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

Citation for published version:
Cipolla, S & Gondzio, J 2022, 'Training very large scale nonlinear SVMs using Alternating Direction Method of Multipliers coupled with the Hierarchically Semi-Separable kernel approximations', *EURO Journal on Computational Optimization*, vol. 10, 100046. https://doi.org/10.1016/j.ejco.2022.100046

Digital Object Identifier (DOI):
10.1016/j.ejco.2022.100046

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
EURO Journal on Computational Optimization

Contents lists available at ScienceDirect

EURO Journal on Computational Optimization

www.elsevier.com/locate/ejco

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

S. Cipolla*, J. Gondzio*

The University of Edinburgh, School of Mathematics, United Kingdom of Great Britain and Northern Ireland

ARTICLE INFO

Keywords:
Computational science
Support vector machines
Hierarchically Semi-Separable kernel approximations
Alternating Direction Method of Multipliers

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 an 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 at 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, in the case of Radial Basis Kernels, a significant speed-up when compared to the *state-*

* Corresponding authors.
E-mail addresses: scipolla@ed.ac.uk (S. Cipolla), j.gondzio@ed.ac.uk (J. Gondzio).

https://doi.org/10.1016/j.ejco.2022.100046

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 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) [6] coupled with *Hierarchically Semi-Separable* (HSS) [7] kernel approximations. Indeed, on the one hand, this framework allows to produce kernel approximations essentially in *a matrix-free* regime and with guaranteed accuracy [12], and, 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 and related works

Support vector machines (SVMs) [4,14] 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:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T Y K Y \mathbf{x} - \mathbf{e}^T \mathbf{x}$$

$$\begin{aligned} \text{s.t. } \mathbf{y}^T\mathbf{x} = 0, \\ x_i \in [0, C] \text{ for all } i = 1, \ldots, d, \end{aligned} \tag{1}$$

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 [24, 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 recognized 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 overparametrization and, in this case, SVMs may represent the *state of the art* for classification, see, e.g., [43,48]. 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 [18,26,35,36]: 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 [9] which delivers 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.

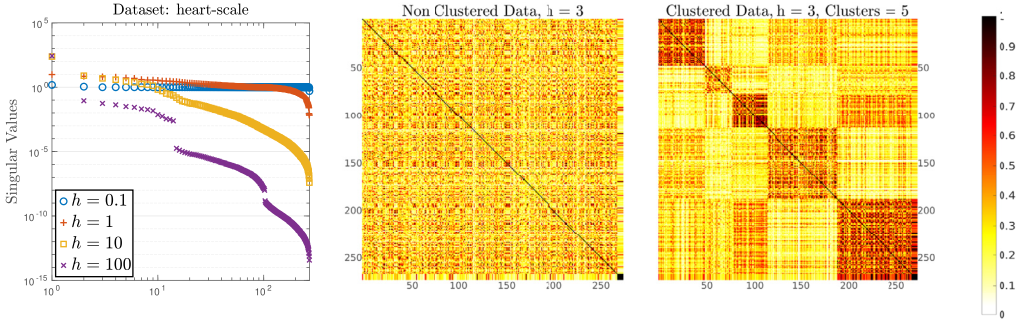


Fig. 1. Left Panel: decay of the singular values for Gaussian Kernel matrices. Right Panel: Gaussian Kernel matrices obtained with/without preliminary data clustering. Dataset: `heart_scale` [9].

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 [15,16,19,21–23,29,30,41,54]. We mention explicitly Nyström-type methods [21,28,49] and random feature maps to approximate the kernel function directly [37] 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 [46, 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 Fig. 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, so that between-classes interactions in the kernel matrix may be represented/well approximated by either sparse or low-rank matrices [42,47,53] (see right panel in Fig. 1 for a pictorial representation of this idea).

