frontiers

Regression and classification with spline-based separable expansions

Nithin Govindarajan [1,*], **Nico Vervliet** [1,2] **and Lieven De Lathauwer** [1,2]

[1] *STADIUS, Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10 bus 2446, B-3001 Leuven, Belgium*
[2] *Group Science, Engineering and Technology, KU Leuven KULAK, E. Sabbelaan 53, B-8500 Kortrijk, Belgium*

Correspondence*:
Corresponding Author
Nithin.Govindarajan@kuleuven.be

ABSTRACT

We introduce a supervised learning framework for target functions that are well approximated by a sum of (few) separable terms. The framework proposes to approximate each component function by a B-spline, resulting in an approximant where the underlying coefficient tensor of the tensor product expansion has a low-rank polyadic decomposition parametrization. By exploiting the multilinear structure, as well as the sparsity pattern of the compactly supported B-spline basis terms, we demonstrate how such an approximant is well-suited for regression and classification tasks by using the Gauss–Newton algorithm to train the parameters. Various numerical examples are provided analyzing the effectiveness of the approach.

Keywords: B-splines, tensor decompositions, canonical polyadic decomposition, supervised learning, Gauss–Newton, classification, regression

1 INTRODUCTION

Approximating multivariate functions in high dimensions quickly becomes infeasible due to the curse of dimensionality. Negative results in the literature (DeVore et al., 1989; Yarotsky, 2017) reveal that to approximate a generic n-times differentiable function in D variables within ϵ-tolerance (measured in the uniform norm), one typically would require $M \gtrsim \epsilon^{-D/n}$ parameters. Fortunately, in machine learning practice, many target functions are of inherently low complexity if examined through the right lens. That is, by selecting a suitable architecture for the approximant such that it resembles the underlying structure of the true function, the curse of dimensionality can typically be avoided. For instance, a common architecture, adapted particularly by deep neural networks (Schmidhuber, 2015), is to express the approximant as a sequence of compositions of simpler functions. The success of such kind of networks in supervised learning tasks has been profound and can be largely attributed to the fact that many phenomena in nature are the result of a sequence of simpler operations; see (Mhaskar and Poggio, 2016).

In this paper, we study another commonly occuring structure in which the target function $f(\boldsymbol{x})$ essentially has low rank and can be expressed as a sum of few separable terms, i.e.,

$$f(\boldsymbol{x}) = \sum_{r=1}^{R} \left(\prod_{d=1}^{D} \phi_{r,d}(x_d) \right) =: [\![\boldsymbol{\Phi}_1(x_1), \ldots, \boldsymbol{\Phi}_D(x_D)]\!] , \tag{1}$$

where $\mathbf{\Phi}_d(x_d) =: \begin{bmatrix}\phi_{1,d}(x_d) & \cdots & \phi_{R,d}(x_d)\end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix}x_1 & \cdots & x_D\end{bmatrix}$. Sums of separable functions describe continuous, infinite-dimensional analogs of (canonical) polyadic decompositions (CPD) of higher-order tensors (Kolda and Bader, 2009; Cichocki et al., 2015; Sidiropoulos et al., 2017), and a direct discretization of the univariate component functions $\phi_{r,d}(x_d)$ results in an approximant in the form of a tensor product expansion, where the underlying coefficient tensor has a low-rank polyadic decomposition parameterization.

We present our initial findings on how one can effectively approximate functions of the type (1) in the context of supervised learning, appearing either in the form of regression or classification tasks. As opposed to earlier work on this topic (Beylkin et al., 2009; Garcke, 2010), a key feature in our framework is the choice to use B-splines or, equivalently, piecewise polynomials, as a means to discretize the individual component functions. B-splines have been a fundamental tool in computational geometry and numerical analysis for many years (De Boor, 1978), however, their explicit usage in pure machine learning applications has been fairly limited, despite the close connections between deep ReLU networks and nonuniform linear splines with adaptive knots (Unser, 2019). B-splines satisfy, in comparison to approximation by pure polynomials, some favorable properties as they are compactly supported—B-splines are nonzero only on a small interval—and allow for more adaptive local approximation of nonlinearities in a function. Furthermore, they are less prone to the unwanted high-oscillatory behavior that may occur in interpolatory fits of function data using high-degree polynomials.

We present an optimization framework to train the model parameters of the proposed approximant given data samples of a specific regression or classification problem. By introducing the low-rank structure, the problem can be viewed as a special case of a linear system with a CPD constrained solution (Boussé et al., 2018) or the computation of a CPD of an incomplete tensor with linearly constrained factor matrices (Vervliet et al., 2017). By exploiting the resulting multilinear structure of the quadratic objective function in a similar way as in (Boussé et al., 2018; Vervliet et al., 2017), we show that the training of the model parameters in the regression problem can be handled effectively with the Gauss–Newton (GN) algorithm with dog-leg trust region(Nocedal and Wright, 2006). In comparison to prior work which used alternating least squares (ALS) scheme (Beylkin et al., 2009; Garcke, 2010) as means to train the model parameters, the GN algorithm is known to exhibit superior convergence properties; see e.g., (Sorber et al., 2013; Vervliet and De Lathauwer, 2019). Building on the results for the GN-based computation of a CPD using alternative cost functions (Vandecapelle et al., 2020a), we show that our algorithm can be altered to accommodate logistic cost functions which are more suitable for classification problems.

The use of tensor decompositions in machine learning problems has been explored in various directions in the past and is closely related to sum-product networks (Poon and Domingos, 2011; Delalleau and Bengio, 2011; Gens and Domingos, 2012; Gens and Pedro, 2013). Apart from the former work on sums of separable functions (Beylkin et al., 2009; Garcke, 2010), the utility of other types of tensor decompositions have also been explored. (Grelier et al., 2018; Liu et al., 2019; Hou and Chaib-Draa, 2015) looked into applications of (hierachical)-Tucker decompositions in machine learning. (Oseledets, 2013; Novikov et al., 2016; Chen et al., 2017) considered the utility of tensor train decompositions in high-dimensional function approximation. (Hendrikx et al., 2019) used symmetric tensor decompositions to exploit structure in multivariate polynomial functions. Finally, (Hashemizadeh et al., 2020) looked into adaptive method to learn tensor networks from data. However, none of the prior works have considered using B-splines extensively in the provided setting before.

The paper is organized as follows. First, in Section 2, we cover some basic mathematical preliminaries on B-splines, low-rank separable expansions, and tensors. Subsequently, the regression and classification

frameworks are introduced in Sections 3 and 4, respectively. Herein, we cover also some numerical examples illustrating the utility of the proposed learning framework. The conclusions are provided in Section 5.

Notation

The following notation is adopted throughout this paper. The symbol $\mathbb{R}$ is reserved to denote the reals. $C([0,1]^D)$ is used to denote the space of continuous function on the unit hypercube $[0,1]^D$. Scalars, (column) vectors, matrices and tensors are denoted by lowercase, bold lowercase, bold uppercase and calligraphic characters, respectively; e.g., $s \in \mathbb{R}$, $\mathbf{v} \in \mathbb{R}^I$, $\mathbf{M} \in \mathbb{R}^{I \times J}$ and $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_D}$. $\mathbf{1}_N \in \mathbb{R}^N$ denotes the vector with all entries set to one. The symbols $\otimes$ and $*$ denote the outer product and the Hadamard, or element-wise product, respectively. The mode-d tensor-matrix product $\cdot_d$ between a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_D}$ and a matrix $\mathbf{M} \in \mathbb{R}^{J \times I_d}$ has the following elementwise definition for $\mathcal{S} = \mathcal{T} \cdot_d \mathbf{M} \in \mathbb{R}^{I_1 \times \ldots \times I_{d-1} \times J \times I_{d+1} \times \ldots I_D}$:

$$s_{i_1,\ldots,i_{d-1},j,i_{d+1},\ldots,i_D} = \sum_{i_d=1}^{I_D} t_{i_1,\ldots,i_{d-1},i_d,i_{d+1},\ldots,i_D} m_{j,i_d}.$$

A polyadic decomposition can be written compactly as

$$[\![\boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D]\!] \coloneqq \sum_{r=1}^{R} \boldsymbol{\gamma}_{r,1} \otimes \cdots \otimes \boldsymbol{\gamma}_{r,D},$$

where factor matrix $\boldsymbol{\Gamma}_d = \begin{bmatrix} \boldsymbol{\gamma}_{1,d} & \cdots & \boldsymbol{\gamma}_{R,d} \end{bmatrix}$ collects factor vectors $\boldsymbol{\gamma}_{r,d}$ as its columns. The bracket notation is also used to compactly express sums of separable functions, i.e.,

$$[\![\boldsymbol{\Phi}_1(x_1), \ldots, \boldsymbol{\Phi}_D(x_D)]\!] \coloneqq \sum_{r=1}^{R} \left(\prod_{d=1}^{D} \phi_{r,d}(x_d) \right),$$

where $\boldsymbol{\Phi}_d(x_d) = \begin{bmatrix} \phi_{1,d}(x_d) & \cdots & \phi_{R,d}(x_d) \end{bmatrix}$.

2 PRELIMINARIES: B-SPLINES, LOW RANK SEPARABLE EXPANSIONS AND TENSORS

In this section, we cover the basic mathematical prelimaries for our proposed regression and classification frameworks which are discussed in Sections 3 and 4, respectively.

2.1 B-splines

A spline is a piecewise polynomial function stitched together in a way such that it maintains a certain degree of smoothness over the approximation interval. A convenient way of expressing splines is through B-splines. A B-spline of order N on the interval $[a,b]$ is constructed as follows. Let $\mathscr{T} = \{t_i\}_{i=0}^{N+M}$ denote the set of knots which satisfy the order relation

$$a = t_0 = \ldots = t_{N-1} \leq t_N \leq t_{N+1} \leq \ldots \leq t_{M+1} = \ldots = t_{M+N} = b.$$

The B-spline basis terms $\{B_{m,N}\}_{m=0}^{M}$ are given by the recursion formula

$$B_{m,N}(x) \coloneqq \frac{x - t_m}{t_{m+N} - t_m} B_{m,N-1}(x) + \frac{t_{m+N+1} - x}{t_{m+N+1} - t_{m+1}} B_{m+1,N-1}(x), \tag{2}$$

where

$$B_{m,0}(x) := \begin{cases} 1 & x \in [t_m, t_{m+1}) \\ 0 & \text{otherwise} \end{cases}.$$

In Figure 1, the B-spline basis terms are graphically depicted for various orders and knot configurations. The B-spline basis terms satisfy the interesting property of being compactly supported. In particular, $B_{m,N}$ can only attain nonzero values on the interval $[t_m, t_{m+N+1})$, i.e.,

$$B_{m,N}(x) = 0, \qquad x \in (-\infty, t_m) \cup [t_{m+N+1}, \infty). \tag{3}$$

To put this into perspective, a polynomial basis can never satisfy such a property: if a polynomial function is zero on a nonempty interval, then it must be zero everywhere. A general B-spline $S(x)$ of order N is formed by taking weighted linear combinations of the basis elements (2). We have the expression:

$$S(x) = \sum_{m=0}^{M} c_m B_{m,N}(x),$$

which we denote more concisely in the vectorized notation

$$S(x) = \begin{bmatrix} B_{0,N}(x) & \cdots & B_{M,N}(x) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_M \end{bmatrix} = \mathbf{B}_{\mathscr{T},N}(x)\boldsymbol{c}. \tag{4}$$

For the sake of readability, we omit the subscripts $\mathscr{T}, N$ in $\mathbf{B}_{\mathscr{T},N}$ in the remainder of the text, i.e., $\mathbf{B}(x) = \mathbf{B}_{\mathscr{T},N}(x)$. However, one must keep in mind that a spline always implicitly involves a set of knots $\mathscr{T}$ and an order N.

It is well-known that (4) is a universal approximator and has the ability to approximate any continuous function with arbitrary accuracy (De Boor, 1978). This fact should come as no surprise, since (4) describes a piecewise degree $N-1$ polynomial, and polynomials themselves are universal approximators as a consequence of the Weierstrass theorem. However, there is a fundamental feature of splines which allow functions to be approximated in a way that pure polynomials cannot: while the approximation power of pure polynomials can only be increased by raising the degree, in the case of splines, one has additional flexibility to raise the total number of polynomial pieces used on the approximation interval. This increase is effectively realized by increasing the knot density.

There are several advantages of being able to control the amount of polynomial pieces. First, it allows for local approximation of regions over which the function experiences dramatic change, since changes to the coefficients in (4) affect the function only locally. Second, by keeping the degree low and increasing the number of knots, one can avoid some of the numerical issues faced by high-degree polynomials. A major drawback of high-degree polynomials is the possible appearance of large (non-physical) oscillations in an interpolatory fit of a function which increase in magnitude as the polynomial degree increases. This undesirable property is commonly referred to as Runge's phenomenon in the approximation theory literature; see e.g., (Trefethen, 2019) for a more detailed exposition on the subject. In the context of machine-learning, large oscillations between the sample points can have an adverse on the generalization error. Henceforth, any use of high-degree polynomials in such contexts must keep Runge's effects at check, i.e., the method of approximation becomes of paramount approximation. This can be achieved in

various ways; e.g., by overparameterizing the polynomial and adding a derivative penalty to the polynomial fit (Chandrasekaran et al., 2013; Rajagopal, 2019), or by using structured interpolation points such as Chebyshev nodes if one has the freedom to pick the function samples freely (Trefethen, 2019). Another alternative is to switch to splines, in which case, the issue is completely bypassed altogether if one keeps the order of the spline low. Third, resolving local phenomena such as kinks, discontinuities, and sharp transitions is difficult using polynomials due to their global nature. Splines, however, can capture local phenomena more easily thanks to their compact support, particularly if the knots are chosen to align with the boundaries of such features.

The performance of a given B-spline for a regression task is dependent on the chosen set of knots and order of the spline. In practice, one typically wants to optimize the number of knots as well as their respective positions in order to avoid under or overfitting of the data. Typically, the idea is to increase the knot frequency at data-dense regions where the underlying function tends to exhibit a high degree of variation. The problem of optimal knot allocation is, however, highly non-trivial and the literature has suggested various approaches to do adaptive knot selection; see (Zhou and Shen, 2001) and the references therein. Fortunately, in most cases, decent performance can already be achieved by distributing the knots evenly with respect to the provided data (De Boor, 1978).

2.2 Multivariate splines and tensors

There are several ways to extend B-splines to the multivariate case. Instead of forming piecewise polynomials on a set of intervals, one constructs a smooth function in a higher-dimensional space by stitching together piecewise polynomials on a collection of polygonal domains. One well-studied framework is to define splines on so-called triangulations which partitions a domain into simplices (Lai and Schumaker, 2007).

