Pattern Recognition 122 (2022) 108337

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Kernelized support tensor train machines

Cong Chen [a], Kim Batselier [b], Wenjian Yu [c,*], Ngai Wong [a,*]

[a] Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong
[b] Delft Center for Systems and Control, Delft University of Technology, Delft, the Netherlands
[c] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 13 September 2020
Revised 15 September 2021
Accepted 18 September 2021
Available online 20 September 2021

Keywords:
Image classification
Tensor
Support tensor machine

ABSTRACT

Tensor, a multi-dimensional data structure, has been exploited recently in the machine learning community. Traditional machine learning approaches are vector- or matrix-based, and cannot handle tensorial data directly. In this paper, we propose a tensor train (TT)-based kernel technique for the first time, and apply it to the conventional support vector machine (SVM) for high-dimensional image classification with very small number of training samples. Specifically, we propose a kernelized support tensor train machine that accepts tensorial input and preserves the intrinsic kernel property. The main contributions are threefold. First, we propose a TT-based feature mapping procedure that maintains the TT structure in the feature space. Second, we demonstrate two ways to construct the TT-based kernel function while considering consistency with the TT inner product and preservation of information. Third, we show that it is possible to apply different kernel functions on different data modes. In principle, our method tensorizes the standard SVM on its input structure and kernel mapping scheme. This reduces the storage and computation complexity of kernel matrix construction from exponential to polynomial. The validity proof and computation complexity of the proposed TT-based kernel functions are provided elaborately. Extensive experiments are performed on high-dimensional fMRI and color images datasets, which demonstrates the superiority of the proposed scheme compared with the state-of-the-art techniques.

1. Introduction

Many real-world data appear in tensor format. In medical neuroimaging, functional magnetic resonance imaging (fMRI) is naturally a three-way tensor. However, fMRI images are often with very high dimensions, at the same time with a small number of samples considering the difficulty of collecting data. This is also known as the small sample size problem. On the other hand, it is preferred to process fMRI data on edge devices for fast diagnosis [1]. Nevertheless, due to the limitation of storage and computation resources, it is troublesome to deploy complicated models, such as deep learning networks, on edge devices. Alternatively, traditional machine learning models become more popular in such a scenario due to their lightweight nature and good generalization ability [2].

To make the traditional vector or matrix-based machine learning algorithms better processing tensor data, it has been popular recently to extend them to their tensorial formats. For example, neighborhood preserving embedding (NPE) was extended to tensor neighborhood preserving embedding (TNPE) in Dai and Yeung [3], support vector machines (SVMs) [4] to support tensor machines (STMs) in Tao et al. [5], and restricted Boltzmann machines to their tensorial formats in Nguyen et al. [6]. By reformulating the machine learning algorithms into their tensorial frameworks, a performance improvement can be observed. The main reasons for this improvement can be summarized as follows. Firstly, these tensor algorithms can naturally utilize the multi-way structure of the original tensor data, which is believed to be useful in many machine learning applications such as visual question answering systems [7] and image completion [8]. Secondly, vectorizing tensor data leads to high-dimensional vectors, which may cause overfitting especially when the training sample size is relatively small [9]. On the contrary, tensor-based approaches usually derive a more structural and robust model that commonly involves much fewer model parameters, which not only alleviates the overfitting problem but also saves a lot of storage and computation resources [10,11].

This paper focuses on developing an efficient classification algorithm to solve the small sample size problem with limited storage and computation resources. Although deep learning models show powerful capability on the image classification task, their size is usually linearly dependent on the data dimension (possibly millions in the case of fMRI), which leads to a huge storage and com-

* Corresponding authors.
E-mail addresses: chencong@eee.hku.hk (C. Chen), k.batselier@tudelft.nl (K. Batselier), yu-wj@tsinghua.edu.cn (W. Yu), nwong@eee.hku.hk (N. Wong).

