High Level High Performance Computing for Multitask Learning of Time-varying Models

Marco Signoretto, Emanuele Frandi, Zahra Karevan and Johan A. K. Suykens

ESAT-STADIUS, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, B-3001 Leuven (BELGIUM)
Email: {marco.signoretto, emanuele.frandi, zahra.karevan, johan.suykens}@esat.kuleuven.be

***Abstract*—We propose an approach suitable to learn multiple time-varying models jointly and discuss an application in data-driven weather forecasting. The methodology relies on spectral regularization and encodes the typical multi-task learning assumption that models lie near a common low dimensional subspace. The arising optimization problem amounts to estimating a matrix from noisy linear measurements within a trace norm ball. Depending on the problem, the matrix dimensions as well as the number of measurements can be large. We discuss an algorithm that can handle large-scale problems and is amenable to parallelization. We then compare high level high performance implementation strategies that rely on JIT decorators. The approach enables, in particular, to offload computations to a GPU without hard-coding computationally intensive operations via a low-level language. As such, it allows for fast prototyping and therefore it is of general interest for developing and testing novel computational models.**

I. Introduction

Time series analysis usually relies on the assumption of time-invariance. In practice in many cases this assumption is violated, which might result in suboptimal models and unreliable predictions. Our interest in the subject originates, in particular, from data-driven weather forecasting. In this setting the variation over time might arise naturally in correspondence of changing weather patterns during the solar year.

In this paper we are concerned with the estimation of a class of models within the general time-varying framework:

$$y_t = f(x_t, t) + \epsilon_t \quad \text{for} \quad t = 1, \ldots, T\ . \tag{1}$$

In the latter y_t and x_t are a scalar and a vector of dimension D, respectively, and we have let f depend explicitly on the time point t. Special cases arise from (1) by specifying the nature of f and that of x_t. In general x_t, termed *regression vector*, subsumes the information available at t; an important special case, in particular, arises from

$$x_t = [y_{t-1},\ y_{t-2},\ \ldots,\ y_{t-p}]^\top \tag{2}$$

which leads to time-varying autoregressive models of order p.

A. Estimating Time-varying Models is Data Consuming

The estimation of time varying models from data is usually more difficult than the estimation of their time-invariant counterparts. In fact, (1) requires to estimate a function $f(\cdot, t)$ locally around each time point t. This makes the estimation procedure even more data consuming than in the time-invariant setting. Among others, [15] has recently studied asymptotic properties of Nadaraya-Watson kernel-based estimators for a class of models that fits in the general framework of (1). It was shown that a way to circumvent the curse of dimensionality, and therefore decrease the amount of data needed for the estimation, consists of imposing structural constraints on f.

B. Multitask Learning Approach

Rather that imposing a specific structure over a single forecasting model, in this work we circumvent the curse of dimensionality by leveraging the relationship between the regression functions $f^{(1)}, f^{(2)}, \ldots, f^{(I)}$ of multiple related models. This might correspond, for instance, to find temperature forecasting models for different locations jointly, which we consider in our experiments. This approach fits in the general framework of Multi-Task Learning (MTL) [7], [4], which in the last decade has emerged as an important avenue of machine learning research. MTL improves the generalization error by pooling information across related learning tasks. This is particularly effective when there are many tasks but only few data per task, as found in practical studies as well as in the theoretical analysis, see e.g. [13] and references therein.

We focus on an approach based on spectral regularization which encodes the typical MTL assumption that models lie near a common low dimensional subspace [13]. The methodology ultimately leads to a convex optimization problem aimed at estimating a matrix under linear measurements. As in practice the matrix can be large, it is essential to rely on scalable algorithms. We discuss a generalized Frank-Wolfe algorithm [11] that is both scalable and amenable to parallel computations. We then elaborate on high level high performance implementations of the aforementioned algorithm; specifically we compare an high level Python implementation with just-in-time (JIT) compiled versions for both CPUs and GPUs. The strategy is generally suitable for fast prototyping in the context of scientific computing. Finally, we present experiments on temperature forecasting across hundreds of weather stations simultaneously.