A more basic approach, which is more amenable to our pursuit of low-rank approximations, is to take Cartesian tensor products of univariate B-splines. This effectively results in a construction where polynomials are defined piecewise on box-shaped domains. To define a tensor product B-spline on a feature space $[a_1, b_1] \times \ldots \times [a_D, b_D]$, a set of knots $\mathscr{T}_d$ for each variable x_d, $d = 1, \ldots, D$, is defined first. A multivariate function can then be approximated by

$$S(\boldsymbol{x}) = \sum_{m_1=0}^{M_1} \sum_{m_2=0}^{M_2} \cdots \sum_{m_D=0}^{M_D} c_{m_1 m_2 \cdots m_D} \prod_{d=1}^{D} B^{(d)}_{m_d, N_d}(x_d).$$

The above expansion can be expressed more concisely by using tensor notation. With the help of mode-d tensor-matrix products, we may write

$$S(\boldsymbol{x}) = \mathcal{C} \cdot_1 \mathbf{B}_1(x_1) \cdot_2 \mathbf{B}_2(x_2) \cdots \cdot_D \mathbf{B}_D(x_D), \tag{5}$$

where $\mathcal{C} \in \mathbb{R}^{(M_1+1)\times\cdots\times(M_D+1)}$ denotes the Dth order coefficient tensor, and $\mathbf{B}_d(x_d) = \begin{bmatrix} B^{(d)}_{1,N_d}(x_d) & \cdots & B^{(d)}_{M_d,N_d}(x_d) \end{bmatrix}$ for some knot set $\mathscr{T}_d$, $d = 1, \ldots, D$. Similar to how (4) is an universal approximator of any continous function on an interval, (5) describes an universal approximator for any multivariate continuous function on the feature space $[a_1, b_1] \times \cdots \times [a_D, b_D]$. Likewise, as a consequence of (3), the basis elements of (5) are also compactly supported.

2.3 Low-rank separable expansions and CPDs

The number of entries in the coefficient tensor $\mathcal{C} \in \mathbb{R}^{(M_1+1)\times\cdots\times(M_D+1)}$ equals $\prod_{d=1}^{D}(M_d+1)$ and grows exponentially with the dimension D of the function domain. This exponential blow-up of the parameter space, which also occurs in splines on triangulations (Lai and Schumaker, 2007), is commonly known as the curse of dimensionality (CoD) by the tensor research community CoD makes problems such as construction and evaluation of the tensor product B-spline (5) intractable. Regardless, for functions which are expressible by a sum of a few separable terms (1), i.e.,

$$f(\boldsymbol{x}) = \sum_{r=1}^{R}\left(\prod_{d=1}^{D}\phi_{r,d}(x_d)\right),$$

approximation by the expansion (5) is grossly inefficient. For such functions, the tensor $\mathcal{C}$ in (5) is of low rank and has a more concise parameterization in the form of a polyadic decomposition into R terms:

$$\mathcal{C}(\boldsymbol{\Gamma}_1,\ldots,\boldsymbol{\Gamma}_d) = [\![\boldsymbol{\Gamma}_1,\ldots,\boldsymbol{\Gamma}_D]\!] = \sum_{r=1}^{R}\boldsymbol{\gamma}_{r,1}\otimes\cdots\otimes\boldsymbol{\gamma}_{r,D}, \tag{6}$$

in which $\boldsymbol{\Gamma}_d$ are the factor matrices of the CPD. To confirm that such a low-rank parameterization is actually possible, one simply has to approximate each component function $\phi_{r,d}$ in (1) by a B-spline

$$\phi_{r,d}(x_d) \approx \mathbf{B}_d(x_d)\boldsymbol{\gamma}_{r,d}. \tag{7}$$

Substituting (7) into (1) results then into the more compact approximant

$$\hat{f}(\mathbf{x};\boldsymbol{\Gamma}_1,\ldots,\boldsymbol{\Gamma}_D) = \mathcal{C}(\boldsymbol{\Gamma}_1,\ldots,\boldsymbol{\Gamma}_D)\cdot_1\mathbf{B}_1(x_1)\cdot_2\mathbf{B}_2(x_2)\cdots\cdot_D\mathbf{B}_D(x_D), \tag{8}$$

where $\mathcal{C} : \mathbb{R}^{(M_1+1)\times R}\times\cdots\times\mathbb{R}^{(M_D+1)\times R}\rightarrow\mathbb{R}^{(M_1+1)\times\cdots\times(M_D+1)}$ is now described by the polyadic decomposition (6). Indeed, in comparison to (5), the approximant (8) can have orders of magnitude fewer parameters if R is low: only $R(\sum_{d=1}^{D}M_d+1)$ parameters are needed.

The ability to approximate functions efficiently through (8) rests on the assumption that the number of separable terms R in (1) are small. Classical results in approximation theory reveal that the level of difficulty in approximating a function can be characterized roughly by its degree of smoothness. Specifically, the rate at which a multivariate function can be approximated by a tensor product expansion of an orthogonal (polynomial) basis is proportionally tied to how many continuous derivatives the function possesses (measured in the Sobolev norm) (Mhaskar and Pai, 2000). Subsequently, a very smooth function which has a commensurate number of continuous derivatives with respect to the number of variables can be very compacty approximated with a few terms in (1). However, such an analysis does not describe the whole picture at all.

The smoothness of a function does not alone capture the "rank of a function". The class of functions well representable through (1) is much richer, as a function can be easily nonsmooth but still of low rank; for example, any function of a sum of separable terms in which several of the component functions are nonsmooth. The work in (Beylkin and Mohlenkamp, 2005; Boussé et al., 2017; Khoromskij, 2018) discusses some interesting mechanisms under which multivariate functions can have good low-rank approximations, which are, for instance, achieved through sums of exponentials or sinc functions. At a very intuitive level, the low-rankness of a function captures the low complexity of a function. The bottom line is

that negative results in (DeVore et al., 1989; Yarotsky, 2017) (see introduction) put a fundamental "speed limit" to general robust approximation of a function, which is exponential in the number of dimensions, but simultaneously attenuated by the degree of smoothness. Any approach more effective must somehow utilize some additional structure in some way. In the case of (8), this additional structure comes in the form of a low-rank tensor, which forms the basis of the success of tensor-based scientific computing (Khoromskij, 2018; Grasedyck et al., 2013; Hackbusch, 2012).

3 REGRESSION

In a regression context, the aim is to learn a function $\hat{f}(\mathbf{x})$ as defined in (8) that approximates the unknown target function $f \in C([0,1]^D)$ defined on a D-dimensional feature space, from a set of I samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^I \subset [0,1]^D \times \mathbb{R}$. To achieve this, we minimize the quadratic objective

$$Q(\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) \coloneqq \frac{1}{2} \sum_{i=1}^{I} \left(\hat{f}(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) - y_i \right)^2. \tag{9}$$

This problem can be interpreted in two ways as discussed in Subsection 3.1. Next, we derive the GN-based algorithm that exploits the low-rank and B-spline structure. Technical details have been moved to Appendix A for readability. To conclude this section, the practical feasibility of the proposed combination of low-rank and B-spline structure is illustrated numerically using synthetic and real-life data in Section 3.3.

3.1 Two interpretations

While a GN-based algorithm is derived in the next subsection, it can be fruitful to discuss links with two other frameworks. First, the objective function (9) can be seen as a linear system of which the solution has a CPD constraint, i.e.,

$$\mathbf{A}\mathbf{c} = \mathbf{y},$$

in which the solution $\mathbf{c} = \text{vec}\,(\mathcal{C}) = \text{vec}\,(\llbracket \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D \rrbracket)$, and $\mathbf{y} = \begin{bmatrix} y_1 & \ldots & y_I \end{bmatrix}^T$. Using the definition of the mode-d tensor-matrix product, we can see that a row $\mathbf{a}_i^{\mathrm{T}}$ in $\mathbf{A} \in \mathbb{R}^{I \times \prod_{d=1}(M_d+1)}$ is given by

$$\mathbf{a}_i^{\mathrm{T}} = \bigotimes_{d=1}^{D} \mathbf{B}_d(x_{i,d}) \tag{10}$$

in which $\otimes$ is the Kronecker product and $\mathbf{B}_d(x_{i,d})$ is a row vector of evaluated B-splines in point $x_{i,d}$. While a general solver such as (Boussé et al., 2018) can be used, exploiting the Kronecker structure is crucial for achieving an efficient algorithm; see, e.g., (Hendrikx et al., 2019) for a polynomial example that also involves symmetry. As (10) does not involve symmetry, a new algorithm is derived here by exploiting the Kronecker structure as well as the B-spline basis structure.

Second, problem (9) can be seen as a decomposition of a Dth-order incomplete tensor $\mathcal{Y}$ in which the dth dimension J_d is the number of unique values $x_{i,d}$ over all samples i, hence $J_d \leq I$. The number of known entries in $\mathcal{Y}$ is equal to I. By denoting the evaluated spline basis matrices $\mathbf{B}_d$ in these unique values $x_{i,d}$ by $\hat{\mathbf{B}}_d$ and by using multilinear calculus to rewrite (8), an incomplete tensor fitting problem is obtained in which a linear constraint is imposed on each factor matrix:

$$\min_{\{\mathbf{\Gamma}_d\}_{d=1}^D} \frac{1}{2} \left|\left| \mathcal{S} * \left(\mathcal{Y} - \left\llbracket \hat{\mathbf{B}}_1 \mathbf{\Gamma}_1, \hat{\mathbf{B}}_2 \mathbf{\Gamma}_2, \ldots, \hat{\mathbf{B}}_D \mathbf{\Gamma}_D \right\rrbracket \right) \right|\right|_{\mathrm{F}}^2, \tag{11}$$

where $\mathcal{S}$ is a binary sampling tensor which is one if the entry is known or sampled, and zero otherwise. A GN algorithm for solving problem (11) is presented in (Vervliet et al., 2017). In this paper, we improve upon this result by further exploiting the sparsity of the basis matrices thanks to the B-spline structure.

3.2 Gauss–Newton algorithm

A basic descent algorithm to solve (9) is the alternating least squares (ALS) scheme (Harshman, 1970). In ALS, one exploits the multilinear structure (6) by freezing all but one of the factor matrices, so that the problem essentially becomes linear at each sub-iteration. Alternatively one may also approach the problem using a GN procedure, which has been shown to have superior convergence properties for tensor related problems with respect to ALS; see, e.g., (Sorber et al., 2013; Vervliet and De Lathauwer, 2019; Vervliet et al., 2017).

In this section, we derive the necessary ingredients—the gradient $\mathbf{g}_{r,d}$ and the GN step $\mathbf{p}_{r,d}$, $r = 1, \ldots, R$, and $d = 1, \ldots, D$—for a GN type algorithm for solving (9). The dogleg trust region approach is used to ensure global convergence as this has proven successful in a tensor context (Sorber et al., 2013; Vervliet and De Lathauwer, 2019). As the dogleg algorithm is a standard algorithm in optimization—see, e.g., (Nocedal and Wright, 2006)—we assume variables $\boldsymbol{\gamma}_{r,d}$ are updated from iteration l to $l+1$ as

$$\boldsymbol{\gamma}_{r,d}^{(l+1)} \leftarrow \boldsymbol{\gamma}_{r,d}^{(l)} + \mathbf{u}_{r,d}^{(l)}, \quad r = 1, \ldots, R, \quad d = 1, \ldots, D,$$

in which $\mathbf{u}_{r,d}^{(l)}$ is the dogleg step. (In the remainder, the iteration superscript $\cdot^{(l)}$ is dropped for simplicity of notation.)

To show that the GN approximation is an appropriate choice for solving problem (9), consider first the Newton step $\tilde{\mathbf{p}}_{r,d}$, which is obtained by solving the linear system of equations

$$\sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \left(\frac{\partial^2 Q(\boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d} \partial \boldsymbol{\gamma}_{\tilde{r},\tilde{d}}} \right) \tilde{\mathbf{p}}_{\tilde{r},\tilde{d}} = -\frac{\partial Q(\boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d}} = -\mathbf{g}_{r,d}, \qquad r = 1, \ldots, R, \quad d = 1, \ldots, D. \tag{12}$$

The left-hand-side matrix is the Hessian of (9), while the right-hand side is the gradient $\mathbf{g}_{r,d}$. Because of the low-rank structure in the coefficient tensor $\mathcal{C}$, both the gradient and Hessian terms in (12) are highly structured. A meticulous derivation of these quantities allows us to express them concisely. Let us define

$$v_{r,d;i} = v_{r,d}(\mathbf{x}_i; \boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D) \coloneqq \prod_{k=1, k \neq d}^{D} \mathbf{B}_k(x_{i,k}) \boldsymbol{\gamma}_{r,k},$$

$$w_{r,d;\tilde{r},\tilde{d};i} = w_{r,d;\tilde{r},\tilde{d}}(\mathbf{x}_i; \boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D) \coloneqq \begin{cases} 0 & d = \tilde{d} \\ \psi_{d,\tilde{d}}(\boldsymbol{x}_i; \boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D) & r = \tilde{r} \\ 0 & r \neq \tilde{r} \end{cases},$$

$$\psi_{d,\tilde{d};i} = \psi_{d,\tilde{d}}(\mathbf{x}_i; \boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D) \coloneqq \prod_{k=1, k \neq d, \tilde{d}}^{D} \mathbf{B}_k(x_{i,k}) \boldsymbol{\gamma}_{\tilde{r},k}.$$

The gradient and Hessian terms can then be expressed as

$$\frac{\partial Q(\mathbf{\Gamma}_1,\ldots,\mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d}} = \sum_{i=1}^{I} \eta_i v_{r,d;i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}), \tag{13}$$

$$\frac{\partial^2 Q(\mathbf{\Gamma}_1,\ldots,\mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d} \partial \boldsymbol{\gamma}_{\tilde{r},\tilde{d}}} = \sum_{i=1}^{I} \xi_i \left(v_{r,d;i}\mathbf{B}_d^{\mathrm{T}}(x_{i,d})\right)\left(v_{\tilde{r},\tilde{d};i}\mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}})\right) + \eta_i w_{r,d;\tilde{r},\tilde{d};i}\mathbf{B}_d^{\mathrm{T}}(x_{i,d})\mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}), \tag{14}$$

in which $\eta_i \in \mathbb{R}$ is the residual, and $\xi_i \in \mathbb{R}$ is a weight term:

$$\eta_i = \eta(\mathbf{x}_i, y_i; \mathbf{\Gamma}_1,\ldots,\mathbf{\Gamma}_D) \coloneqq \hat{f}(\mathbf{x}_i; \mathbf{\Gamma}_1,\ldots,\mathbf{\Gamma}_D) - y_i, \tag{15}$$

$$\xi_i \coloneqq 1. \tag{16}$$

(The weight ξ_i will be important for the classification case; see Section 4.) Note that (13) and (14) consist of only I terms, i.e., the number of samples.