https://doi.org/10.1016/j.patcog.2021.108337
0031-3203/

putation consumption. Moreover, deep learning models often need a large amount of data for training, which is not a good fit in the scenario we considered. Few-shot learning [12] is a recently popular technique to deal with small sample size problems. In [12], the authors proposed DeepFMRI, in which a pre-trained CNN is utilized to extract features to facilitate the subsequent training on the given small number of training samples. We note that the pre-trained is trained on extra generated data. However, there is no guarantee that those extra data have a similar distribution as the given training data. Therefore, the pre-trained model may not work well for some specific datasets. Moreover, the overall deep model in DeepFMRI needs to be carefully designed, which is also time-consuming. In this paper, we propose a kernelized support tensor train machine (K-STTM), which only has $\mathcal{O}(M)$ parameters, where M is the training sample number, and it is not related to the data dimension at all. Moreover, no pre-training and manual design are needed. In the proposed K-STTM, we first employ the tensor train (TT) decomposition [13] to decompose the given tensor data so that a more compact and informative representation of it can be derived. Secondly, we define a TT-based feature mapping strategy to derive a high-dimensional TT in the feature space. This strategy enables us to apply different feature mappings on different data modes, which naturally provides a way to leverage the multi-mode nature of tensorial data. Thirdly, we propose two ways to build the kernel matrix efficiently with the consideration of the consistency with the TT inner product and preservation of information. The constructed kernel matrix is then used by kernel machines to solve the image classification problems.

The proposed K-STTM is a tensorial extension of the conventional SVM. Different from the existing SVM tensorial extensions, K-STTM preserves three advantages.

- K-STTM is naturally a nonlinear classifier. Commonly, real-life data are not linearly separable. However, most existing SVM tensorial extensions are often based on a linear model (no kernel trick) and cannot deal with nonlinear classification problems.
- The proposed tensorial kernel mapping scheme is valid and we provide the proof for the first time (refer to Section 4.4). This is a theoretical guarantee for the successful training of a kernelized support tensor machine. However, this proof is absent in existing tensorial extensions of SVM.
- Due to the non-uniqueness of tensor decomposition, the kernel mappings of two similar tensors may be very different, which may lead to different predicting results (refer to Section 4.6). This is obviously unwanted. To solve this, for the first time, we propose a data decomposition scheme to make sure that similar tensors would have similar kernel mappings in the high-dimensional feature domain.

The superiority of our methods is validated through extensive fMRI and color image classification experiments. It is observed that our methods have significant improvements over other related state-of-the-art classification methods, including traditional machine learning methods (vector or tensor-based) and 3D convolutional neural networks. Furthermore, we propose an efficient kernel matrix construction method, which reduces the computational complexity from $\mathcal{O}(M^2I^d)$ to $\mathcal{O}(dIR^4 + M^2R_d^2I_d)$. Applying different kernel functions on different data modes is also investigated and shows more than 10% accuracy improvement compared with the baselines on most datasets.

2. Related works

As one of the most typical supervised learning algorithms, SVM [4] has achieved enormous success in pattern classification by minimizing the Vapnik–Chervonenkis dimensions and structural risk. However, a standard SVM can not deal with tensorial input directly. The first work that extends SVM to handle tensorial input is [5]. More precisely, a supervised tensor learning (STL) scheme was proposed to train a support tensor machine (STM), where the hyperplane parameters are modeled as a rank-1 tensor instead of a vector. For the parameter training, they employed the alternating projection optimization method.