The remainder of the paper is organized as follows. In the next Section we discuss our approach to learn multiple time-varying models jointly. Section III presents and compares different implementation strategies for the proposed algorithm. Section IV deals with an application in data-driven weather forecasting. Finally, we draw our concluding remarks in Section V.

II. Learning Related Time Series Models via Spectral Regularization

A. Model Representation

For a positive integer I, we denote by $[I]$ the set $\{1, 2, \ldots, I\}$. In this work we are concerned with finding regression functions:

$$f^{(i)}: \begin{array}{ccc} \mathcal{X} \times [T] & \rightarrow & \mathbb{R} \\ (x,t) & \mapsto & y \end{array} \tag{3}$$

where $i \in [I]$ denotes the *task index*, t is a time point within the discrete set $[T]$ and $x \in \mathcal{X} \subset \mathbb{R}^D$ will subsume the information available at t, common to all the tasks. Each regression function will serve as a time-varying model (1) and will be taken from the same *Reproducing Kernel Hilbert Space* (RKHS) [3], [6], denoted by $\mathcal{H}_k$. Specifically we consider the RKHS spanned by the separable kernel:

$$k\left((x,t),(x',t')\right) = x^\top x' g\left(t,t'\right) \tag{4}$$

where $g(t,t')$ is a positive definite kernel associated to a RKHS of functions over $[T]$. A simple case arises from the Gaussian RBF kernel:

$$g\left(t,t'\right) = \exp\left(-\left(t-t'\right)^2/\sigma^2\right) . \tag{5}$$

Alternatively, it might be desirable to encode specific invariances over time. In particular, one might want to rely on a periodic kernel such as:

$$g\left(t,t'\right) = \exp\left(-\sin\left(\pi|t-t'|/T\right)/\sigma^2\right) . \tag{6}$$

This is desirable for the temperature forecasting purposes mentioned in the Introduction. Note that for $T = 365$, in fact, (6) implies that $f^{(i)}(x,1) \approx f^{(i)}(x,365)$, in line with the period of revolution around the sun. Finally, note that for both (5) and (6), σ is a user-defined parameter; importantly, taking $\sigma \rightarrow \infty$ leads to time-invariant models, which therefore arise as a special case of our framework.

B. Operator Learning

Let $e^{(i)} \in \mathbb{R}^I$ be the canonical basis vector defined by

$$e^{(i)}_j = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise} . \end{cases}$$

In order to estimate all the functions jointly we introduce the auxiliary operator $F : \mathbb{R}^I \rightarrow \mathcal{H}_k$ defined via the linear functional relationship:

$$\left\langle k_{(x,t)}, Fe^{(i)} \right\rangle = f^{(i)}(x,t) \tag{7}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in $\mathcal{H}_k$ and we have written $k_{(x,t)}$ to mean the function

$$k_{(x,t)} : (x',t') \mapsto k\left((x,t),(x',t')\right) .$$

By (7), the problem of learning the set of functions $\left(f^{(i)}\right)_{i\in[I]}$ is turned into the problem of learning F. This represents an instance of the general problem of learning *compact operators*[1] between RKHSs, which was addressed in [1] within the context of collaborative filtering. Denote by

$$z = (x,t) \in \mathcal{Z} = \mathcal{X} \times [T]$$

a generic point given by a regression vector and the corresponding time index. In the following we write $\mathcal{D}^{(i)}$ to mean the task-specific dataset consisting of $N_i > 0$ observed input-output pairs associated to $f^{(i)}$:

$$\mathcal{D}^{(i)} = \{(z_n, y_n) \ : \ n \in [N_i]\} \subset \mathcal{Z} \times \mathbb{R} . \tag{8}$$