1.2. Motivations and contribution

This work represents a methodological contribution for the efficient solution of SVM problems. In particular, the aim of this work is to propose and analyze the use of the *Hierarchically Semi-Separable* (HSS) matrix representation [7] for the solution of large scale kernel SVMs. Indeed, the use of HSS approximations of kernel matrices has been already investigated in [12,38] for the solution of large scale Kernel Regression problems. The main reason for the choice of the HSS structure also in the SVM context can be summarized as follows:

1. using the STRUctured Matrix PACKage (STRUMPACK) [39] 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 [12]) essentially based on an adaptive randomized clustering and neighboring-based preprocessing of the data: in the preprocessing step employed by STRUMPACK, *approximate 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 geometry of the data to obtain valuable algebraic approximations of the kernel matrix. Indeed, the HSS structure does not require K to be low-rank, but only some off-diagonal parts to be rank-deficient, at least, after some suitable preprocessing. Broadly speaking, the preprocessing takes advantage of the fact that the interaction between two well separated clusters of data points can be approximated accurately when expressed in terms of the interaction between a smaller number of representative points from each cluster [38]. The implicit assumption made when using the HSS structure to approximate kernel matrices is that, after the preprocessing explained above and due to the exponential decay of many kernels, the resulting *small intra-group distances* can be approximated by low-rank matrices and hence the resulting kernel matrices can be *well approximated* by the HSS structure (see [38, Fig. 1a]);
2. the resulting approximations allow fast approximate kernel matrix computations with linear scalability for the computation of matrix-vector products and solution of linear systems, see [7,8,39].

In particular, we trace the main contribution of this work in unveiling a particularly efficient interaction between the HSS structure and ADMM [6] in the SVMs case, see Section 2. When problem (1) is suitably reformulated in a form exploitable by ADMM, the solution of just 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 very efficient optimization phase for a fixed value of C (see Section 3.3). Moreover, it is important to note 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 [25,52] analyze the use of ADMM for SVMs: in [52] ADMM has been used to solve linear SVMs with feature selection whereas in [25] 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. Nevertheless, as highlighted at the end of Section 1.1, the ability to approximate kernel matrices with low-rank ones depends on the chosen kernel parameters (see the left panel in Fig. 1). On the other hand, the optimal values of the kernel parameters are, in turn, data-dependent. Indeed, in general, when training a kernel based SVM, the kernel parameters for which the best performance is achieved in terms of classification accuracy, are not known before hand, and, to the best of our knowledge, the validity of *the small numerical rank assumption* is one of the main limitations for training Kernel SVM using kernel approximations. The efficient combination of ADMM with kernel approximations applicable in cases where the small numerical rank of the kernel matrix is not assumed, represents the key element of novelty of our approach when compared to the existing literature.

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 [6] is our choice of an (efficient) solution technique for problem (3). In Algorithm 1 we summarize its main steps:

Algorithm 1: ADMM.

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

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

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

Stating 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 [3, 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:

Algorithm 2: Closed form ADMM for problem (3).

```
for $k = 0, 1, \dots$ 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
```

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 [6].

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 employ here is the Hierarchically Semi-Separable (HSS) approximation of the kernel matrix in the form proposed in [12]. 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 [7]. The accurate description of the HSS compression technique in the case of kernel matrices is out of the scope of this work and, for this reason, we refer the reader to [12, Sec. II.B – II.C] for the full details. See also [32] and [39, Sec. 2.1] for more details on the HSS structure. The particular version of HSS we choose for our purposes is HSS-ANN (Hierarchically Semi-Separable - Approximate Nearest Neighbours), introduced in [12]. We mention explicitly the the features of HSS-ANN which have driven our choice:

- instead of using a randomized sampling (see [32]) to approximate column range of sub-matrices of K, this approach uses the kernel function to assess the similarity between data points and hence to identify the dominating entries of the kernel matrix. In particular, the columns corresponding to dominating Approximate Nearest Neighbours (ANN, see [31,51]) of the data points are selected to produce approximations of the column basis of particular sub-matrices of K, see [12, Sec. II.B]. As a result, the overall sampling strategy fully exploits the geometry of the underlying data-set and has a reduced cost when compared to the earlier HSS construction approaches, see [12, Sec. II-A] for a detailed comparison.
- the overall complexity of the HSS-ANN construction (excluding the preprocessing *clustering* 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 [12, Sec. II.C and Alg. 3]. The storage complexity of HSS-ANN is $O(dr)$;
- as a result of the previous two items, HSS-ANN exhibits better performance in terms of efficiency and approximation quality when compared with its predecessor and direct competitor, namely `ASKIT/INV-ASKIT` [10,11,31], which uses a block-diagonal-plus-low-rank hierarchical matrix format to construct an approximate representation, see [12, Sec. III];
- after the compression, the (shifted) HSS kernel matrix approximation can be factorized into a ULV-type form [39, Sec. 2.4], which exploits the special structure of the HSS generators coming from the Interpolative Decomposition, see [12, Alg. 3] and

references therein. This factorization, computed just once for fixed h in our approach, has a cost of $O(r^2 d)$ and can be used to solve linear systems involving the (shifted) kernel matrix in complexity $O(rd)$, see [39, Sec. 2.5].

3.2. Implementation details

In Algorithm 3 we summarize the pseudo-code of our implementation. It is based on `STRUMPACK` library (`Version 5.1.0`) [20,44] which provides efficient routines for the approximation $\tilde{K}$ of a kernel matrix K and efficient routines for the solution of the corresponding shifted linear systems. In particular, in Line 1 of Algorithm 3 such $\tilde{K}$ is obtained and, in Line 3, the ULV-type factorization of the matrix $\tilde{K}_\beta := \tilde{K} + \beta I$ is computed for the efficient solution of linear systems of the form $\tilde{K}_\beta \mathbf{x} = \mathbf{b}$. It is worth noting, at this stage, that for a fixed kernel value h the approximation $\tilde{K}$ and the ULV-type factorization of $\tilde{K}_\beta$ are computed just once and then *reused* for all the values C in the grid search (see Line 7 of Algorithm 3). Lines 9 - 14 of Algorithm 3 correspond to the ADMM optimization routine, see Algorithm 2 (resp. Algorithm 1). The "`x minimization`" step, which represents the dominant step in terms of computational cost, see Line 11, is performed resorting to the ULV-type factorization previously computed. In Lines 15 - 17 of Algorithm 3 the bias b is computed. It is important to note that, in practice, the bias is obtained averaging over all the support vectors that lie on the margin (see Line 17) 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, once more, Line 17 in Algorithm 3. Finally, in Lines 18 - 20 of Algorithm 3 we report the details for the "`Label Assignment`" of the testing instances.

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

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

Algorithm 3: SVM training/testing using Strumpack and ADMM.

```
Input: $K$ kernel function, $h$ kernel parameter, $\beta$ 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.
$\tilde{K}$ = HSScompression($K(F_{train}, F_{train}), h$) ;
$\tilde{K}_\beta = \tilde{K} + \beta I$;
$[U, L, V]$ = ULVfactorization($\tilde{K}_\beta$) ;
$\mathbf{w} = (ULV)^{-1}\mathbf{e}$;
$w_1 = \mathbf{e}^T\mathbf{w}$;
$\mathbf{w} = Y_{train}\mathbf{w}$ ;
for $C \in \{C_1, \dots, C_{max}\}$ do
    Initialize $\mathbf{x}_0, \mathbf{z}_0, \boldsymbol{\mu}_0$ ;
    for $k = 0, 1, \dots, MaxIt$ 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, see Algorithm 2 */
        $\mathbf{z}^{k+1} = \Pi_{[0,C]}(\mathbf{x}^{k+1} - \frac{1}{\beta}\boldsymbol{\mu}^k)$ ;        /* z minimization, see Algorithm 2 */
        $\boldsymbol{\mu}^{k+1} = \boldsymbol{\mu}^k - \beta(\mathbf{x}^{k+1} - \mathbf{z}^{k+1})$ ;        /* Multiplier Update, see Algorithm 2 */
    end
    Define $\mathbf{z_y} = Y_{train}\mathbf{z}^{MaxIt}$ ;        /* Computing Bias */
    Define $\bar{e}_j = 1$ if $0 < (\mathbf{z}_{MaxIt})_j < C$ or $\bar{e}_j = 0$ otherwise ;
    $b = \frac{1}{\|\bar{\mathbf{e}}\|_1}(\mathbf{z_y}^T\tilde{K}\bar{\mathbf{e}} - \sum_{j\,:\bar{e}_j\neq 0}(\mathbf{y}_{train})_j)$ ;
    for $j = 1, \dots, m$ do
        $(\tilde{\mathbf{y}}_{test})_j = sign(\sum_{i=1}^d(\mathbf{z_y})_iK((\mathbf{f}_{train})_i, (\mathbf{f}_{test})_j) + b)$ ;        /* Label Assignement */
    end
end
```