In the GN procedure, the second term in the Hessian is dropped so that (14) reduces to the Gramian of the Jacobian:

$$\mathbf{G}_{r,d;\tilde{r},\tilde{d}} \coloneqq \sum_{i=1}^{I} \xi_i \left(v_{r,d;i}\mathbf{B}_d^{\mathrm{T}}(x_{i,d})\right)\left(v_{\tilde{r},\tilde{d};i}\mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}})\right). \tag{17}$$

This Gramian can be a good approximation of the Hessian as the second term depends on the residual η_i, which becomes small near a global optimum and zero if an exact solution exists, i.e., if $Q(\mathbf{\Gamma}_1,\ldots,\mathbf{\Gamma}_D) = 0$. Moreover, the term $w_{r,d;\tilde{r},\tilde{d};i}$ is often zero. The Gramian is positive semidefinite and allows for a locally convex approximation of the objective function. By dropping the second term in (14), the system

$$\sum_{\tilde{d}=1}^{D}\sum_{\tilde{r}=1}^{R} \mathbf{G}_{r,d;\tilde{r},\tilde{d}}\mathbf{p}_{\tilde{r},\tilde{d}} = -\frac{\partial Q(\mathbf{\Gamma}_1,\ldots,\mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d}}, \qquad r = 1,\ldots,R, \quad d = 1,\ldots,D, \tag{18}$$

is solved in lieu of (12).

There are several provisions taken to solve the linear system (18) in an effective manner. Similar to earlier work (Sorber et al., 2013; Vervliet and De Lathauwer, 2019), system (18) is solved iteratively using the conjugate gradient (CG) method. The CG matrix only requires evaluations of matrix-vector products—in (18), the matrix is the Gramian—which can be computed efficiently by exploiting the structure in the problem in analogous way as done in (Hendrikx et al., 2019; Vervliet et al., 2017); see Appendix A for the exact algorithmic details.

While CG requires a positive definite matrix, the method works for the positive semidefinite Gramian as well in practice, as both the starting point—the Cauchy point—for the CG method and the right hand side of (18) are both orthogonal to the null space of the Gramian. As the trust-region approach ensures sufficient decrease, which is needed for convergence of the GN method, the number of (inner) CG iterations can be limited, lowering the computational cost further.

A key benefit of using splines is that the Gramian (18) enjoys an additional sparsity structure on top of the multilinear structure already caused by CPD constraint (6). If one exploits the sparsity pattern in the spline basis (recall that, unlike polynomials, the spline basis elements are compactly supported), the time complexity for evaluating the gradient and Gramian-vector product are both $\mathcal{O}(INRD)$ flop. Herein,

$N := \max N^{(d)}$ denotes the maximum order of the splines used in (8). In practice, N is intentionally kept low (typically $N = 4$) in order to avoid usage of high-degree polynomials in the approximation. The complexity then effectively reduces down to $\mathcal{O}(IRD)$ flop. These savings are significant if one takes into consideration the maximum number of spline coefficients in the univariate component functions, i.e., $M := \max M^{(d)}$. A failure to take advantage of the sparsity structure would have increased the complexity to $\mathcal{O}(IMRD)$ flop. This illustrates another advantage of B-splines over pure polynomials as there is no sparsity pattern to be exploited in the latter case.

3.3 Numerical examples

A Matlab implementation of the proposed GN algorithm for the objective function (9) has been made. For the tensor and optimization related functions we use Tensorlab 3.0 (Vervliet et al., 2016). Next, we cover some interesting examples that examine the behavior of the proposed regression framework.

3.3.1 Case study I

We will start off with a function which is nonsmooth but of low rank, i.e.,

$$f(\boldsymbol{x}) = |x_1||x_2| + \sin(2\pi x_1)\cos(2\pi x_2) + x_1^2 x_2, \quad \boldsymbol{x} \in [-1, 1] \times [-1, 1]. \tag{19}$$

The function (19) has a kink on the lines $x_1 = 0$ and $x_2 = 0$ which is caused by the first separable term $|x_1||x_2|$. Despite of its nonsmooth features, it is still of low rank, particularly $R = 3$. Subsequently, (19) can be well approximated with $\mathcal{O}(M)$ terms using (8), instead of the $\mathcal{O}(M^2)$ terms needed with (5). In Figure 3 we actually demonstrate this numerically. The figure contains a plot of the absolute error between the obtained approximant (8) by solving (9) using the GN algorithm with random initialization and the true function (19). For our experiment, we took 8000 random samples of the function uniformly over the domain for our training set. For each of the component functions (7), we chose a uniform knot distribution with $M = 31$ and $N = 4$. From the plots, one can clearly see that the first two low-rank approximations (i.e., $R = 1$, and $R = 2$) are of poor quality. This is expected, since a rank-one or a rank-two approximation cannot capture the full complexity of the function. In fact, the results will not even improve if more data samples are taken and the number of knots are increased for each of the component functions (7). The rank-three approximation is on the other hand of significantly better quality. This is expected because the true function (19) also contains three separable terms. Furthermore, the higher-rank approximations with $R > 3$ do not seem to add much value to the approximation. This also exhibits the property that the rank can be overestimated without loss in quality.

To test the effectiveness of using B-splines, we compared our results with how the approximation would look like if Chebyshev polynomials were used instead. It is well-known that the function $f(x) = |x|$ can be easily subjugated to the Runge phenomenon. Without delving too much into the technical details, bad fits can particularly arise if the points picked are uniformly distributed and the degree of the polynomials is close to the number of function samples; see Figure 2. Intuitively, one would expect that Runge's effects would re-appear also in the fit of (19). In Figure 4, this is actually confirmed. Apart from the quality of approximation, Figure 5 displays the average computation time required to pass through a single iteration of the GN algorithm for the B-spline basis as well as the Chebyshev basis. From the figure it can be clearly deduced that the computation time grows proportionally with the size of the Chebyshev basis, whereas it remains constant for the spline basis if only the number of knots are increased.

3.3.2 Case study II

The next example which we consider looks very similar to our first example, the only exception being that the first separable term is "rotated" by 45 degrees:

$$f(\boldsymbol{x}) = \left|\frac{\sqrt{2}}{2}x_1 - \frac{\sqrt{2}}{2}x_2\right|\left|\frac{\sqrt{2}}{2}x_1 + \frac{\sqrt{2}}{2}x_2\right| + \sin(2\pi x_1)\cos(2\pi x_2) + x_1^2 x_2, \quad \boldsymbol{x} \in [-1,1]\times[-1,1]. \tag{20}$$

This change is significant, however, since the rotation breaks the rank-one structure of the first term. Unlike our previous example, the function (20) has rank strictly greater than $R = 3$. Given that the nonsmooth portion of the function is no longer aligned with the coordinates x_1 and x_2, one would expect that an approximation with (8) would do quite poorly. In comparison to Case study I, this is also the case. Figure 6 displays the obtained results for different rank values using the exact same configuration as in Figure 3. The absolute errors are clearly larger. Surprisingly, however, the global trends of the function are still well-captured with a rank $R = 3$ approximation. Furthermore, a rank $R = 6$ approximation is of comparable quality to its counterpart of Figure 3. The results indicate that, even if a function may be mathematically of high rank, good approximations that capture the global features of the function can already be achieved with relatively few rank-one terms.

3.3.3 Case study III: NASA airfoil self noise

The final example which we consider is the task of fitting an experimentally obtained dataset. We examine the NASA dataset (Brooks et al., 1989) obtained from the UCI machine learning repository (Dua and Graff, 2017) which comprises of data concerning different size NACA 0012 airfoils subjected to various wind tunnel speeds and angles of attack. The goal is to model the sound, or self noise generated by the airfoil, as a function of the frequency, angle of attack, chord length, free-stream velocity, and the suction-side displacement thickness. The dataset has been normalized and contains 1503 measurements in total. We randomly split the dataset into a training dataset (1202 samples) and a test dataset (301 samples). Figure 7 displays the obtained results by using the approximant (8) to fit the data by means of minimizing the objective function (9). As in the previous examples, a uniformly distributed knot distribution was chosen for the component functions (7), and shown in Figure 7 is the root mean squared error (RMSE) and maximal absolute error obtained for various configurations. Generating a fit directly with (5) would have been prohibitively expensive, since the dependent variable is a function of five variables. However, Figure 7 reveals that the target function possesses low-rank structure, and can be well modeled by $R = 5$ separable terms, where we see a sudden drop in the training and test error. Furthermore, it shows that the inclusion of any additional separable terms does not amount to significant performance improvements of the fit.

4 CLASSIFICATION

If the goal is to classify samples into two classes, other loss functions than least squares (9) are more suitable. In Subsection 4.1, we show how the logistic loss function can be combined with B-splines and the low-rank approximation of the coefficient tensors following (8). Next, the GN algorithm is generalized to accommodate this loss function in Subsection 4.2. Finally, the utility of low-rank approximations in combination with B-splines for classification problems is illustrated via numerical experiments.

4.1 Logistic cost function

A (binary) classification function defined on a D-dimensional feature space can be modeled as the map $g : [0,1]^D \to \{0,1\}$ given by

$$g(\mathbf{x}) = \begin{cases} 0 & f(\mathbf{x}) \leq 0 \\ 1 & f(\mathbf{x}) > 0 \end{cases},$$

where $f \in C([0,1]^D)$ is a continuous function whose zero level set partitions the domain into the two classification sets. Instead of enforcing a hard threshold between the two sets, one can smoothen this transition by replacing the step function with the logistic function $\sigma_\alpha : t \mapsto 1/(\exp(-\alpha t)+1)$, i.e.

$$g_\alpha(\mathbf{x}) \coloneqq (\sigma_\alpha \circ f)(\mathbf{x}) = \sigma_\alpha(f(\mathbf{x})),$$

where the parameter $\alpha > 0$ controls the gradient in in the transition zone. To incorporate the low-rank constraint and the B-spline basis, the function g_α is replaced by the approximant

$$\hat{g}_\alpha(\mathbf{x}; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) \coloneqq \sigma_\alpha \circ \hat{f}(\mathbf{x}; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D), \tag{21}$$

where $\hat{f}$ refers again to (8). Given a collection of data samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^I \subset [0,1]^D \times \{0,1\}$, where $y_i \in \{0,1\}$ denote the labels, the performance of a classifier $\hat{g}_\alpha$ can be characterized by the tendency of the quantity

$$0 \leq \prod_{y_i=0} (1 - \hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)) \prod_{y_i=1} \hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) \leq 1,$$

to lean towards unity. By applying a logarithmic transformation, the recovery of a good classifier $\hat{g}_\alpha$ can hence be obtained by minimizing the objective function

$$L_\alpha(\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) \coloneqq -\sum_{i=1}^I y_i \log \hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) + (1-y_i) \log\left(1 - \hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)\right). \tag{22}$$

4.2 Generalized Gauss–Newton algorithm

Even though the GN algorithm is derived for least squares problems, it can be generalized easily to accommodate other loss functions. Following general results in (Schraudolph, 2002) and for tensor decompositions (Vandecapelle et al., 2020a,b), the generalized GN algorithm can be derived similarly to the strategy in Subsection 3.2. In the dogleg trust-region framework, the necessary GN direction $\mathbf{p}_{r,d}$ is derived starting from the linear system:

$$\sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \left(\frac{\partial^2 L_\alpha(\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d} \partial \boldsymbol{\gamma}_{\tilde{r},\tilde{d}}} \right) \mathbf{p}_{\tilde{r},\tilde{d}} = -\frac{\partial L_\alpha(\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d}}, \qquad r = 1, \ldots, R, \quad d = 1, \ldots, D,$$

in which the right-hand side is the gradient and is given by

$$\frac{\partial L_\alpha(\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d}} = \sum_{i=1}^I \eta_i v_{r,d;i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}),$$

and the blocks of the Hessian in the left-hand side are given by

$$\frac{\partial^2 L_\alpha(\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d} \partial \boldsymbol{\gamma}_{\tilde{r},\tilde{d}}} = \sum_{i=1}^I \xi_i \left(v_{r,d;i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}) \right) \left(v_{\tilde{r},\tilde{d};i} \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \right) + \eta_i w_{r,d;\tilde{r},\tilde{d};i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}) \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}).$$

The main difference with respect to the least squares objective (9) is that the weights $\eta_i, \xi_i \in \mathbb{R}$ are now given by

$$\eta_i = \eta_i(\mathbf{x}_i, y_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) \coloneqq \alpha \left(\hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) - y_i \right), \tag{23}$$

$$\xi_i = \xi_i(\mathbf{x}_i; \Gamma_1, \ldots, \Gamma_D) \coloneqq \alpha^2 \hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) \left(1 - \hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)\right). \tag{24}$$

Similar to Subsection 3.2, a GN type approximation to the Hessian can be made by neglecting the second term in (24), resulting in the linear system

$$\sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \mathbf{G}_{r,d;\tilde{r},\tilde{d}} \mathbf{p}_{\tilde{r},\tilde{d}} = -\frac{\partial L_\alpha(\mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D)}{\partial \boldsymbol{\gamma}_{r,d}}, \qquad r = 1, \ldots, R, \quad d = 1, \ldots, D, \tag{25}$$

in which

$$\mathbf{G}_{r,d;\tilde{r},\tilde{d}} \coloneqq \sum_{i=1}^{I} \xi_i \left(v_{r,d;i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}) \right) \left(v_{\tilde{r},\tilde{d};i} \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \right).$$

This is again a good approximation to the Hessian as η_i is small if the residual $\hat{g}_\alpha(\mathbf{x}_i; \mathbf{\Gamma}_1, \ldots, \mathbf{\Gamma}_D) - y_i$ is small, η_i is small, and many of the values $w_{r,d;\tilde{r},\tilde{d};i} = 0$. As a consequence of the positivity of the weights (24), the Gramian (17) associated with the logistic objective is also positive semidefinite, and the CG method can again be used to solve the linear system (25) iteratively using only matrix-vector products.

4.3 Numerical examples

The proposed generalized GN has been implemented in Matlab using Tensorlab 3.0 (Vervliet et al., 2016). We examine the behavior of the classification framework with some illustrative examples.

4.3.1 Case study IV

At first, we assess the performance of the framework on some synthetically generated datasets. We consider classification sets that take on increasingly more complex geometric shapes on the two-dimensional plane. Since the geometric shapes are composed of simpler geometric operations, it is expected that a low-rank approximant of type (21) would be sufficient to obtain a decent separation between the two classes. In Figures 8, 9, 10 and 11, these expectations are confirmed, in which the classification sets are recovered by solving (22) for some randomly generated labeled data. The plots in the figures reveal that by increasing the number of separable terms in (21), better classification can be achieved. The number of separable terms needed to get a decent separation between the two classes is roughly proportional to how "complex" the classification sets themselves are. The circular shaped set (Figure 8) required only $R = 2$ terms for a decent separation. From a theoretical standpoint, this is expected, since one can model a circular boundary by the level sets of the rank-two separable function

$$f(\mathbf{x}) = e^{-\left(x_1^2 + x_2^2\right)} - a = \left[\!\!\left[\begin{bmatrix} e^{-x_1^2} & 1 \end{bmatrix}, \begin{bmatrix} e^{-x_2^2} & -a \end{bmatrix} \right]\!\!\right].$$

On the other hand, the donut-shaped set (Figure 9) required $R = 3$ terms. Again, this is expected, since the donut-shaped set can be modeled through the level sets of the rank-three separable function

$$f(\mathbf{x}) = e^{-\left(x_1^2 + x_2^2\right)} - e^{-b\left(x_1^2 + x_2^2\right)} - a = \left[\!\!\left[\begin{bmatrix} e^{-x_1^2} & e^{-bx_1^2} & 1 \end{bmatrix}, \begin{bmatrix} e^{-x_2^2} & e^{-bx_2^2} & a \end{bmatrix} \right]\!\!\right].$$

The no-entry sign shaped classification set (Figure 10) and the final example (Figure 11) are more complex, but interestingly require only $R = 4$ and $R = 5$ terms, respectively. Similar to the regression problem, overestimation of the rank values do not seem to lead to a performance degradation.