A learning problem suitable for the present setting is then:

$$\min \left\{ \sum_{i\in[I]} \sum_{(z,y)\in\mathcal{D}^{(i)}} \left(y - \left\langle k_z, Fe^{(i)} \right\rangle\right)^2 : \|F\|_1 \leq \tau, F \in \mathcal{F} \right\} \tag{9}$$

where $\tau > 0$ is a user-defined parameter. We denoted by $\mathcal{F}$ the set of finite-rank operators from $\mathbb{R}^I$ to $\mathcal{H}_k$ and by $\|F\|_1 = \sum_{i\in[I]} \sigma_i(F)$ the *nuclear* (also known as *trace*) norm defined upon the singular values $\sigma_1(F) \geq \sigma_2(F) \geq \cdots \geq \sigma_I(F) \geq 0$.

By solving (9) we aim to find a low-complexity operator that is able to explain the data. The trace norm is the convex envelope of the rank function within the spectral norm unit ball [9]. This motivates its use under the typical MTL assumption that parameter vectors lie near a common low dimensional subspace, see [13] and references therein.

C. Optimization Problem and Model Evaluation

Let L denote the dimension of the subspace spanned by the kernel functions (4) centered at the input data points within the entire pool of tasks. It follows from the *representer theorem* given in [1] that any solution $\hat{F}$ of (9) lies in the span of LI rank-1 operators that are constructed upon the input patterns and the task indicators:

$$\hat{F} = \sum_{l\in[L]} \sum_{i\in[I]} \hat{A}_{li} F^{(l,i)} . \tag{10}$$

Equation (10) implies that finding $\hat{F}$, and therefore a set of optimal regression functions, can be turned into the problem of finding an optimal weight matrix $\hat{A} \in \mathbb{R}^{L\times I}$. In the present setting, the problem formulation to find $\hat{A}$ assumes specific traits, which are not proper of the general case discussed in [1]. What makes our setting specific are the following facts: *1*) the kernel function (4) is separable and depends linearly on each regression vector, and *2*) the problem involves finitely many time points. In order to further elaborate on this, consider the set $\mathcal{T}$ consisting of the input-task-output triple obtained from pooling the task-specific datasets:

$$\mathcal{T} = \left\{(z,i,y) \ : \ (z,y) \in \mathcal{D}^{(i)}, \ i \in [I]\right\} . \tag{11}$$

We denote by $N = \sum_{i\in[I]} N_i$ the cardinality of $\mathcal{T}$. In general, finding $\hat{A}$ within the framework of [1] would require, as a preprocessing step, the factorization of a $N \times N$ kernel matrix. Computing such a factorization, however, is generally unfeasible even for moderately large N. In the present context, on the other hand, the aforementioned points *1* and *2* make it sufficient to factorize only the kernel matrix $G \in \mathbb{R}^{T\times T}$ constructed upon the set of time points:

$$G_{tt'} = g(t,t') \text{ for } t,t' \in [T] . \tag{12}$$

This is important since we are interested in the setting where $N \gg T$. In the following we let

$$H = [h_1, h_2, \cdots, h_T] \in \mathbb{R}^{P\times T}$$

[1]Note that in the present context the operator F is *finite-rank* and hence compact.

be any factor matrix satisfying $G = H^\top H$; furthermore, we let $M(a_i)$ be a $D \times P$ matrix obtained by reshaping the i-th column vector of $A = [a_1, a_2, \cdots, a_I] \in \mathbb{R}^{L \times I}$, where we have $L = PD$ in the present context. Consider now the function

$$E(A) = \sum_{((x,t),i,y)\in\mathcal{T}} \left(y - x^\top M(a_i) h_t\right)^2 . \quad (13)$$

The following proposition follows from the general results found in [1], which are specialized to the structure of the present problem. It summarizes the optimization problem that needs to be solved in the present setting, as well as the formula required to evaluate the corresponding time-series models.

