0.5, `hss_abs_tol`= 0.05, `hss_max_rank`= 2000, `hss_approximate_neighbors`= 512.

References

[1] H. Avron, K. L. Clarkson, and D. P. Woodruff. "Faster kernel ridge regression using sketching and preconditioning". In: *SIAM J. Matrix Anal. Appl.* 38.4 (2017), pp. 1116–1138. ISSN: 0895-4798. DOI: 10.1137/16M1105396. URL: https://doi.org/10.1137/16M1105396.

[2] D. P. Bertsekas. *Nonlinear programming*. Second. Athena Scientific Optimization and Computation Series. Athena Scientific, Belmont, MA, 1999, pp. xiv+777. ISBN: 1-886529-00-0.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 144–152.

[4] L. Bottou and C.-J. Lin. "Support vector machine solvers". In: *Large scale kernel machines* 3.1 (2007), pp. 301–320.

[5] S. Boyd, N. Parikh, and E. Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[6] S. Chandrasekaran, M. Gu, and W. Lyons. "A fast adaptive solver for hierarchically semiseparable representations". In: *Calcolo* 42.3-4 (2005), pp. 171–185. ISSN: 0008-0624. DOI: 10.1007/s10092-005-0103-3. URL: https://doi.org/10.1007/s10092-005-0103-3.

[7] S. Chandrasekaran, M. Gu, and T. Pals. "A fast ULV decomposition solver for hierarchically semiseparable representations". In: *SIAM J. Matrix Anal. Appl.* 28.3 (2006), pp. 603–622. ISSN: 0895-4798. DOI: 10.1137/S0895479803436652. URL: https://doi.org/10.1137/S0895479803436652.

[8] C.-C. Chang and C.-J. Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

[9] G. Chávez et al. "Scalable and Memory-Efficient Kernel Ridge Regression". In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2020, pp. 956–965. DOI: 10.1109/IPDPS47924.2020.00102.

[10] S. Cipolla and J. Gondzio. *Random multi-block ADMM: an ALM based view for the QP case*. 2020. arXiv: 2012.09230 [math.OC].

[11] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[12] P. Drineas and M. W. Mahoney. "On the Nyström method for approximating a Gram matrix for improved kernel-based learning". In: *J. Mach. Learn. Res.* 6 (2005), pp. 2153–2175. ISSN: 1532-4435.

[13] P. Drineas et al. “Fast approximation of matrix coherence and statistical leverage”. In: *J. Mach. Learn. Res.* 13 (2012), pp. 3475–3506. ISSN: 1532-4435.

[14] R.-E. Fan, P.-H. Chen, and C.-J. Lin. “Working set selection using second order information for training support vector machines”. In: *J. Mach. Learn. Res.* 6 (2005), pp. 1889–1918. ISSN: 1532-4435.

[15] R.-E. Fan et al. “Working set selection using second order information for training support vector machines.” In: *Journal of machine learning research* 6.12 (2005).

[16] S. Fine and K. Scheinberg. “Efficient SVM training using low-rank kernel representations”. In: *Journal of Machine Learning Research* 2.Dec (2001), pp. 243–264.

[17] P. Ghysels et al. “A robust parallel preconditioner for indefinite systems using hierarchical matrices and randomized sampling”. In: *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2017, pp. 897–906.

[18] A. Gittens and M. W. Mahoney. “Revisiting the Nyström method for improved large-scale machine learning”. In: *J. Mach. Learn. Res.* 17 (2016), Paper No. 117, 65. ISSN: 1532-4435.

[19] G. H. Golub and C. F. Van Loan. *Matrix computations*. Vol. 3. JHU Press, 2012.

[20] N. Halko, P. G. Martinsson, and J. A. Tropp. “Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions”. In: *SIAM Rev.* 53.2 (2011), pp. 217–288. ISSN: 0036-1445. DOI: 10.1137/090771806. URL: https://doi.org/10.1137/090771806.

[21] T. Hofmann, B. Schölkopf, and A. J. Smola. “Kernel methods in machine learning”. In: *Ann. Statist.* 36.3 (2008), pp. 1171–1220. ISSN: 0090-5364. DOI: 10.1214/009053607000000677. URL: https://doi.org/10.1214/009053607000000677.

[22] S.-A. Huang and C.-H. Yang. “A hardware-efficient ADMM-based SVM training algorithm for edge computing”. In: *arXiv preprint arXiv:1907.09916* (2019).

[23] T. Joachims. *Making large-scale SVM learning practical. In, B. Scholkopf, C. Burges, A. Smola,(eds.): Advances in Kernel Methods-Support Vector Learning*. 1998.

[24] S. Kumar, M. Mohri, and A. Talwalkar. “Sampling methods for the Nyström method”. In: *J. Mach. Learn. Res.* 13 (2012), pp. 981–1006. ISSN: 1532-4435.