In Figure 12, the performance of the classifier (21) is also compared with existing techniques such as support vector machines (SVMs) using an RBF or polynomial kernel, and a single hidden layer neural network (NN). By considering the classification problem in Figure 11 and taking the median result over 50 experiments (in each of which randomly generated training and test datasets were taken from a uniform distribution), the top two plots in Figure 12 show the obtained fraction incorrect (FiC) for the various methods as a function of the number of training samples. From the plots, it is evident that the classifier (21) with $R = 7$, $M = 16$ (224 parameters) consistently achieves lower training and test errors for this problem, when compared to a NN with 50 nodes (200 parameters) or an RBF-kernel-based SVM. These lower error rates are furthermore obtained with relatively shorter CPU time for completing the training; see bottom plot in Figure 12. Specifically, when compared to SVMs, the required CPU time grows at a milder pace with respect to the size of training dataset.

4.3.2 Case study V

In the next example, we subject the classification problem considered in Figure 11 to noise. We examine how the classifier (21) behaves if the boundary between the two classes is ambiguous or if a small random subset of the samples in both classes is intentionally mislabeled. Figure 13 and Figure 14 display the obtained results. From the figures it can be observed that under these noisy circumstances, the performance of (21) degrades in the same rate as those of established methods of classification.

4.3.3 Case study VI: banknote authentication

The last example that we consider is the task of training a classifier to separate genuine and forged banknotes from each other. The UCI machine learning repository (Dua and Graff, 2017) contains a dataset where a collection of banknote images (forged and genuine) have been postprocessed in order to determine their variance, skewness, kurtosis, and entropy. These four features may be used to train a classifier. In Figure 15, the labeled dataset is displayed in the top four plots. The dataset has been normalized and split into a training set consisting of 1097 samples and a test dataset consisting of 275 samples. The bottom two plots in Figure 15 display the obtained results by using the approximant (21) to fit the data by means of minimizing the objective function (22). The classification error is shown for various configurations of R and M. A uniform knot distribution was chosen for the component functions (7) and the order is set to $N = 4$. From the plots it can be observed that as few as $R = 2$ separable terms are sufficient to get a complete separation between the two classes. In other words, the dataset may live in a higher-dimensional space (4D), but loosely speaking the classification problem is of comparable complexity with respect to the earlier synthetic example in Figure 8, which also only required two rank-one terms.

5 CONCLUSIONS AND FUTURE WORK

We have introduced a supervised learning framework for regression and classification tasks which aims to approximate target functions with a sum of (few) separable terms. Each of the univariate component functions in the separable terms were discretized by a B-spline, resulting in an approximant where the underlying coefficient tensor of the tensor product expansion has a low rank polyadic decomposition parametrization. By taking advantage of the multilinear structure and the sparsity pattern of the compactly supported B-spline basis, a Gauss–Newton algorithm was introduced to train the model efficiently.

We have provided some illustrative examples which reveal the rationale behind the proposed framework and the low-rank mechanisms that are at play. In particular, the presented real-life experiments show that

low-rank structures do appear in practice and using approximants of the type (8) or (21) do have their merit as they allow one to break the curse of dimensionality that is imposed by a direct tensor product B-spline basis (5). Henceforth, further development of this framework of supervised learning is worthwhile and could address more sophisticated, real-life machine learning problems.

There are several directions in which the present work can be extended or further explored. These include particularly matters such as appropriate (adaptive) knot selection heuristics. Furthermore, an extension to multiclass classification using softmax regression has not yet been considered to keep the current exposition concise, but is relatively straightforward to do. Finally, more complicated tensor decomposition architectures can be explored and further investigated. Particularly those of a hierarchical nature sound promising, given that many complex decision processes can be viewed as the consequence of smaller intermediate steps.

AUTHOR CONTRIBUTIONS

NG and NV conjointly developed the theory and Matlab implementation. NG is the main contributor to the numerical experiments and also wrote the first draft of the manuscript. NV reworked the presentation of the algorithms. LDL conceived the idea and supervised the project. All authors contributed to manuscript revision, read, and approved the submitted version.

FUNDING

This work was supported by the *Research Foundation Flanders (FWO)* via projects G086518N, G086318N, and via postdoc grant 12ZM220N; *KU Leuven Internal Funds* via projects C16/15/059 and IDN/19/014; *Fonds de la Recherche Scientifique—FNRS and the Fonds Wetenschappelijk Onderzoek—Vlaanderen* under EOS project No. 30468160 (SeLMA). This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program.

ACKNOWLEDGEMENTS

We would like to thank Muzaffer Ayvaz, Martijn Boussé, Michiel Vandecappelle, and Ethan Epperly for their useful remarks and suggestions.

A EVALUATION OF THE GRADIENT AND GRAMIAN-VECTOR PRODUCTS

The gradient and Gramian-vector products associated with the objective functions (9) and (22) can be evaluated efficiently by exploiting both the multilinear structure resulting from the low-rank approximation of the coefficient tensor, and the compact support property of the B-spline basis functions. In this appendix, we first develop vectorized expressions for auxiliary variables. Next, these expressions are used to compute the gradient and Gramian-vector products. Finally, we discuss the complexity of the algorithm.

A.1 Vectorized expressions for the gradient and Gramian

To facilitate further computations, we first construct a matrix $\mathbf{A}_d$, $d = 1, \ldots, D$, which contains the set of M B-spline basis functions evaluated in $\mathbf{x}_i$, $i = 1, \ldots, I$ as its columns:

$$\mathbf{A}_d \coloneqq \begin{bmatrix} \mathbf{B}_d^{\mathrm{T}}(x_{1,d}) & \cdots & \mathbf{B}_d^{\mathrm{T}}(x_{I,d}) \end{bmatrix} \in \mathbb{R}^{M \times I}.$$

The values η_i and ξ_i are constructed according to (15) and (16), resp., for the regression case, or (23) and (24) for the classification case, and are collected in the vectors $\boldsymbol{\eta} \in \mathbb{R}^I$ and $\boldsymbol{\xi} \in \mathbb{R}^I$, respectively.

The gradient $\mathbf{g}_{r,d} \coloneqq \frac{\partial Q}{\partial \boldsymbol{\gamma}_{r,d}}$ in the regression case, or $\mathbf{g}_{r,d} \coloneqq \frac{\partial L_\alpha}{\partial \boldsymbol{\gamma}_{r,d}}$ in the classification case, can then be expressed in vectorized form as:

$$\begin{aligned}
\mathbf{g}_{r,d} &= \sum_{i=1}^{I} \eta_i v_{r,d;i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}) \\
&= \begin{bmatrix} \mathbf{B}_d^{\mathrm{T}}(x_{1,d}) & \cdots & \mathbf{B}_d^{\mathrm{T}}(x_{I,d}) \end{bmatrix} \left(\begin{bmatrix} v_{r,d;1} \\ \vdots \\ v_{r,d;I} \end{bmatrix} * \begin{bmatrix} \eta_1 \\ \vdots \\ \eta_I \end{bmatrix} \right) \\
&= \begin{bmatrix} \mathbf{B}_d^{\mathrm{T}}(x_{1,d}) & \cdots & \mathbf{B}_d^{\mathrm{T}}(x_{I,d}) \end{bmatrix} \left(\begin{bmatrix} \prod_{k=1,k\neq d}^{D} \mathbf{B}_k(x_{i,k}) \boldsymbol{\gamma}_{r,k} \\ \vdots \\ \prod_{k=1,k\neq d}^{D} \mathbf{B}_k(x_{i,k}) \boldsymbol{\gamma}_{r,k} \end{bmatrix} * \begin{bmatrix} \eta_1 \\ \vdots \\ \eta_I \end{bmatrix} \right) \\
&= \mathbf{A}_d \left(\left(\underset{k=1,k\neq d}{\overset{D}{*}} \mathbf{A}_k^{\mathrm{T}} \boldsymbol{\gamma}_{r,k} \right) * \boldsymbol{\eta} \right).
\end{aligned}$$

Similarly, the required Gramian-vector products, i.e., the products of the Gramian built using blocks $\mathbf{G}_{r,d;\tilde{r},\tilde{d}}$, and the vector built from $\mathbf{z}_{\tilde{r},\tilde{d}}$, can be expressed as

$$\begin{aligned}
\mathbf{w}_{r,d} &\coloneqq \sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \mathbf{G}_{r,d;\tilde{r},\tilde{d}} \mathbf{z}_{\tilde{r},\tilde{d}} \\
&= \sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \left(\sum_{i=1}^{I} \xi_i \left(v_{r,d;i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}) \right) \left(v_{\tilde{r},\tilde{d};i} \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \right) \right) \mathbf{z}_{\tilde{r},\tilde{d}} \\
&= \sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \sum_{i=1}^{I} \left(v_{r,d;i} \mathbf{B}_d^{\mathrm{T}}(x_{i,d}) \right) \xi_i v_{\tilde{r},\tilde{d};i} \left(\mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \mathbf{z}_{\tilde{r},\tilde{d}} \right) \\
&= \sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \begin{bmatrix} v_{r,d;1} \mathbf{B}_d^{\mathrm{T}}(x_{1,d}) & \cdots & v_{r,d;I} \mathbf{B}_d^{\mathrm{T}}(x_{I,d}) \end{bmatrix} \left(\begin{bmatrix} \xi_1 \\ \vdots \\ \xi_I \end{bmatrix} * \begin{bmatrix} v_{r,d;1} \\ \vdots \\ v_{r,d;I} \end{bmatrix} * \begin{bmatrix} \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \mathbf{z}_{\tilde{r},\tilde{d}} \\ \vdots \\ \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \mathbf{z}_{\tilde{r},\tilde{d}} \end{bmatrix} \right) \\
&= \begin{bmatrix} \mathbf{B}_d^{\mathrm{T}}(x_{1,d}) & \cdots & \mathbf{B}_d^{\mathrm{T}}(x_{I,d}) \end{bmatrix} \left(\begin{bmatrix} v_{r,d;1} \\ \vdots \\ v_{r,d;I} \end{bmatrix} * \left(\sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_I \end{bmatrix} * \begin{bmatrix} v_{r,d;1} \\ \vdots \\ v_{r,d;I} \end{bmatrix} * \begin{bmatrix} \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \\ \vdots \\ \mathbf{B}_{\tilde{d}}(x_{i,\tilde{d}}) \end{bmatrix} \mathbf{z}_{\tilde{r},\tilde{d}} \right) \right) \\
&= \mathbf{A}_d \left(\left(\underset{k=1,k\neq d}{\overset{D}{*}} \mathbf{A}_k^{\mathrm{T}} \boldsymbol{\gamma}_{r,k} \right) * \boldsymbol{\xi} * \left(\sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \left(\underset{k=1,k\neq d}{\overset{D}{*}} \mathbf{A}_k^{\mathrm{T}} \boldsymbol{\gamma}_{\tilde{r},k} \right) * \mathbf{A}_{\tilde{d}}^{\mathrm{T}} \mathbf{z}_{\tilde{r},\tilde{d}} \right) \right).
\end{aligned}$$

By introducing the variables

$$\mathbf{Q}_d \coloneqq \mathbf{A}_d^{\mathrm{T}} \boldsymbol{\Gamma}_d \tag{26}$$

and

$$\mathbf{X}_d \coloneqq \underset{k=1,k\neq d}{\overset{D}{*}} \mathbf{Q}_d \tag{27}$$

we can simplify these expressions to

$$\mathrm{g}_{r,d} = \mathbf{A}_d \left(\mathbf{X}_d[:, r] * \boldsymbol{\eta} \right), \tag{28}$$

$$\mathbf{w}_{r,d} = \mathbf{A}_d \left(\mathbf{X}_d[:, r] * \boldsymbol{\xi} * \left(\sum_{\tilde{d}=1}^{D} \sum_{\tilde{r}=1}^{R} \mathbf{X}_{\tilde{d}}[:, \tilde{r}] * \left(\mathbf{A}_{\tilde{d}}^{\mathrm{T}} \mathbf{z}_{\tilde{r},\tilde{d}} \right) \right) \right). \tag{29}$$

For the quadratic objective function (9) used in regression, we have, as a consequence of (15) and (16), that

$$\boldsymbol{\eta} = \hat{\mathbf{y}} - \mathbf{y}, \qquad \boldsymbol{\xi} = \mathbf{1}_N, \tag{30}$$

where

$$\hat{\mathbf{y}} \coloneqq \begin{bmatrix} \hat{f}(\mathbf{x}_i; \boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D) \\ \vdots \\ \hat{f}(\mathbf{x}_i; \boldsymbol{\Gamma}_1, \ldots, \boldsymbol{\Gamma}_D) \end{bmatrix} = \left(\overset{D}{\underset{d=1}{*}} \mathbf{X}_d \right) \mathbf{1}_R, \qquad \text{and} \qquad \mathbf{y} \coloneqq \begin{bmatrix} y_1 \\ \vdots \\ y_I \end{bmatrix}. \tag{31}$$

In the case of the logarithmic objective function (22), we have the expressions

$$\boldsymbol{\eta} = \alpha \left(\sigma_\alpha \left(\hat{\mathbf{y}} \right) - \mathbf{y} \right), \qquad \text{and} \qquad \boldsymbol{\xi} = \alpha^2 \sigma_\alpha \left(\hat{\mathbf{y}} \right) * \left(1 - \sigma_\alpha \left(\hat{\mathbf{y}} \right) \right), \tag{32}$$

as a consequence of (23) and (24).

A.2 Summary of computational steps

In each GN or generalized GN iteration, the gradient and $\mathrm{it}_{\mathrm{CG}}$ Gramian-vector products are required, in which $\mathrm{it}_{\mathrm{CG}}$ is the number of CG iterations to solve the linear systems (18) or (25), respectively. Let us define the shorthand notations:

$$\mathbf{H}_d \coloneqq \begin{bmatrix} \mathrm{g}_{1,d} & \cdots & \mathrm{g}_{R,d} \end{bmatrix}, \qquad \mathbf{Z}_d \coloneqq \begin{bmatrix} \mathbf{z}_{1,d} & \cdots & \mathbf{z}_{R,d} \end{bmatrix}, \qquad \mathbf{W}_d \coloneqq \begin{bmatrix} \mathbf{w}_{1,d} & \cdots & \mathbf{w}_{R,d} \end{bmatrix}.$$

The steps for evaluating the gradient and Gramian-vector product can then be summarized as:

Step 1. Precompute $\mathbf{Q}_d \in \mathbb{R}^{I \times R}$ as defined in (26).
Step 2. Precompute $\mathbf{X}_d \in \mathbb{R}^{I \times R}$ defined in (27).
Step 3. Evaluate $\boldsymbol{\eta} \in \mathbb{R}^I$ and $\boldsymbol{\xi} \in \mathbb{R}^I$ according to (30) or (32), depending on which objective function is being considered. The vector $\hat{\mathbf{y}}$ defined in (31) is computed as $\hat{\mathbf{y}} = \left(\overset{D}{\underset{d=1}{*}} \mathbf{X}_d \right) \mathbf{1}_R$.
Step 4. Evaluate $\mathbf{H}_d \in \mathbb{R}^{(M_d+1) \times R}$ and $\mathbf{W}_d \in \mathbb{R}^{(M_d+1) \times R}$ using the expressions (28) and (29), respectively.