to the solution $\bar{\mathbf{x}}$ of the original problem (1). Indeed, using a similar technique to the one presented in [19, Sec. 4.1.], for any unitary invariant form we obtain

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

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. Indeed, surprisingly enough, it has been observed multiple times that for kernel methods even poor approximations of the kernel can suffice to achieve near-optimal performance [2,40]. On the other hand, it is also important to note that if the matrix K has the *HSS property* (see [32, Sec. 3]), and assuming that $\tilde{K}$ is computed by truncating every HSS block with a truncation tolerance $O(\varepsilon)$, then also the global error $\|K - \tilde{K}\|$ stays of order $O(\varepsilon)$: specific results are available for the Frobenius and spectral norms, see [50, Corollary 4.3] and [27, Theorem 4.7]. In particular, supposing that every HSS block of K can be approximated with an error $O(\varepsilon)$ by a matrix of rank r, if a randomized sampling procedure is used to produce the low-rank approximations with oversampling parameter p,

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

Dataset	Features	Training Set Dim.	$\lvert Train_+ \rvert$	Test Set Dim.	$\lvert Test_+ \rvert$
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

then $\tilde{K}$ is a global $O(\varepsilon)$ approximation with probability at least $1 - 6p^{-p}$ (see [32, Sec. 2.3] and discussion in [33, Sec. 3.2]).

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 2.2 GHz, 20M Cache, 7.2 GT/s QPI, Turbo (4x10Cores) 256 GB RAM. The code is publicly available at the address https://github.com/StefanoCipolla/Strumpack_ADMM. In the following, we report on its performance in the case of the Gaussian Kernel, but similar computational results are expected to hold for Laplacian and ANOVA kernels since the efficiency of HSS-ANN has been demonstrated also in these cases, see [12, Fig. 5].

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 the proposed approach. It is worth mentioning that computational experience confirms that a different choice of this parameter may lead to a better classification performance for particular test examples. 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 $d \geq 10^6$.

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

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 3
RACQP. †† = 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	††	

In Table 2 we report the results obtained using `LIBSVM Version 3.25` [9], which implements specialized algorithms for the SVM problem (LIBSVM uses a Sequential Minimal Optimization type decomposition method [5,17,36]). In Table 3 we report the results obtained using RACQP [34] (where a multi-block generalization of ADMM is employed, see also [13,45] for related theoretical analysis).

In particular, the kernel parameter h and the penalization term C were estimated by running a grid-check when instances were solved using our proposal (the HSS-ANN accuracy parameters used were those specified in Table 5 since our proposal achieved (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 was 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