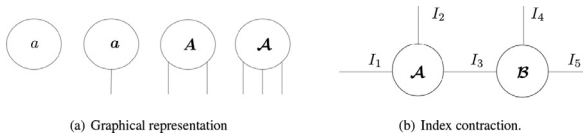
Although STM is capable to classify tensorial data directly, the expressive power of a rank-1 weight tensor is limited, which often leads to a poor classification accuracy. To increase the model expression capacity, several works were proposed recently based on the STL scheme. Ref. [14] employs a more general tensor structure, i.e., the canonical polyadic (CP) format, to replace the rank-1 weight tensor in STM. However, it is an NP-complete problem to determine the CP-rank. In [15], the STM is generalized to a support Tucker machine (STuM) by representing the weight parameter as a Tucker tensor. Nevertheless, the number of model parameters in STuM is exponentially large, which often leads to a large storage and computational complexity. To overcome this, Ref. [16] proposed a support tensor train machine (STTM), which assumes the potential weight tensor format is a TT. By doing so, the corresponding optimization problem is more scalable and can be solved efficiently. The aforementioned works are all based on the assumption that the given tensorial data are linearly separable. However, this is not the case in most real-world data. It is worth noting that though STTM sounds like the linear case of the proposed K-STTM, they are totally different when the linear kernel is applied on K-STTM. Specifically, K-STTM and STTM use two totally different schemes to train the corresponding model. For K-STTM, it first constructs the kernel matrix with the proposed TT-based kernel function, and then solves the standard SVM problem. However, in STTM, it assumes the parameter in the classification hyperplane can be modeled as a TT, and only updates one TT-core at a time by reformulating the training data.

To extend the linear tensorial classifiers to the nonlinear case, the authors in He et al. [17] proposed a nonlinear supervised learning scheme called dual structure-preserving kernels (DuSK). Specifically, based on the CP tensor structure, they define a corresponding kernel trick to map the CP format data into a higher-dimensional feature space. Through the introduction of the kernel trick, DuSK can achieve a higher classification accuracy. However, there is no proof to show that the proposed kernel mapping scheme is valid, i.e., there exists a feature space in which the inner product result is equivalent to the kernel function value of the original data space. We note that this is the theoretical guarantee for the successful training of an SVM-based method. Moreover, since DuSK is based on the CP decomposition, the non-deterministic polynomial (NP)-complete problem on the rank determination still exists. Through introducing a kernelized CP tensor factorization technique, the same research group in He et al. [17] further proposed the Multi-way Multi-level Kernel model [18] and kernelized support tensor machine model [19]. Nevertheless, the validity of the kernel mapping scheme and the CP-rank determination issues still exists.

To avoid the above issues, we propose the K-STTM, which not only introduces the customized kernel function to handle nonlinear classification problems, but also achieves an efficient model training since the scalable TT format is employed.

3. Preliminaries

In this section, we review some basic tensor notations and operations, together with the related tensor train decomposition method.



(a) Graphical representation (b) Index contraction.

Fig. 1. (a) depicts the graphical representation of a scalar a, vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$, and third-order tensor $\mathcal{A}$. (b) shows the index contraction between two 3-way tensors $\mathcal{A}$ and $\mathcal{B}$.

3.1. Tensor basics

Tensors in this paper are multi-dimensional arrays that generalize vectors (first-order tensors) and matrices (second-order tensors) to higher orders. A dth-order or d-way tensor is denoted as $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ and the element of $\mathcal{A}$ by $\mathcal{A}(i_1, i_2 \ldots, i_d)$, where $1 \leq i_k \leq I_k$, $k = 1, 2, \ldots, d$. The numbers $I_1, I_2, \ldots, I_d$ are called the dimensions of the tensor $\mathcal{A}$. We use boldface capital calligraphic letters $\mathcal{A}$, $\mathcal{B}$,...to denote tensors, boldface capital letters $\boldsymbol{A}$, $\boldsymbol{B}$,...to denote matrices, boldface letters $\boldsymbol{a}$, $\boldsymbol{b}$,...to denote vectors, and roman letters a, b,...to denote scalars. An intuitive and useful graphical representation of scalars, vectors, matrices and tensors is depicted in Fig. 1(a). The unconnected edges, also called free legs, are the indices of the tensor. Therefore scalars have no free legs, while a matrix has 2 free legs. We will employ these graphical representations to visualize the tensor networks and operations in the following sections whenever possible and refer to Orús [20] for more details. We now briefly introduce some important tensor operations.