Algorithm 1 provides a pseudocode on how these formulas can be evaluated in practice.

A.3 Complexity

As the number of iterations of the (generalized) GN algorithm is highly variable and depends, among others, on the initialization, the per-iteration complexity is derived here. Every iteration, the gradient is computed once, the function value $\mathrm{it}_{\mathrm{TR}}$ times (usually once), and the Gramian-vector is computed $\mathrm{it}_{\mathrm{CG}}$ times for different vectors $\mathbf{z}_{r,d}$ but the same variables $\boldsymbol{\Gamma}_d$, $d = 1, \ldots, d$, $r = 1, \ldots, R$. Typical values for $\mathrm{it}_{\mathrm{CG}}$ are 15, 25, or 50.

Algorithm 1 Gradient and Gramian-vector product computations.

function GRAMIANPRODUCT($\{\mathbf{Z}_d\}_{d=1}^D, \{\mathbf{A}_d\}_{d=1}^D, \{\mathbf{X}_d\}_{d=1}^D, \boldsymbol{\xi}$)
  $\mathbf{v} \leftarrow 0$
  for $d = 1 : D$ **do**
    for $r = 1 : R$ **do**
      $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{X}_d[:, r] * \left(\mathbf{A}_d^\mathsf{T} \mathbf{Z}_d[:, r]\right)$
  $\mathbf{v} \leftarrow \boldsymbol{\xi} * \mathbf{v}$
  $\{\mathbf{W}_d\}_{d=1}^D \leftarrow$ GRADIENT($\mathbf{v}, \{\mathbf{A}_d\}_{d=1}^D, \{\mathbf{X}_d\}_{d=1}^D$)
  return $\{\mathbf{W}_d\}_{d=1}^D$
function GRADIENT($\boldsymbol{\eta}, \{\mathbf{A}_d\}_{d=1}^D, \{\mathbf{X}_d\}_{d=1}^D$)
  for $d = 1 : D$ **do**
    for $r = 1 : R$ **do**
      $\mathbf{H}_d[:, r] \leftarrow \mathbf{A}_d \left(\mathbf{X}_d[:, r] * \boldsymbol{\eta}\right)$
  return $\{\mathbf{H}_d\}_{d=1}^D$

In the derivation of the complexity, we take $N = \max_d N_d$, which typically satisfies $N \leq 4$. The number of spline basis function M_d, which grows linearly with the number of knots, is determined by $M = \max_d M_d$. The number of samples is I and the rank of the coefficient tensor is chosen to be R.

For general basis functions, precomputing $\mathbf{Q}_d$ and $\mathbf{X}_d$ in Steps 1 and 2 costs $\mathcal{O}(DIMR)$ and $\mathcal{O}\left(D^2IMR\right)$ flop, respectively. The computation of the gradient (Step 3) involves the (transformed) residual $\boldsymbol{\eta}$, which is also used in the computation of the function value, and takes $\mathcal{O}(DIR)$ flop. Finally, to compute the (generalized) GN step iteratively, the CG algorithm performs it_{CG} Gramian-vector products (Step 4), each of which takes $\mathcal{O}(DIMR)$ flop. Hence, overall the computational cost is dominated by the Gramian-vector products: $\mathcal{O}(\text{it}_{\text{CG}} DIMR)$ flop per GN iteration.

In this paper, we propose to not use general basis functions, but we use B-splines instead. This has a computational advantage as the matrices $\mathbf{A}_d$ and $\mathbf{X}_d$ are now sparse, having approximately N nonzeros per sample. By exploiting the sparsity, the complexity can effectively be reduced to $\mathcal{O}(\text{it}_{\text{CG}} DINR)$ flop per GN iteration. Hence the dependence on the potentially large number of basis functions M is removed and replaced by the low order of the B-splines N.

REFERENCES

Beylkin, G., Garcke, J., and Mohlenkamp, M. J. (2009). Multivariate regression and machine learning with sums of separable functions. *SIAM Journal on Scientific Computing* 31, 1840–1857

Beylkin, G. and Mohlenkamp, M. J. (2005). Algorithms for numerical analysis in high dimensions. *SIAM Journal on Scientific Computing* 26, 2133–2159

Boussé, M., Vervliet, N., Domanov, I., Debals, O., and De Lathauwer, L. (2018). Linear systems with a canonical polyadic decomposition constrained solution: Algorithms and applications. *Numerical Linear Algebra with Applications* 25, e2190

Boussé, M., Debals, O., and De Lathauwer, L. (2017). Tensor-based large-scale blind system identification using segmentation. *IEEE Trans. Signal Process.* 65, 5770–5784

Brooks, T., Pope, D., and Marcolini, A. (1989). *Airfoil self-noise and prediction*. Tech. Rep. RP-1218, NASA

Chandrasekaran, S., Jayaraman, K. R., and Mhaskar, H. N. (2013). Minimum sobolev norm interpolation with trigonometric polynomials on the torus. *Journal of Computational Physics* 249, 96–112

Chen, Z., Batselier, K., Suykens, J. A., and Wong, N. (2017). Parallelized tensor train learning of polynomial classifiers. *IEEE transactions on neural networks and learning systems* 29, 4621–4632

Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., et al. (2015). Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE signal processing magazine* 32, 145–163

De Boor, C. (1978). *A practical guide to splines*, vol. 27 (springer-verlag New York)

Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *Advances in neural information processing systems*. 666–674

DeVore, R. A., Howard, R., and Micchelli, C. (1989). Optimal nonlinear approximation. *Manuscripta mathematica* 63, 469–478

[Dataset] Dua, D. and Graff, C. (2017). UCI machine learning repository

Garcke, J. (2010). Classification with sums of separable functions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (Springer), 458–473

Gens, R. and Domingos, P. (2012). Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*. 3239–3247

Gens, R. and Pedro, D. (2013). Learning the structure of sum-product networks. In *International conference on machine learning*. 873–880

Grasedyck, L., Kressner, D., and Tobler, C. (2013). A literature survey of low-rank tensor approximation techniques. *GAMM-Mitteilungen* 36, 53–78

Grelier, E., Nouy, A., and Chevreuil, M. (2018). Learning with tree-based tensor formats. *arXiv preprint arXiv:1811.04455*

Hackbusch, W. (2012). *Tensor spaces and numerical tensor calculus*, vol. 42 (Springer)

Harshman, R. A. (1970). Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers Phonetics* 16, 1–84

Hashemizadeh, M., Liu, M., Miller, J., and Rabusseau, G. (2020). Adaptive tensor learning with tensor networks. *arXiv preprint arXiv:2008.05437*

Hendrikx, S., Boussé, M., Vervliet, N., and De Lathauwer, L. (2019). Algebraic and optimization based algorithms for multivariate regression using symmetric tensor decomposition. In *Proceedings of the 2019 IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP 2019)*. 475–479

Hou, M. and Chaib-Draa, B. (2015). Hierarchical Tucker tensor regression: application to brain imaging data analysis. In *2015 IEEE International Conference on Image Processing (ICIP)* (IEEE), 1344–1348

Khoromskij, B. N. (2018). *Tensor numerical methods in scientific computing*, vol. 19 (Walter de Gruyter GmbH & Co KG)

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review* 51, 455–500

Lai, M.-J. and Schumaker, L. L. (2007). *Spline functions on triangulations*. 110 (Cambridge University Press)

Liu, Y., Pan, J., and Ng, M. (2019). Tucker decomposition network: Expressive power and comparison. *arXiv preprint arXiv:1905.09635*

Mhaskar, H. N. and Pai, D. V. (2000). *Fundamentals of approximation theory* (CRC Press)

Mhaskar, H. N. and Poggio, T. (2016). Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications* 14, 829–848

Nocedal, J. and Wright, S. (2006). *Numerical optimization* (Springer Science & Business Media)

Novikov, A., Trofimov, M., and Oseledets, I. (2016). Exponential machines. *arXiv preprint arXiv:1605.03795*

Oseledets, I. V. (2013). Constructive representation of functions in low-rank tensor formats. *Constructive Approximation* 37, 1–18

Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)* (IEEE), 689–690

Rajagopal, A. (2019). *High-Dimensional Polynomial Approximation with Applications in Imaging and Recognition*. Ph.D. thesis, University of California, Santa Barbara

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks* 61, 85–117

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation* 14, 1723–1738

Sidiropoulos, N. D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E. E., and Faloutsos, C. (2017). Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing* 65, 3551–3582

Sorber, L., Van Barel, M., and De Lathauwer, L. (2013). Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization. *SIAM J. Optim.* 23, 695–720

Trefethen, L. N. (2019). *Approximation theory and approximation practice*, vol. 164 (SIAM)

Unser, M. (2019). A representer theorem for deep neural networks. *Journal of Machine Learning Research* 20, 1–30

Vandecapelle, M., Vervliet, N., and De Lathauwer, L. (2020a). Inexact generalized Gauss–Newton for scaling the canonical polyadic decomposition with non-least-squares cost functions. *IEEE Journal of Selected Topics in Signal Processing* , 1–1

Vandecapelle, M., Vervliet, N., and De Lathauwer, L. (2020b). A second-order method for fitting the canonical polyadic decomposition with non-least-squares cost. *IEEE Transactions on Signal Processing* 68, 4454–4465

Vervliet, N. and De Lathauwer, L. (2019). Numerical optimization-based algorithms for data fusion. In *Data Handling in Science and Technology* (Elsevier), vol. 31. 81–128

Vervliet, N., Debals, O., and De Lathauwer, L. (2017). Canonical polyadic decomposition of incomplete tensors with linearly constrained factors. *Technical Report 16–172, ESAT-STADIUS, KU Leuven, Leuven, Belgium*

Vervliet, N., Debals, O., Sorber, L., Van Barel, M., and De Lathauwer, L. (2016). Tensorlab 3.0. Available online at `https://www.tensorlab.net`

Yarotsky, D. (2017). Error bounds for approximations with deep ReLU networks. *Neural Networks* 94, 103–114

Zhou, S. and Shen, X. (2001). Spatially adaptive regression splines and accurate knot selection schemes. *Journal of the American Statistical Association* 96, 247–259

FIGURE CAPTIONS

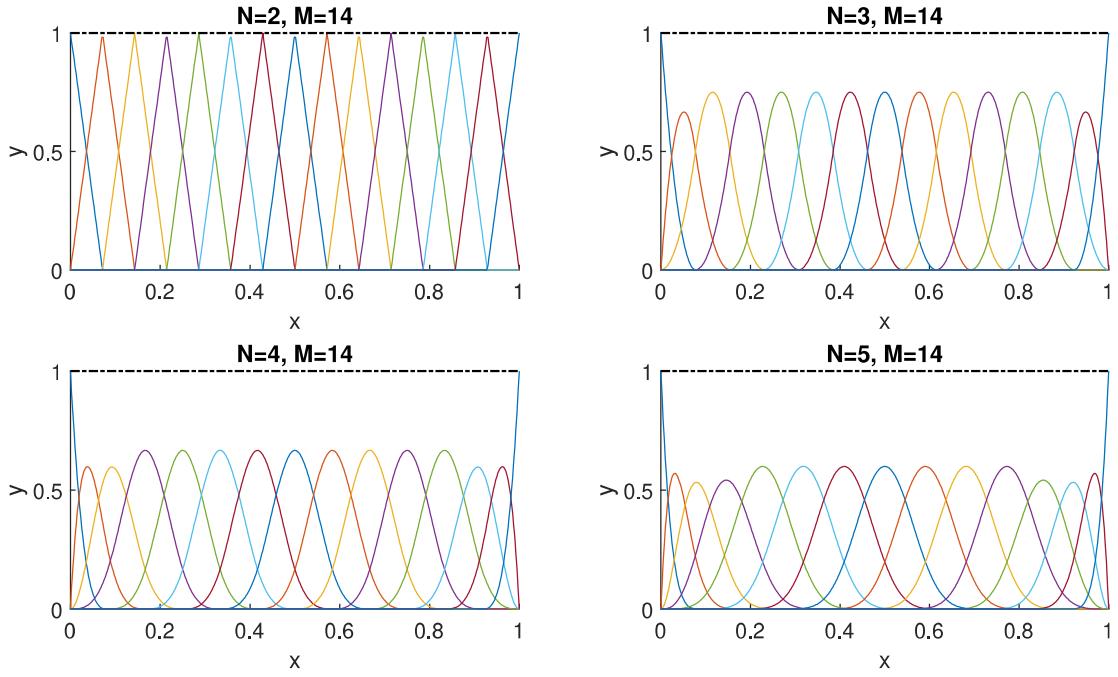


Figure 1. The B-spline basis terms (2) are compactly supported. Shown are the basis terms for orders $N = 2, 3, 4, 5$. The knots $\mathscr{T} = \{t_i\}_{i=0}^{N+M}$ are distributed uniformly: $t_i = 0$ for $i = 0, \ldots, N-1$, $t_i = (i - N + 1)/(M - N + 2)$ for $i = N, \ldots, M$, and $t_i = 1$ for $i = M+1, \ldots, N$. Shown also in black is the sum of all the B-spline basis terms, i.e., $\sum_{m=0}^{M} B_{m,N}(x) = 1$.

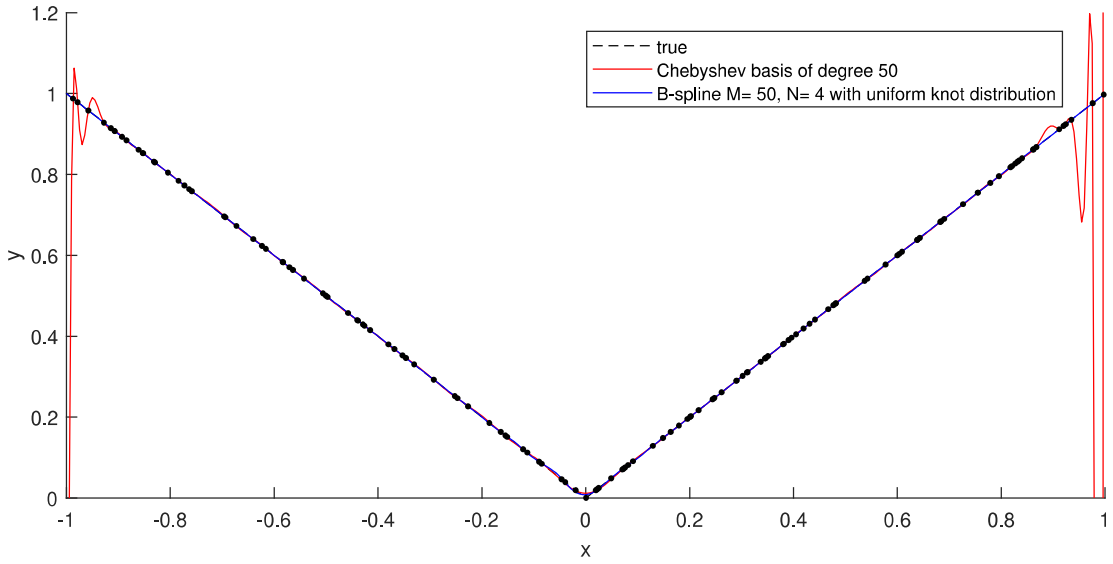


Figure 2. If an ordinary least-squares objective function is used to construct an approximation, high-degree polynomials become more susceptible to the Runge's phenomenon, resulting in wild oscillations near the boundary of the approximation interval. By keeping the order of the B-splines low, the Runge's effects can be circumvented. Shown in the plot is a least-squares fit of the function $f(x) = |x|$ on the interval $[-1, 1]$. We used 160 random function samples (using a uniform distribution) to perform the regression. The samples are plotted in black dots.