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

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

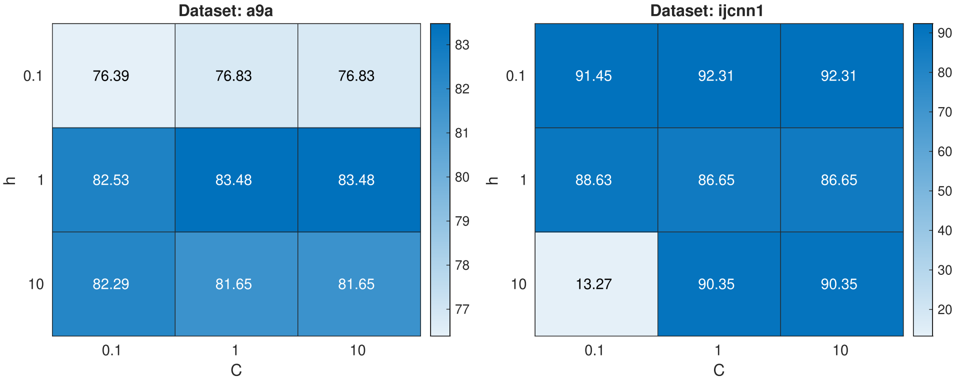
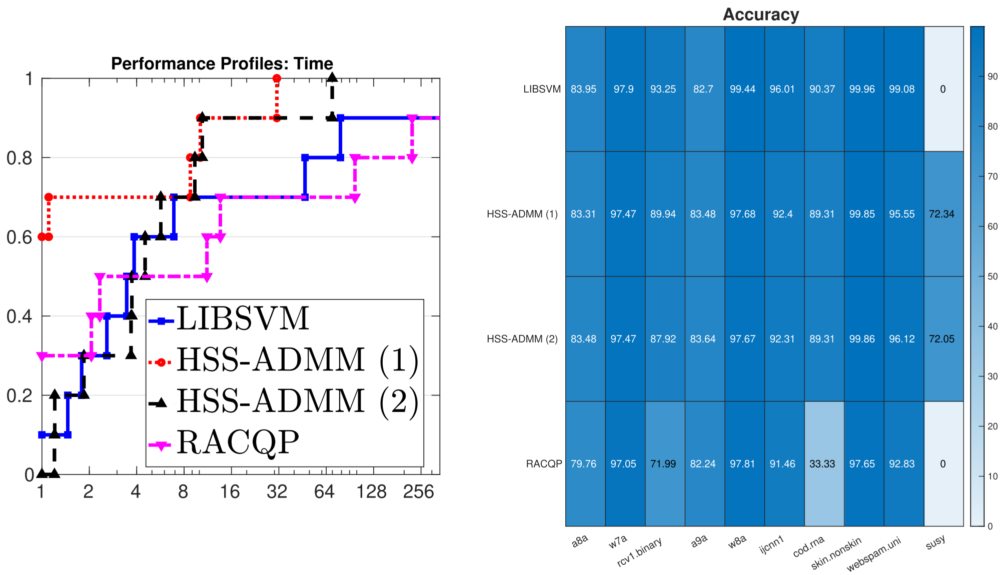


Fig. 2. Heatmap of the classification accuracy for the datasets `a9a` and `ijcnn1`.

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 10 s 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 Fig. 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). All the previous observations are further highlighted by Fig. 3, where the results presented in Table 2 (`LIBSVM`), Table 3 ((`RACQP`), Table 4 (`HSS-ADMM (1)`) and Table 5 (`HSS-ADMM (2)`)) are pictorially summarized.

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



	a8a	w7a	rcv1.binary	a9a	w8a	ijcnn1	cod.rna	skin.nonskin	webspam.uni	susy
LIBSVM	83.95	97.9	93.25	82.7	99.44	96.01	90.37	99.96	99.08	0
HSS-ADMM (1)	83.31	97.47	89.94	83.48	97.68	92.4	89.31	99.85	95.55	72.34
HSS-ADMM (2)	83.48	97.47	87.92	83.64	97.67	92.31	89.31	99.86	96.12	72.05
RACQP	79.76	97.05	71.99	82.24	97.81	91.46	33.33	97.65	92.83	0

Fig. 3. Summary of the results contained in Table 2 (`LIBSVM`), Table 3 (`RACQP`), Table 4 (`HSS-ADMM (1)`) and Table 5 (`HSS-ADMM (2)`).