Proposition 1. *If $\hat{A} = [\hat{a}_1, \hat{a}_2, \cdots, \hat{a}_I]$ is a solution to*

$$\min \left\{ E(A) \ : \ \|A\|_1 \leq \tau, \ A \in \mathbb{R}^{PD \times I} \right\}, \quad (14)$$

then for any $i \in [I]$ we have

$$\hat{f}^{(i)} : (x,t) \mapsto x^\top M(\hat{a}_i) h_t, \quad (15)$$

where $\hat{f}^{(i)} \in \mathcal{H}_k$ is an optimal regression function associated via (7) *to a solution of* (9)*.*

D. Generalized Frank-Wolfe Algorithm

Problem (14) represents a special instance of the general task of recovering a matrix within a trace norm ball, based upon a collection of noisy linear measurements. For such problems the high dimensionality of the matrix unknown A usually prevents the use of second order information in the solution strategy. One then typically resorts to first order methods. An approach that has attracted a lot of attention consists in specializing the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) given in [5]. This requires to compute at each iteration the full Singular Value Decomposition (SVD) of A and therefore the approach is still not suitable to the case where the dimensions of A are very large. Instead, here we focus on an instance of a generalized Frank-Wolfe (FW) method, which recently attracted renewed attention for large-scale problems [11]. The main iteration of the algorithm is given below, see also [2] for details and references.

Algorithm 1 Generalized Frank-Wolfe Algorithm

Input: user-defined hyper-parameter $\tau > 0$
$A_0 = 0$
for $i = 1, 2, \ldots$ **do**
(I) $X_i \leftarrow \nabla E(A_{i-1})$
(II) $Q_i \leftarrow \tau\, uv^\top \quad : \quad u^\top X_i v = \sigma_1(X_i)$
(III) $A_i \leftarrow (1-\alpha_i) A_{i-1} + \alpha_i Q_i, \quad \alpha_i \in [0,1]$
end for
$\hat{A} \leftarrow A_i$
Output: A solution $\hat{A}$ to problem (14)

Each iteration of the FW algorithm gives a feasible estimate, i.e. a matrix that lies within the nuclear norm ball in (14). Rather that the full SVD of A, the FW algorithm only requires to compute the left and right leading singular vectors in step II, which we approached in practice via the power method [10]. The lower iteration complexity is traded by a worse convergence rate with respect to FISTA. In practice, however, the lower cost per iteration makes the FW algorithm a much better choice for large scale problems, see [2] for details.

Before continuing, we address the choice of the step size α_i in step III of Algorithm 1. Letting $\alpha_i = 2/(i+1)$ provably leads to convergence to a global solution. This holds true with respect to the very general class of problems to which the FW algorithm applies, see [2]. A drawback is that this does not ensure a decrease in the objective function at each iteration. In the present setting, however, the quadratic nature of (13) makes it possible to compute the step size ensuring the largest decrease in the objective. Specifically, first order optimality conditions yield:

$$\alpha_i = \Pi_{[0,1]} \left(\frac{\sum_{((x,t),i,y)\in\mathcal{T}} n(A_{i-1}, Q_i; x, t, i, y)}{\sum_{((x,t),i,y)\in\mathcal{T}} d(A_{i-1}, Q_i; x, t, i, y)} \right), \quad (16)$$

where we denoted by $\Pi_{[0,1]}$ the projection on the unit interval and we have:

$$n(A, Q; x, t, i, y) = -\left(x^\top M(a_i) h_t\right)\left(x^\top M(q_i) h_t\right) + \left(x^\top M(a_i) h_t\right)^2 + y\left(x^\top M(q_i) h_t\right) - y\left(x^\top M(a_i) h_t\right) \quad (17a)$$

$$d(A, Q; x, t, i, y) = \left(x^\top M(a_i) h_t - x^\top M(q_i) h_t\right)^2 . \quad (17b)$$

Our next goal is to discuss high-level software implementations of the FW algorithm and compare their performance; we focus on an approach that is generally suitable for fast prototyping in the context of scientific computing.