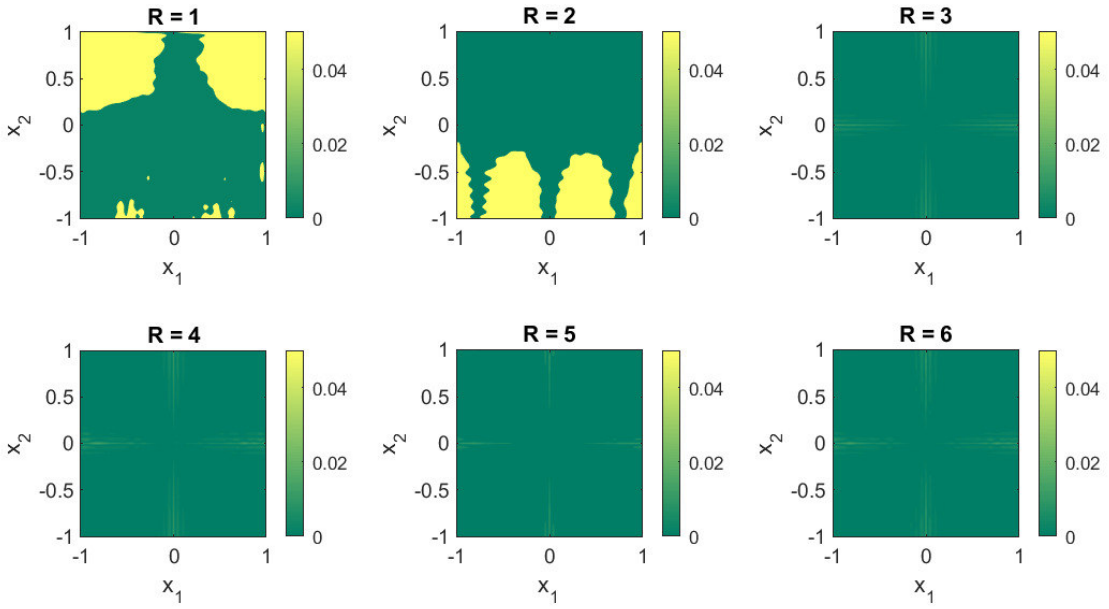


Figure 3. A $R = 3$ separable expansion is necessary to obtain a fit that is capable of capturing the global features of the function (19). We used 8000 random function samples for our training set. The fits were obtained with a B-spline basis comprising of a uniformly distributed knot set and $M = 35$, $N = 4$. The plots depict the absolute error between the true function (19) and the obtained approximation using (8).

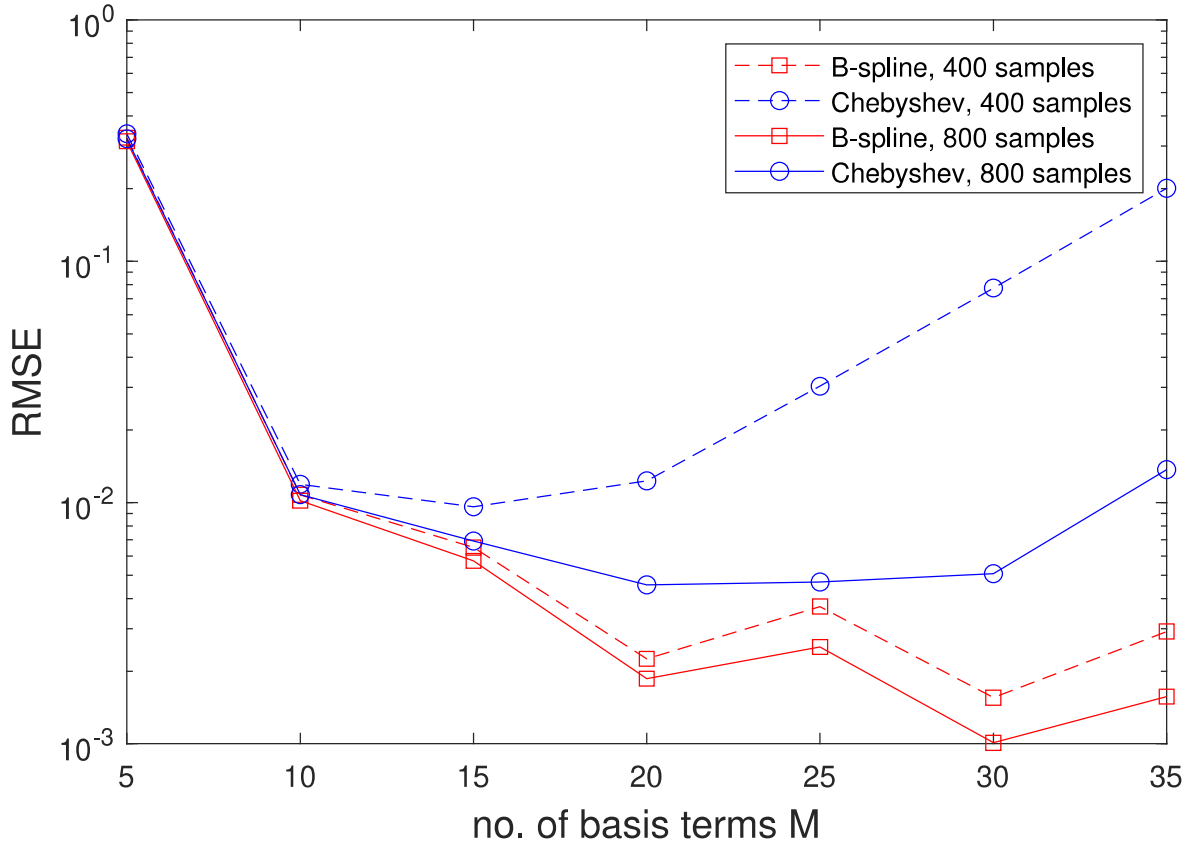


Figure 4. Runge's effects, as illustrated graphically in Figure 2, resurface again when the function (19) is approximated with a sum of separable terms (8) where the B-spline basis is replaced with a Chebyshev basis. This occurs due to the non-analytic term $|x_1||x_2|$ which makes (19) non-differentiable on the lines $x_1 = 0$ and $x_2 = 0$. If a fit is generated through minimizing the least-squares objective function (9), a Chebyshev basis will, unlike for B-splines, start exhibiting degrading performance when the number of basis terms becomes large with respect to the size of the dataset. This can be observed in the plot above, which shows the median RMSE obtained over 50 experiment, in each of which a different random set of function samples (derived from a uniform distribution) were used. For the experiments, the B-spline basis comprised of a uniformly distributed knot set. The oscillations in RMSE curve for the B-spline basis can be attributed to the fact that the knots will not always lie exactly on the non-differentiable region of the function.

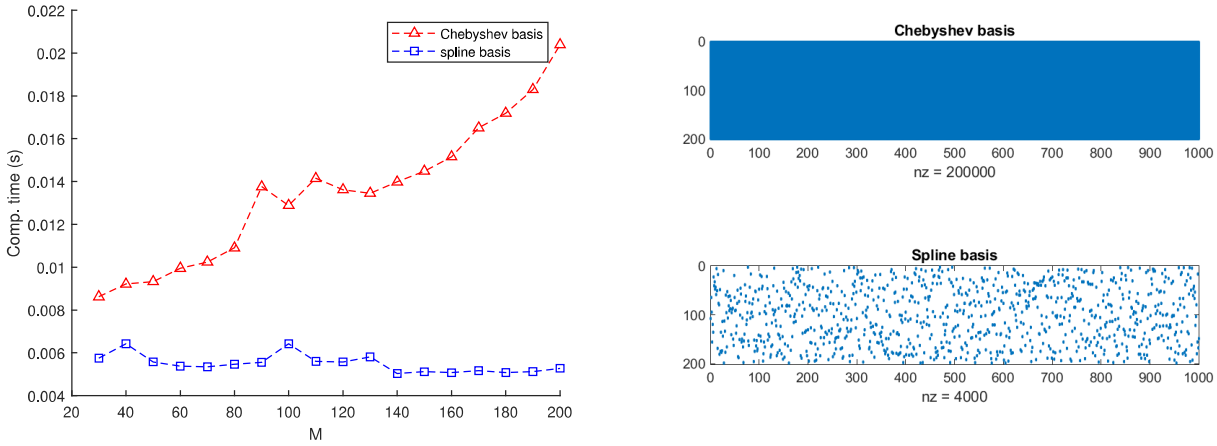


Figure 5. The exploitation of the sparsity in the B-spline basis matrix $A_d := [\boldsymbol{B}_d^{\mathrm{T}}(x_{1,d}) \quad \cdots \quad \boldsymbol{B}_d^{\mathrm{T}}(x_{I,d})]$ significantly accelerates the GN step. The left plot shows the average required computation time to pass through one cycle of the GN algorithm for a B-spline basis of order $N = 4$ and a Chebyshev basis. We used 1000 random data points to fit the function (19). The rank is set to $R = 3$. The right plots provide a glimpse of the sparsity structure of the matrix A_d for a specific d. This sparsity is exploited in the implementation. In the case of the Chebyshev basis, there is no sparsity pattern to be exploited.

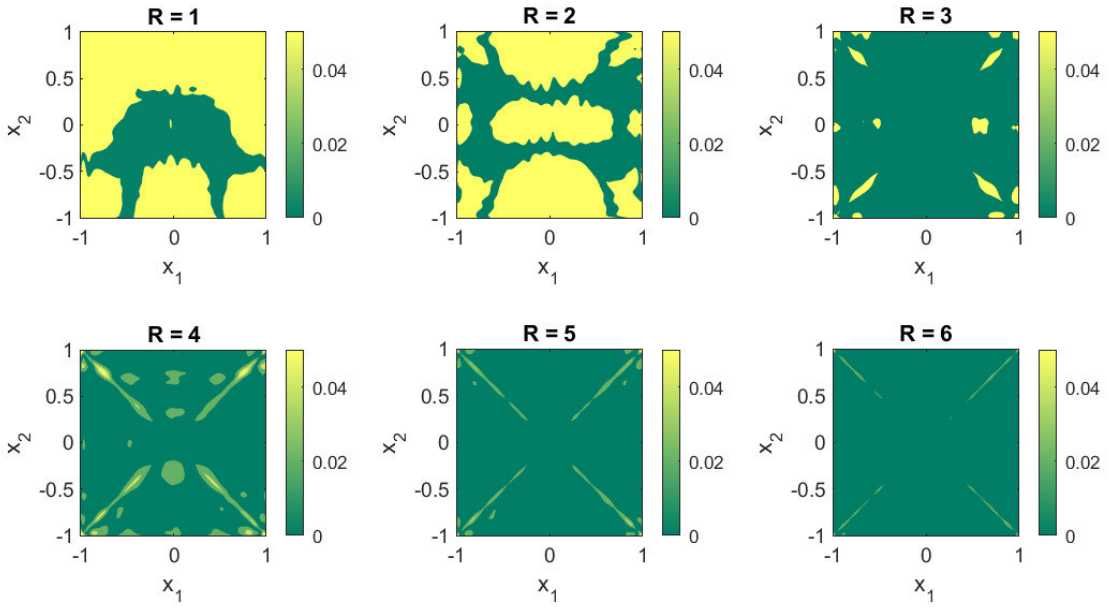


Figure 6. Replacing the term $|x_1||x_2|$ in (19) with $\left|\frac{\sqrt{2}}{2}x_1 - \frac{\sqrt{2}}{2}x_2\right| \left|\frac{\sqrt{2}}{2}x_1 + \frac{\sqrt{2}}{2}x_2\right|$ destroys the rank-three structure of (19). Yet, an $R = 6$ separable expansion is sufficient to get a an approximation that is capable of capturing the global features of (20). We used 8000 random function samples for our training set. The fits were obtained with a B-spline basis comprising of a uniformly distributed knot set and $M = 35$, $N = 4$. The plots show the absolute error between the true function (19) and the obtained approximation using (8).

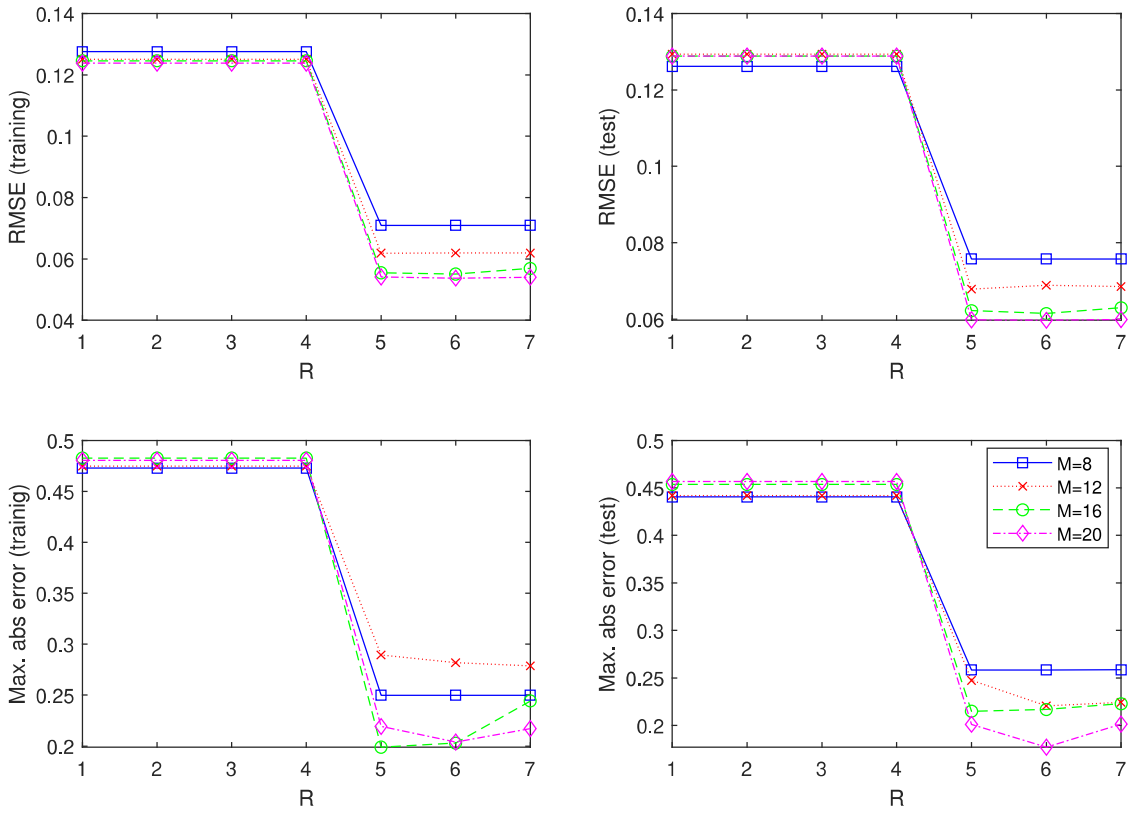


Figure 7. The NASA dataset of the UCI machine learning repository is well modeled by a $R = 5$ separable function, where we see a sudden drop in the training and test error. This indicates the benefits of using (8) over a direct tensor product expansion (5), where no low-rank structures are exploited. For generating the results, the knot distribution is kept uniform for all experiments.