4. Conclusions and future work

In this work we proposed an ADMM-based scheme (see Algorithm 3) which employs HSS-ANN approximations (see [12] and Section 3) to train SVMs. Numerical experiments obtained using `STRUMPACK` [44] in a sequential architecture, show that our proposal compares favorably with `LIBSVM` [9] and `RACQP` [34] 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. Finally, as subject of future work, the authors will consider performing a thorough comparison of the efficiency of the proposed framework against other possible couplings of the type *ADMM+Kernel Approximation*.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by Oracle Labs through the project “Randomly Sampled Cyclic Alternating Direction Method of Multipliers”.

References

[1] H. Avron, K.L. Clarkson, D.P. Woodruff, Faster kernel ridge regression using sketching and preconditioning, SIAM J. Matrix Anal. Appl. 38 (4) (2017) 1116–1138.

[2] F. Bach, Sharp analysis of low-rank kernel matrix approximations, in: Conference on Learning Theory, in: PMLR, 2013, pp. 185–209.

[3] D.P. Bertsekas, Nonlinear Programming, second ed., Athena Scientific Optimization and Computation Series, Athena Scientific, Belmont, MA, 1999, xiv+777.

[4] B.E. Boser, I.M. Guyon, 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.

[5] L. Bottou, C.-J. Lin, Support vector machine solvers, in: Large Scale Kernel Machines, vol. 3.1, 2007, pp. 301–320.

[6] S. Boyd, N. Parikh, E. Chu, Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers, Now Publishers Inc., 2011.

[7] S. Chandrasekaran, M. Gu, W. Lyons, A fast adaptive solver for hierarchically semiseparable representations, Calcolo 42 (3–4) (2005) 171–185.

[8] S. Chandrasekaran, M. Gu, T. Pals, A fast ULV decomposition solver for hierarchically semiseparable representations, SIAM J. Matrix Anal. Appl. 28 (3) (2006) 603–622.

[9] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, in: ACM Transactions on Intelligent Systems and Technology, vol. 2, 3.2011, 27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[10] D.Y. Chenhan, W.B. March, G. Biros, An $n \log n$ parallel fast direct solver for kernel matrices, in: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2017, pp. 886–896.

[11] D.Y. Chenhan, et al., INV-ASKIT: a parallel fast direct solver for kernel matrices, in: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2016, pp. 161–171.

[12] 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.

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

[14] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.

[15] P. Drineas, M.W. Mahoney, On the Nyström method for approximating a Gram matrix for improved kernel-based learning, J. Mach. Learn. Res. 6 (2005) 2153–2175.

[16] P. Drineas, et al., Fast approximation of matrix coherence and statistical leverage, J. Mach. Learn. Res. 13 (2012) 3475–3506.

[17] R.-E. Fan, P.-H. Chen, C.-J. Lin, Working set selection using second order information for training support vector machines, J. Mach. Learn. Res. 6 (2005) 1889–1918.

[18] R.-E. Fan, et al., Working set selection using second order information for training support vector machines, J. Mach. Learn. Res. 6 (12) (2005).

[19] S. Fine, K. Scheinberg, Efficient SVM training using low-rank kernel representations, J. Mach. Learn. Res. 2 (Dec. 2001) 243–264.

[20] 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.

[21] A. Gittens, M.W. Mahoney, Revisiting the Nyström method for improved large-scale machine learning, J. Mach. Learn. Res. 17 (2016) 117.

[22] G.H. Golub, C.F. Van Loan, Matrix Computations, vol. 3, JHU Press, 2012.

[23] N. Halko, P.G. Martinsson, J.A. Tropp, Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288.

[24] T. Hofmann, B. Schölkopf, A.J. Smola, Kernel methods in machine learning, Ann. Stat. 36 (3) (2008) 1171–1220.

[25] S.-A. Huang, Y.-Y. Hsieh, C.-H. Yang, Design optimization for ADMM-based SVM training processor for edge computing, in: 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), IEEE, 2021, pp. 1–5.