[25] E. Liberty et al. “Randomized algorithms for the low-rank approximation of matrices”. In: *Proc. Natl. Acad. Sci. USA* 104.51 (2007), pp. 20167–20172. ISSN: 0027-8424. DOI: 10.1073/pnas.0709640104. URL: https://doi.org/10.1073/pnas.0709640104.

[26] M. W. Mahoney. “Randomized algorithms for matrices and data”. In: *Found. Trends Mach. Learn.* 123–224.3 (2011).

[27] W. B. March, B. Xiao, and G. Biros. “ASKIT: approximate skeletonization kernel-independent treecode in high dimensions”. In: *SIAM J. Sci. Comput.* 37.2 (2015), A1089–A1110. ISSN: 1064-8275. DOI: 10.1137/140989546. URL: https://doi.org/10.1137/140989546.

[28] P. G. Martinsson. "A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix". In: *SIAM J. Matrix Anal. Appl.* 32.4 (2011), pp. 1251–1274. ISSN: 0895-4798. DOI: 10.1137/100786617. URL: https://doi.org/10.1137/100786617.

[29] K. Mihic, M. Zhu, and Y. Ye. "Managing Randomization in the Multi-Block Alternating Direction Method of Multipliers for Quadratic Optimization". In: *Math. Program. Comp.* (2020). DOI: https://doi.org/10.1007/s12532-020-00192-5.

[30] E. Osuna, R. Freund, and F. Girosit. "Training support vector machines: an application to face detection". In: *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE. 1997, pp. 130–136.

[31] J Platt. *Fast training of support vector machines using sequential minimal optimization, In, B. Scholkopf, C. Burges, A. Smola,(eds.): Advances in Kernel Methods-Support Vector Learning*. 1998.

[32] A. Rahimi, B. Recht, et al. "Random Features for Large-Scale Kernel Machines." In: *NIPS*. Vol. 3. 4. Citeseer. 2007, p. 5.

[33] E. Rebrova et al. "A study of clustering techniques and hierarchical matrix formats for kernel ridge regression". In: *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2018, pp. 883–892.

[34] F.-H. Rouet et al. "A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization". In: *ACM Trans. Math. Software* 42.4 (2016), Art. 27, 35. ISSN: 0098-3500. DOI: 10.1145/2930660. URL: https://doi.org/10.1145/2930660.

[35] T. Sarlos. "Improved approximation algorithms for large matrices via random projections". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE. 2006, pp. 143–152.

[36] S. Si, C.-J. Hsieh, and I. S. Dhillon. "Memory efficient kernel approximation". In: *J. Mach. Learn. Res.* 18 (2017), Paper No. 20, 32. ISSN: 1532-4435.

[37] *STRUMPACK website*. http://portal.nersc.gov/project/sparse/strumpack/..

[38] R. Sun, Z.-Q. Luo, and Y. Ye. "On the efficiency of random permutation for ADMM and coordinate descent". In: *Math. Oper. Res.* 45.1 (2020), pp. 233–271. ISSN: 0364-765X. DOI: 10.1287/moor.2019.0990. URL: https://doi.org/10.1287/moor.2019.0990.

[39] E. E. Tyrtyshnikov. *A brief introduction to numerical analysis*. Birkhäuser Boston, Inc. Boston, MA, 1997, pp. xii+202. ISBN: 0-8176-3916-0. DOI: 10.1007/978-0-8176-8136-4. URL: https://doi.org/10.1007/978-0-8176-8136-4.

[40] R. Wang et al. "Block basis factorization for scalable kernel evaluation". In: *SIAM J. Matrix Anal. Appl.* 40.4 (2019), pp. 1497–1526. ISSN: 0895-4798. DOI: 10.1137/18M1212586. URL: https://doi.org/10.1137/18M1212586.

[41] C. Williams and M. Seeger. "Using the Nyström method to speed up kernel machines". In: *Proceedings of the 14th annual conference on neural information processing systems*. CONF. 2001, pp. 682–688.

[42] B. Xiao and G. Biros. "Parallel algorithms for nearest neighbor search problems in high dimensions". In: *SIAM J. Sci. Comput.* 38.5 (2016), S667–S699. ISSN: 1064-8275. DOI: 10.1137/15M1026377. URL: https://doi.org/10.1137/15M1026377.

[43] G.-B. Ye, Y. Chen, and X. Xie. "Efficient variable selection in support vector machines via the alternating direction method of multipliers". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 832–840.

[44] Y. You et al. "Accurate, fast and scalable kernel ridge regression on parallel and distributed systems". In: *Proceedings of the 2018 International Conference on Supercomputing*. 2018, pp. 307–317.

[45] K. Zhang and J. T. Kwok. "Clustered Nyström method for large scale manifold learning and dimension reduction". In: *IEEE Transactions on Neural Networks* 21.10 (2010), pp. 1576–1587.