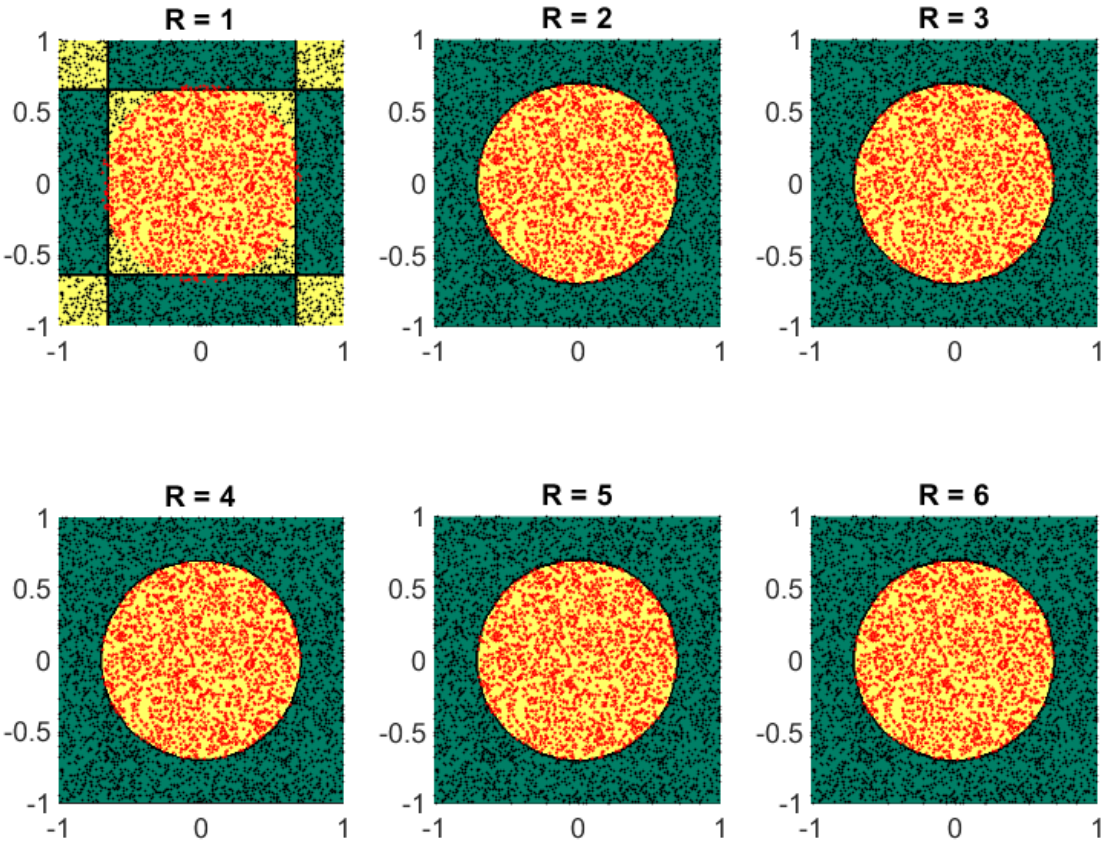


Figure 8. For a simple circular shaped classification set, two separable terms suffice for a good classification. Shown are the obtained results on a synthetic dataset by using the logistic loss function (22). The recovered classification sets are marked by the yellow and green contours, whereas the labeled data used for training are marked by red and black dots.

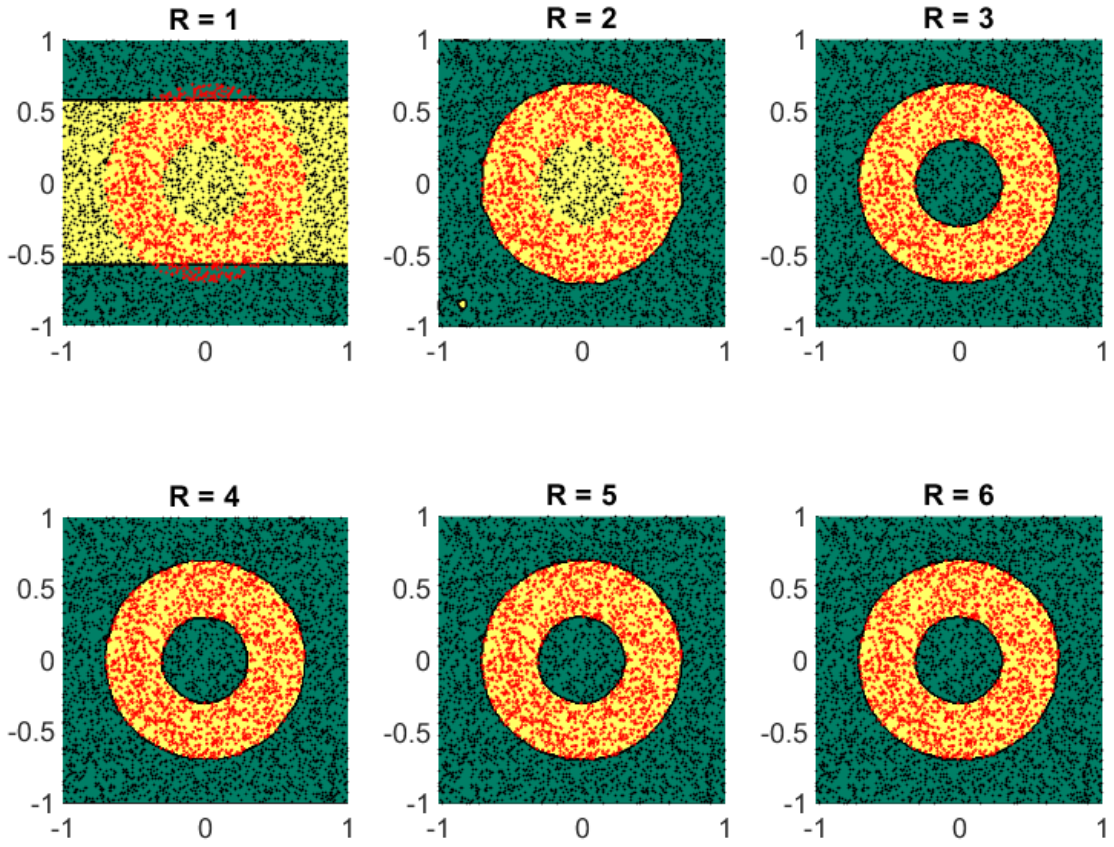


Figure 9. For a donut-shaped classification set, three separable terms are enough for a good classification. Shown are the obtained results on a synthetic dataset by using the logistic loss function (22). The recovered classification sets are marked by the yellow and green contours, whereas the labeled data used for training are marked by red and black dots.

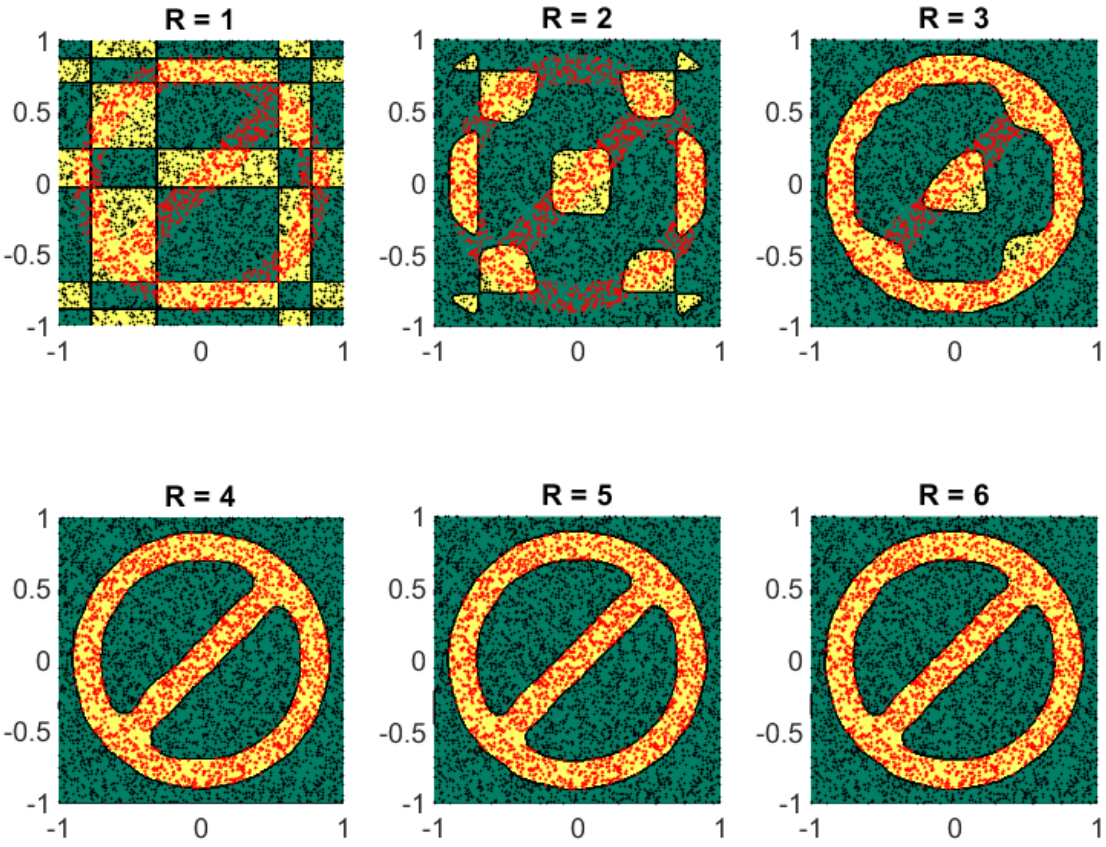


Figure 10. For a classification set shaped in the form of a no-entry sign, at least four separable terms are required for a good classification. Shown are the obtained results on a synthetic dataset by using the logistic loss function (22). The recovered classification sets are marked by the yellow and green contours, whereas the labeled data used for training are marked by red and black dots.

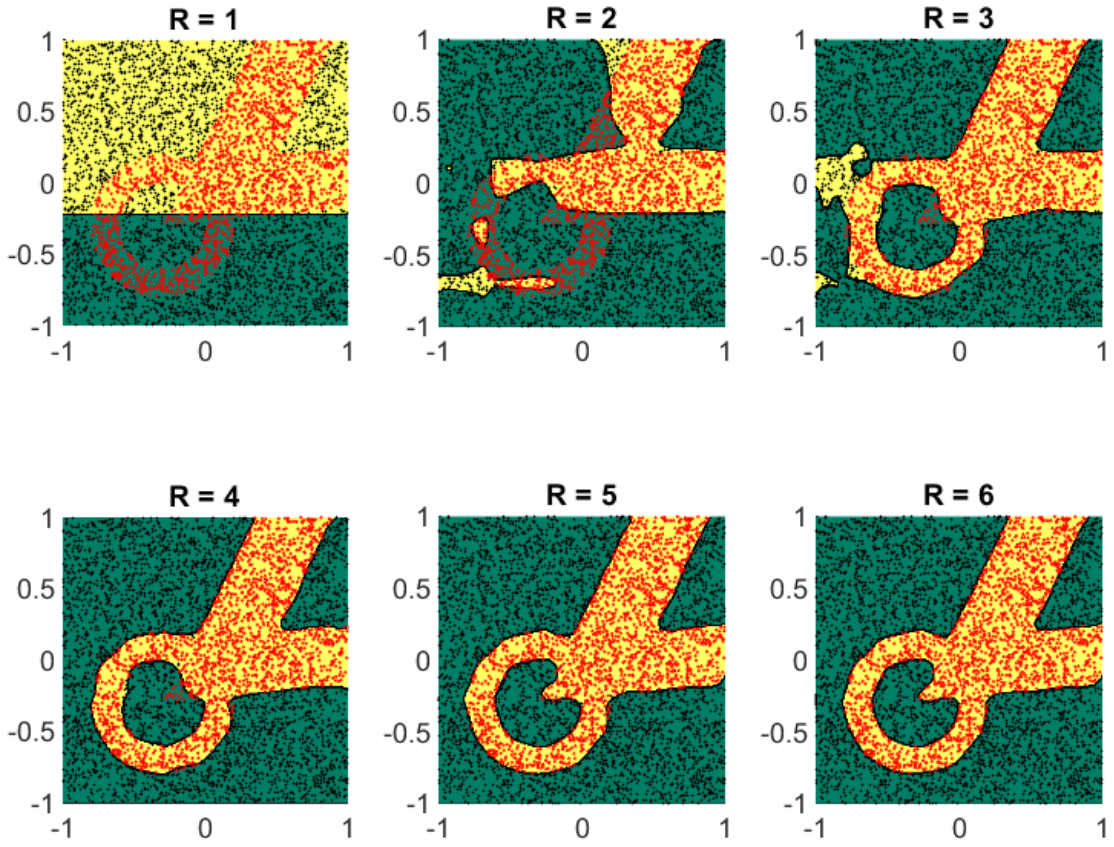


Figure 11. For this particular classification set, at least five separable terms are required for a decent classification. Shown are the obtained results on a synthetic dataset by using the logistic loss function (22). The recovered classification sets are marked by the yellow and green contours, whereas the labeled data used for training are marked by red and black dots.