III. High-level, High-performance Implementations

Fast prototyping is a desirable — if not essential — feature to facilitate the workflow of developing computational models. This has led to the wide-spreading of commercial frameworks for technical computing, such as MATLAB, as well as high-level programming languages such as R and Python. These interpreted languages provide desirable high-level abstractions from computers' instruction set architecture. This, however, comes at the price of having significantly lower performance in comparison to the compiled alternatives, such as C or Fortran. The situation has been improved by the development of Just-In-Time (JIT) compilers [12], which are now allowing for high-level, high-performance implementations of algorithms.

A. Python and JIT Compilation via Decorators

We implemented each of the key steps of Algorithm 1 via one or more Python functions[2]. As a first case, we relied on the high level mathematical abstraction provided by the Numpy and Scipy packages, which leads to readable and compact code. In the following we refer to this approach simply as *Python*. Alternatively, the computationally intensive steps where coded in Python without relying on high level mathematical abstractions; JIT *decorators* where then used to annotate functions for deferred compilation at callsite. Specifically we used the Numba[3] `jit` module and the Numbapro[4]

[2]We plan to release our software and the related documentation after the conference.

[3]http://numba.pydata.org/

[4]http://docs.continuum.io/numbapro/. See also http://goo.gl/Q4a5Dt

`cuda.jit` module, referred to in the following as *JIT* and *CUDA-JIT*, respectively. Whereas the former uses the CPU, the latter offloads the computation to a Nvidia CUDA GPU. This specific choices were made because the use of decorators allows to improve performance at a very reasonable cost. In particular, it avoids the hassle of hard-coding computationally intensive operations via a low-level compiled language.

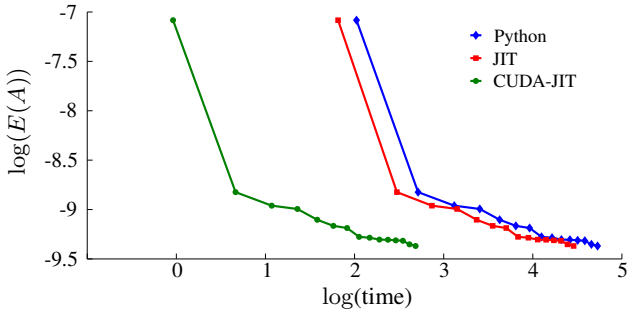

Fig. 1: Comparison of objective values as a function of time (seconds) for the different implementations of Algorithm 1.

B. Implementation of the FW algorithm

Note that all the key steps in the FW algorithm are amenable to parallelization. The computation of ∇E in step I can be distributed across CUDA cores simply considering each element in $\mathcal{T}$ separately. The same comment applies to the steps required in the computation of the optimal α_i in III, which is performed based on (16). Finally, to compute the leading singular vectors in step II we relied on the power method. This, in turn, simply amounts to computing a single matrix-matrix multiplication followed by a matrix-vector multiplication per iteration of the power method. Each of these tasks is trivially parallelized, and can therefore profit from offloading computation to a GPU.

TABLE I: Average time per iteration (in seconds) of the different implementations of Algorithm 1.

I		Python	JIT	CUDA-JIT
20	Mean	$9.82e-01$	$7.45e-01$	$1.12e-01$
	std dev	$1.01e-02$	$9.23e-03$	$2.44e-03$
40	Mean	$1.94e+00$	$1.47e+00$	$2.16e-01$
	std dev	$2.05e-02$	$1.83e-02$	$1.57e-03$
80	Mean	$3.87e+00$	$2.92e+00$	$4.43e-01$
	std dev	$2.28e-02$	$1.50e-02$	$3.65e-03$
160	Mean	$7.83e+00$	$5.92e+00$	$9.90e-01$
	std dev	$5.39e-02$	$3.76e-02$	$6.16e-03$
320	Mean	$1.62e+01$	$1.23e+01$	$2.09e+00$
	std dev	$3.69e-01$	$3.16e-01$	$9.64e-03$
640	Mean	$3.44e+01$	$2.66e+01$	$4.49e+00$
	std dev	$1.02e+00$	$7.75e-01$	$1.55e-02$