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

[27] D. Kressner, S. Massei, L. Robol, Low-rank updates and a divide-and-conquer method for linear matrix equations, SIAM J. Sci. Comput. 41 (2) (2019) A848–A876.

[28] S. Kumar, M. Mohri, A. Talwalkar, Sampling methods for the Nyström method, J. Mach. Learn. Res. 13 (2012) 981–1006.

[29] E. Liberty, et al., Randomized algorithms for the low-rank approximation of matrices, Proc. Natl. Acad. Sci. USA 104 (51) (2007) 20167–20172.

[30] M.W. Mahoney, Randomized algorithms for matrices and data, Found. Trends Mach. Learn. 123–224 (3) (2011).

[31] W.B. March, B. Xiao, G. Biros, ASKIT: approximate skeletonization kernel-independent treecode in high dimensions, SIAM J. Sci. Comput. 37 (2) (2015) A1089–A1110.

[32] P.G. Martinsson, A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix, SIAM J. Matrix Anal. Appl. 32 (4) (2011) 1251–1274.

[33] S. Massei, L. Robol, D. Kressner, hm-toolbox: MATLAB software for HODLR and HSS matrices, SIAM J. Sci. Comput. 42 (2) (2020) C43–C68.

[34] K. Mihic, M. Zhu, Y. Ye, Managing randomization in the multi-block alternating direction method of multipliers for quadratic optimization, Math. Program. Comput. (2020).

[35] E. Osuna, R. Freund, 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.

[36] 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.

[37] A. Rahimi, B. Recht, et al., Random features for large-scale kernel machines, in: NIPS, vols. 3, 4, 2007, p. 5.

[38] 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.

[39] F.-H. Rouet, et al., A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization, ACM Trans. Math. Softw. 42 (4) (2016) 27.

[40] A. Rudi, R. Camoriano, L. Rosasco, Less is more: Nyström computational regularization, in: NIPS, 2015, pp. 1657–1665.

[41] 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.

[42] S. Si, C.-J. Hsieh, I.S. Dhillon, Memory efficient kernel approximation, J. Mach. Learn. Res. 18 (2017) 20.

[43] S. Singh, et al., SVM based system for classification of microcalcifications in digital mammograms, in: 2006 International Conference of the Ieee Engineering in Medicine and Biology Society, IEEE, 2006, pp. 4747–4750.

[44] STRUMPACK website, http://portal.nersc.gov/project/sparse/strumpack/.

[45] R. Sun, Z.-Q. Luo, Y. Ye, On the efficiency of random permutation for ADMM and coordinate descent, Math. Oper. Res. 45 (1) (2020) 233–271.

[46] E.E. Tyrtyshnikov, A Brief Introduction to Numerical Analysis, Birkhäuser Boston, Inc., Boston, MA, 1997, xii+202.

[47] R. Wang, et al., Block basis factorization for scalable kernel evaluation, SIAM J. Matrix Anal. Appl. 40 (4) (2019) 1497–1526.

[48] L. Wei, et al., A study on several machine-learning methods for classification of malignant and benign clustered microcalcifications, IEEE Trans. Med. Imaging 24 (3) (2005) 371–380.

[49] C. Williams, M. Seeger, Using the Nyström method to speed up kernel machines, in: Proceedings of the 14th Annual Conference on Neural Information Processing Systems, 2001, pp. 682–688.

[50] Y. Xi, et al., Superfast and stable structured solvers for Toeplitz least squares via ran- domized sampling, SIAM J. Matrix Anal. Appl. 35 (1) (2014) 44–72.

[51] B. Xiao, G. Biros, Parallel algorithms for nearest neighbor search problems in high dimensions, SIAM J. Sci. Comput. 38 (5) (2016) S667–S699.

[52] G.-B. Ye, Y. Chen, 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.

[53] 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.

[54] K. Zhang, J.T. Kwok, Clustered Nyström method for large scale manifold learning and dimension reduction, IEEE Trans. Neural Netw. 21 (10) (2010) 1576–1587.