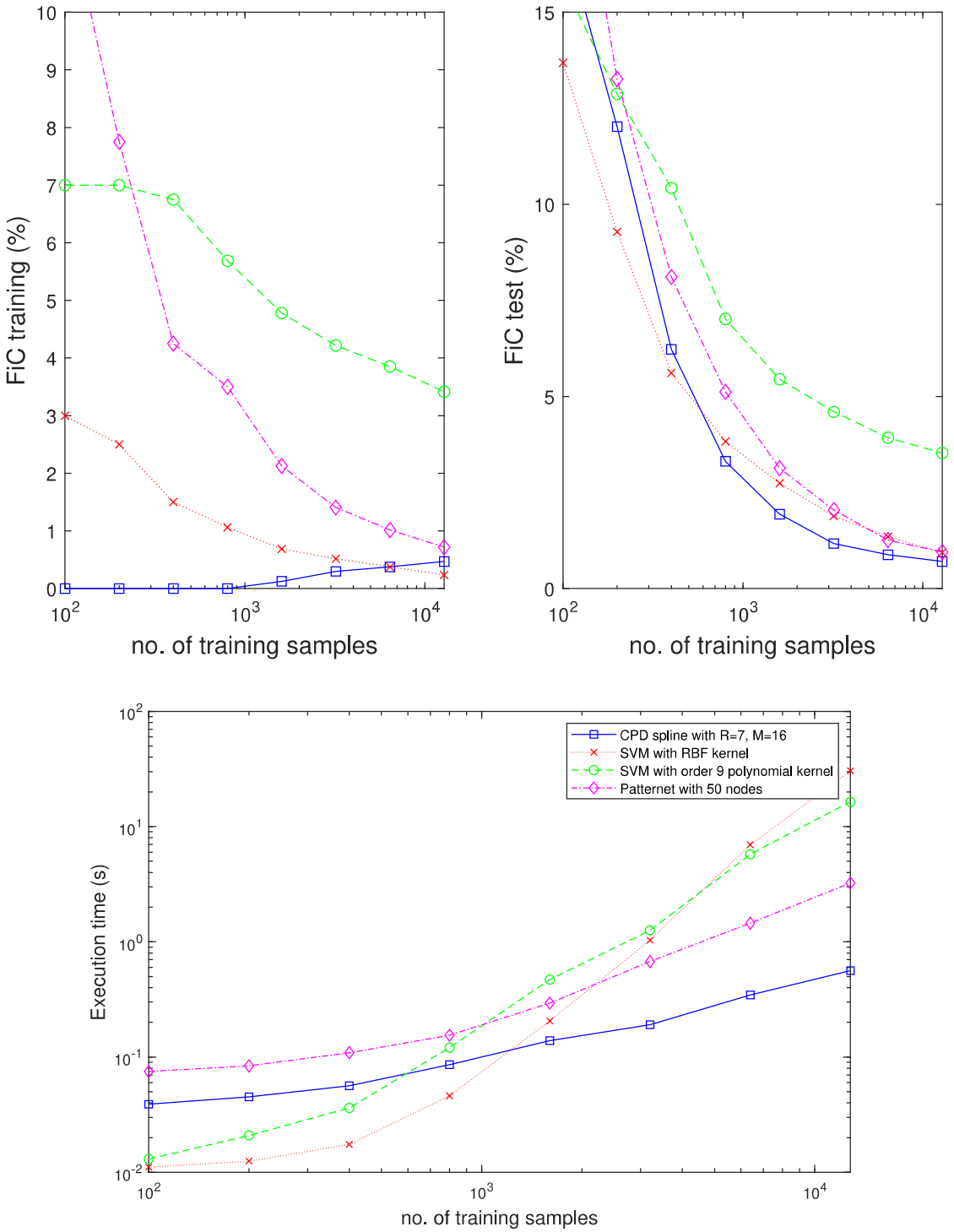


Figure 12. The classifier (21) is competitive with well-established techniques in terms of classification accuracy and computation time. Shown is the median result obtained from 50 randomly generated training and test datasets for the classification problem in Figure 11. Specifically, for the SVMs and NN, the built-in Matlab routines `fitcsvm` and `patternnet` were used to develop the results. Whereas the top two plots display the obtained fraction incorrect (FiC) for the training and test dataset, respectively, the bottom plot shows the required CPU time to complete the training. The test dataset size always equals 5000 samples.

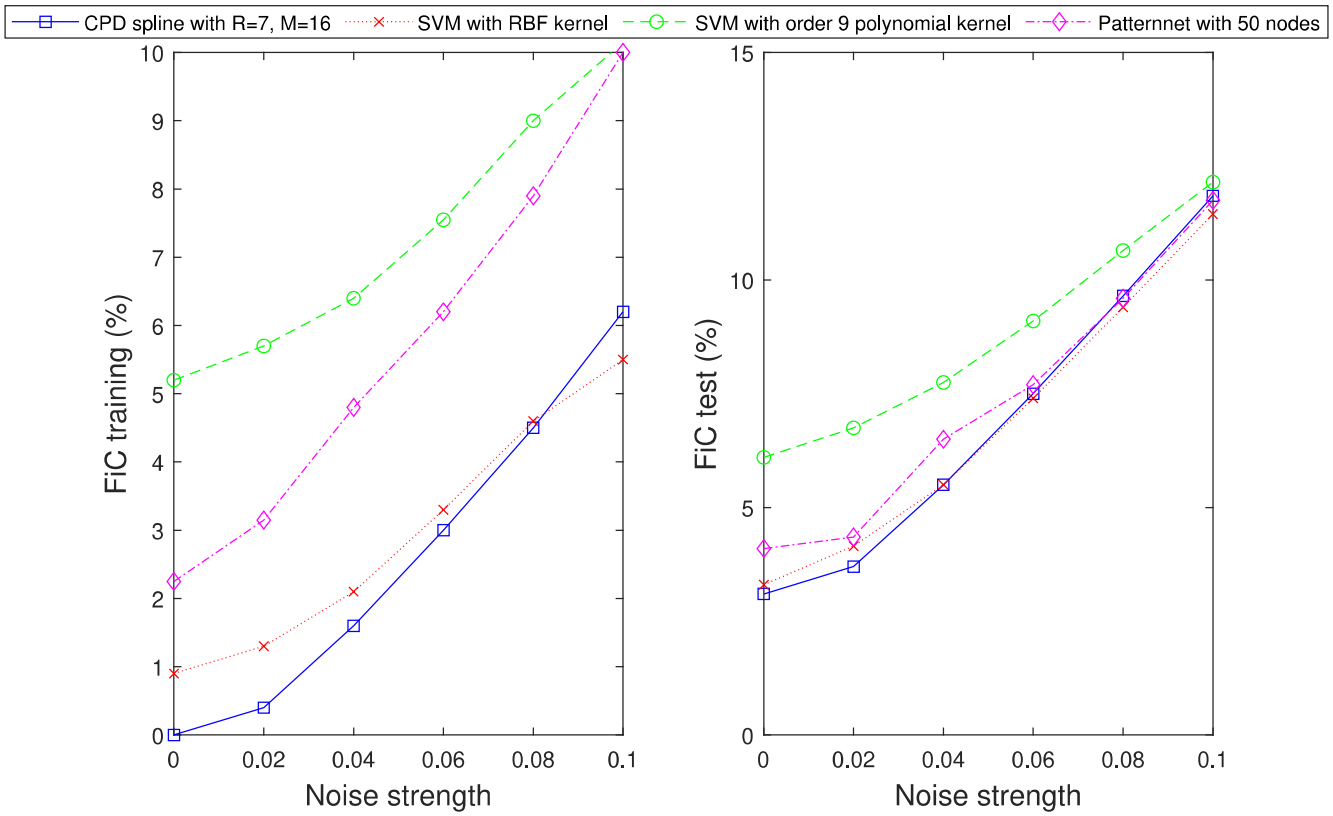


Figure 13. Similar to well-established techniques for classification, the classifier (21) is capable of separating classification sets with ambiguous boundaries. Shown are the obtained results for the classification problem considered in Figure 11, however, now the data samples are randomly perturbed by a vector of specific length (indicated by noise strength). The perturbation generates an ambiguous boundary between the two classes. The number of samples for the training and test dataset is set to 1000.

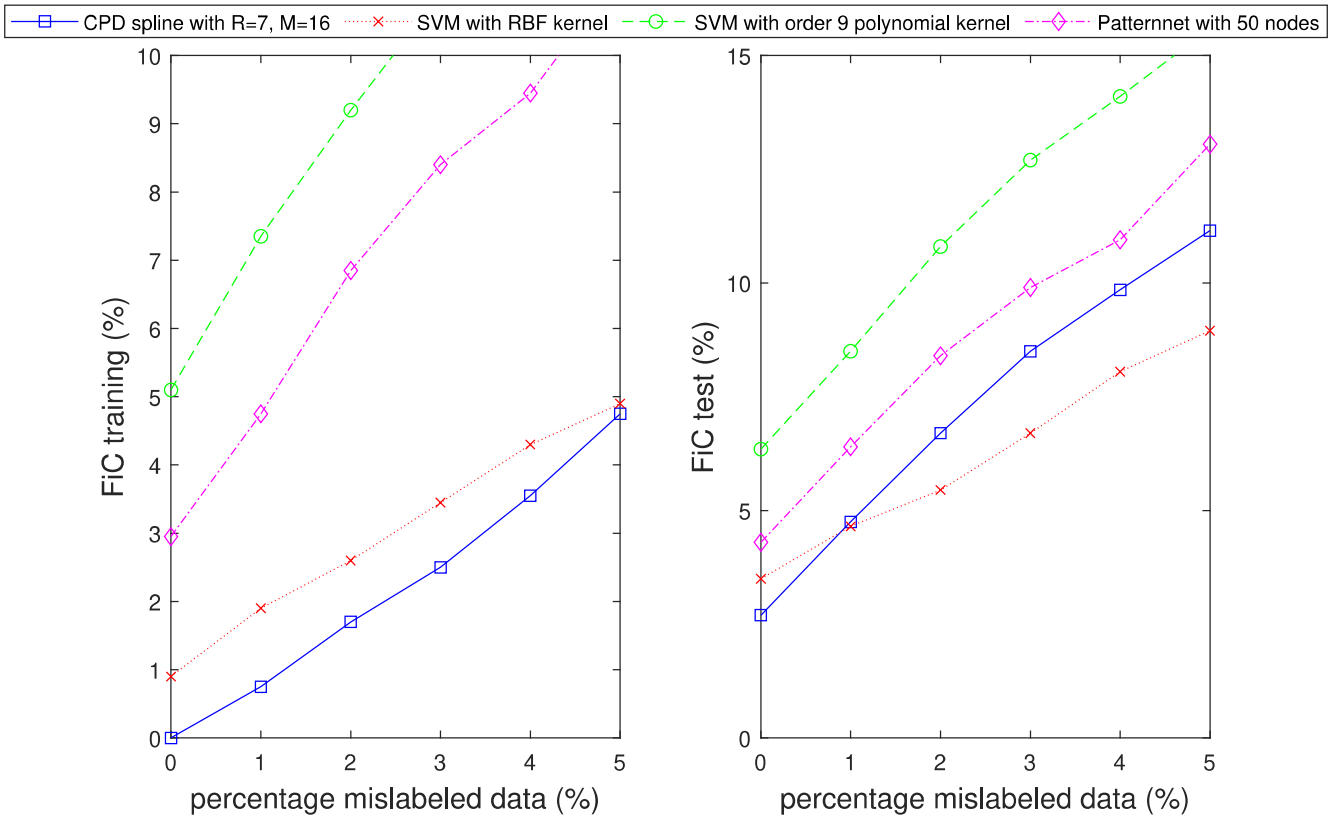


Figure 14. Similar to well-established techniques for classification, the classifier (21) is robust against mislabeled data. Shown are the obtained results for the classification problem considered in Figure 11, however, now in a situation where a particular percentage of data is intentionally mislabeled. The number of samples for the training and test dataset is set to 1000.

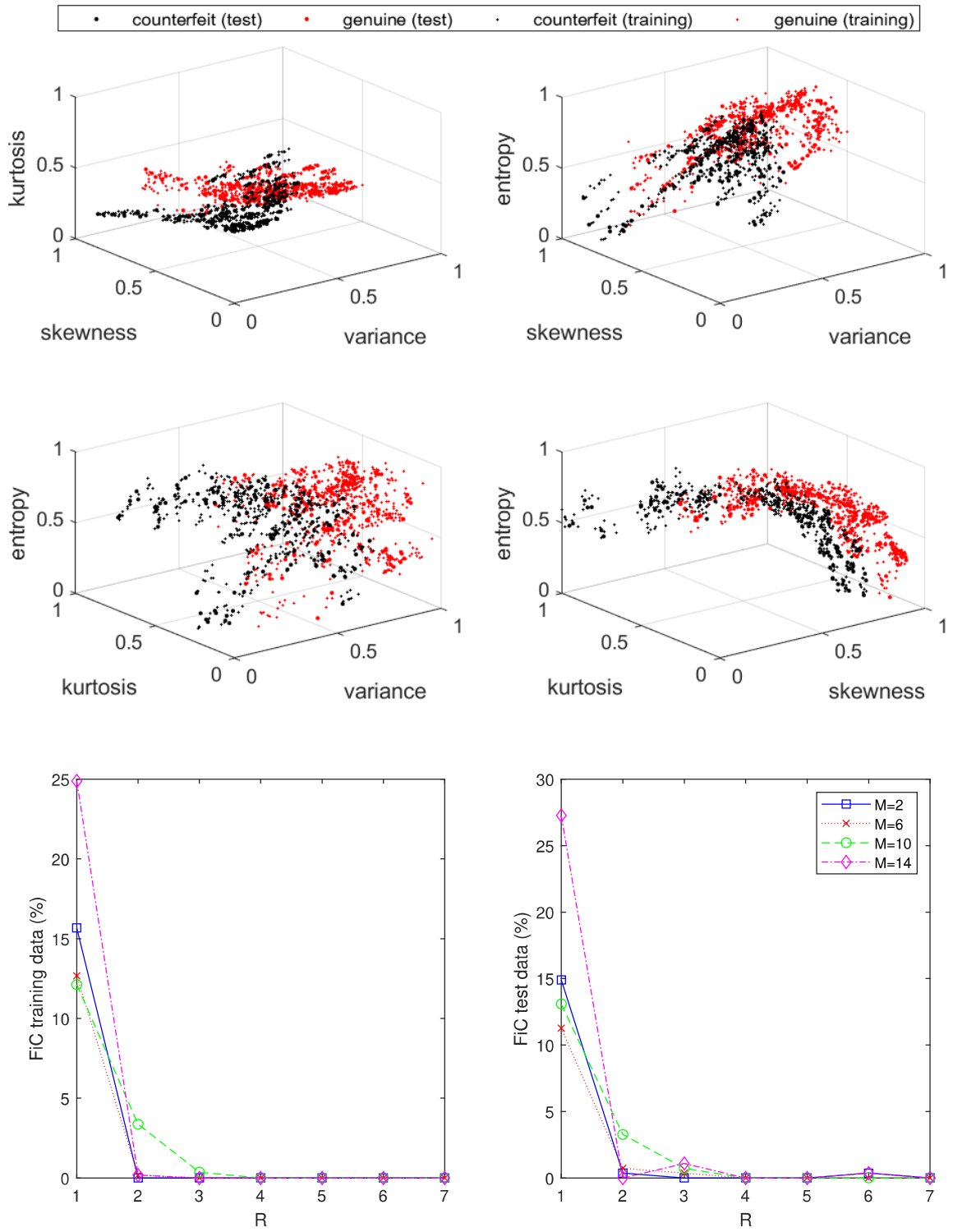


Figure 15. Even though the labeled data of the banknote problem live in a four-dimensional space, the classification problem can still be effectively "solved" with a low-rank classifier (21). Specifically, as seen in the bottom plot, an $R = 2$ separable expansion is sufficient to obtain a well-performing classifier with marginal error rates. The top four plots display the labeled data associated with the banknote dataset from the UCI machine learning repository.