C. Performance Comparison

We compared the different implementation strategies on a synthetic test case that consists of learning a varying number of regression functions. In the context of MTL each of these functions corresponds to a task, as detailed in Section II-A. For each task 500 input-output observations were considered. Correspondingly, the cardinality of $\mathcal{T}$ in this case grows linearly with the number of tasks I, which also corresponds to the second dimension of the matrix A that is computed by the FW algorithm. All the experiments in this paper were carried on `g2.2xlarge` AMAZON EC2 instances[5]. Figure 1 represents the time evolution of the objective function of problem (14) along the successive iterations of the FW algorithm for a representative case with $I = 160$. Table I reports the average time per iteration of the native Python implementation and the JIT compiled versions.

In light of the speed-up that can be achieved, in the following we rely on the CUDA-JIT implementation. Our next goal is to test the proposed approach to learn time varying models jointly on a data-driven weather forecasting application.

IV. Data-driven Weather Forecasting

Numerical weather forecasting dates back to the beginning of the last century. However it became feasible only much later thanks to the development of computers. Standard techniques require the simulation of huge fluid dynamics and thermodynamics PDEs. This is both computationally very intensive and prone to approximation errors that occur in the modeling phase and in the successive discretization step. It is therefore of interest to explore data-driven alternatives, which can profit from the growing collection of meteorological data that are available nowadays.

A. Forecasting Temperatures by Time-varying Models

Here we focus on the task of forecasting 1-day ahead the temperatures at multiple weather stations simultaneously. The approach ultimately amounts to solving multiple instances of the optimization problem (14). Each station is associated to a time-varying forecasting model specified as in Section II-A. The periodic kernel (6) was considered to model the dependency over time. The kernel parameter σ and the nuclear norm parameter τ were tuned to ensure the best performance.

B. Data Gathering and Processing

Weather data were collected from the National Climatic Data Center[6] of the National Oceanic and Atmospheric Administration (NOAA) which gathers measurements from several thousands land-based stations. In this paper we considered only those stations for which a minimal number of observations in the last two decades were present. This led to the selection of 350 stations located in the US with observations ranging from mid 1998 until late 2010. The distribution of the stations is shown in Figure 2. For each day, feature vectors from each station were concatenated. Each feature vector consisted of two core features containing minimum and maximum temperature; some stations further included measurements of precipitation, snow depth, average daily wind speed, temperature at time of observation, fastest 2-minute and 5-second wind speeds and their directions. As a preprocessing step, features were normalized to have zero mean and unit variance. Subsequently, Kernel Principal Component Analysis was applied to extract a subset of 100 dominant nonlinear features. Finally, a feature identically equal to 1 was added to the resulting regression vector to account for a bias term.

[5]Details on the hardware can be found at http://goo.gl/rwZoKz
[6]http://goo.gl/5VyRYY



Fig. 2: Distribution of weather stations.

C. Experimental Results

The aforementioned regression vector subsumes the global information available at a given observed day. Our goal is to predict the maximal temperature for the next day at each station. We considered the MTL-based approach discussed in Section II. To this end, we used the data up to the end of 2002 for training and validating models, and the rest for testing. Results were compared to a baseline obtained from Least-Squares Support Vector Machine (LS-SVM) models [14] computed via the LS-SVMlab toolbox [8] with linear kernel[7]. In this case, a model per station is obtained independently from the others. Figure shows 1-step ahead predictions versus realized maximal temperatures for a representative station. The corresponding mean squared errors were 61 for LS-SVM and 28 for MTL.

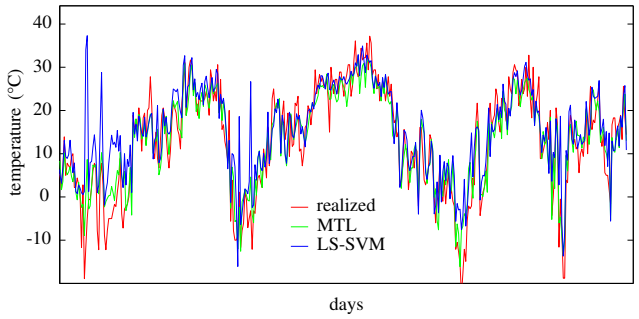

Fig. 3: Forecasted and realized temperatures for a single station.

V. Conclusion

In this paper we have proposed an approach to learn multiple time-varying models jointly. Time-invariant models arise as a special case of the model class that we have considered. The approach is based upon spectral regularization and leads to an optimization problem involving a matrix unknown. To tackle this problem, we have discussed a scalable generalized FW algorithm that has the potential to deal with large scale problems. In relation to this algorithm, we have shown how one can achieve almost effortlessly a significant speed-up in high level implementations, by means of JIT decorators. As a final remark, we note that the proposed approach could be adapted to account for an additional Tikhonov-type penalty with very minor modifications of Algorithm 1. This could be used, in particular, to encode additional information on the similarity between the models. One could also consider stochastic variants and achieve an additional speed-up by avoiding a full pass through the observations. However, care needs to be taken in this case to ensure convergence, which demands for further research.

[7]Note that this yields the same model class as for the TI case above

Acknowledgments

The authors thank Continuum Analytics, Inc for useful support on `numbapro`. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC AdG A-DATADRIVE-B (290923). This paper reflects only the authors' views, the Union is not liable for any use that may be made of the contained information. Research Council KUL: GOA/10/09 MaNet, CoE PFV/10/002 (OPTEC), BIL12/11T; Flemish Government: FWO: projects: G.0377.12 (Structured systems), G.088114N (Tensor based data similarity); PhD/Postdoc grants. IWT: projects: SBO POM (100031); iMinds Medical Information Technologies SBO 2014 Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO, Dynamical systems, control and optimization, 2012-2017).

References

[1] J. Abernethy, F. Bach, T. Evgeniou, and J.P. Vert. A new approach to collaborative filtering: Operator estimation with spectral regularization. *Journal of Machine Learning Research*, 10:803–826, 2009.

[2] A. Argyriou, M. Signoretto, and J. A. K. Suykens. Hybrid conditional gradient-smoothing algorithms with applications to sparse and low rank regularization. In J. A. K. Suykens, M. Signoretto, and A. Argyriou, editors, *Regularization, Optimization, Kernels and support vector machines*. Chapman & Hall/CRC, 2014.

[3] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

[4] J. Baxter. Theoretical models of learning to learn. In S. Thrun and L. Pratt, editors, *Learning to learn*, pages 71–94. Kluwer Academic Publishers, 1998.

[5] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[6] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Kluwer Academic Publishers, 2004.

[7] R. Caruana. Multitask learning. In S. Thrun and L. Pratt, editors, *Learning to learn*. Kluwer Academic Publishers, 1998.

[8] K. De Brabanter, P. Karsmakers, F. Ojeda, C. Alzate, J. De Brabanter, K. Pelckmans, B. De Moor, J. Vandewalle, and J. A. K. Suykens. LS-SVMlab toolbox user's guide version 1.8. *Internal Report 10-146, ESAT-SISTA, K.U.Leuven (Leuven, Belgium)*, 2010.

[9] M. Fazel, H. Hindi, and S.P. Boyd. A rank minimization heuristic with application to minimum ordersystem approximation. In *Proceedings of the American Control Conference, 2001*, volume 6, pages 4734–4739, 2001.

[10] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[11] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 427–435, 2013.

[12] Prasad A Kulkarni. Jit compilation policy for modern machines. In *ACM SIGPLAN Notices*, volume 46, pages 773–788. ACM, 2011.

[13] A. Maurer and M. Pontil. Excess risk bounds for multitask learning with trace norm regularization. In *Conference on Learning Theory (COLT)*, volume 30, pages 55–76, 2013.

[14] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least squares support vector machines*. World Scientific, 2002.

[15] M. Vogt. Nonparametric regression for locally stationary time series. *The Annals of Statistics*, 40(5):2601–2633, 2012.