Definition 1 (Tensor index contraction)**.** A tensor index contraction is the sum over all possible values of the repeated indices in a set of tensors.

For example, the following contraction of two 3-way tensors $\mathcal{A}$ and $\mathcal{B}$

$$\mathcal{C}(i_1, i_2, i_4, i_5) = \sum_{i_3=1}^{I_3} \mathcal{A}(i_1, i_2, i_3)\, \mathcal{B}(i_3, i_4, i_5),$$

over the i_3 index produces a four-way tensor $\mathcal{C}$. We also present the graphical representation of this contraction in Fig. 1(b), where the summation over i_3 is indicated by the connected edge. After this contraction, the tensor diagram contains four free legs indexed by i_1, i_2, i_4, i_5, respectively.

Definition 2 (Tensor mode-. k**product)** The mode-k product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \ldots \times I_k \times \ldots \times I_d}$ with a matrix $\boldsymbol{U} \in \mathbb{R}^{P_k \times I_k}$ is denoted as $\mathcal{B} = \mathcal{A} \times_k \boldsymbol{U}$ and defined by

$$\mathcal{B}(i_1, \ldots, i_{k-1}, j, i_{k+1}, \ldots, i_d) = \sum_{i_k=1}^{I_k} \boldsymbol{U}(j, i_k)\, \mathcal{A}(i_1, \ldots, i_k, \ldots, i_d),$$

where $\mathcal{B} \in \mathbb{R}^{I_1 \times \ldots \times I_{k-1} \times P_k \times I_{k+1} \times \ldots \times I_d}$. We note that Tensor mode-k product is a special case of tensor index contraction.

Definition 3 (Tensor inner product)**.** For two tensors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$, their inner product $\langle \mathcal{A}, \mathcal{B} \rangle$ is defined as

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \ldots \sum_{i_d=1}^{I_d} \mathcal{A}(i_1, i_2, \ldots, i_d) \mathcal{B}(i_1, i_2, \ldots, i_d).$$

Definition 4 (Tensor Frobenius norm)**.** The Frobenius norm of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$ is defined as $||\mathcal{A}||_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}$.

3.2. Tensor train decomposition

Here we briefly introduce the tensor train (TT) decomposition that will be utilized in the proposed K-STTM. A TT decomposition [13] represents a d-way tensor $\mathcal{A}$ as d 3-way tensors $\mathcal{A}^{(1)}$, $\mathcal{A}^{(2)}$,..., $\mathcal{A}^{(d)}$ such that a particular entry of $\mathcal{A}$ is written as the matrix product

$$\mathcal{A}(i_1, \ldots, i_d) = \mathcal{A}^{(1)}(:, i_1, :) \ldots \mathcal{A}^{(d)}(:, i_d, :), \tag{1}$$

where $\mathcal{A}^{(k)}(:, i_k, :)$ is naturally a matrix since we fix the second index. Each tensor $\mathcal{A}^{(k)}$, $k = 1, \ldots, d$, is called a *TT-core* and has dimensions $R_k \times I_k \times R_{k+1}$. Storage of a tensor as a TT therefore reduces from $\prod_{i=1}^{d} I_i$ down to $\sum_{i=1}^{d} R_i I_i R_{i+1}$. In order for the left-hand-side of (1) to be a scalar we require that $R_1 = R_{d+1} = 1$. The remaining R_k values are called the *TT-ranks*. A simple illustration of utilizing TT decomposition to factorize a 3-way tensor $\mathcal{A}$ is shown in Fig. 2. Note that the first and last TT cores are matrices since $R_1 = R_4 = 1$. A specific element in $\mathcal{A}$ is then computed as a vector-matrix-vector product. Fig. 3 demonstrates the general TT-decomposition of a d-way tensor $\mathcal{A}$, where the edges connecting the different circles indicate the matrix-matrix products of (1). We define the simplifying notation $TT(\mathcal{A})$, which denotes a TT decomposition of a d-way tensor $\mathcal{A}$ with user-specified TT-ranks.

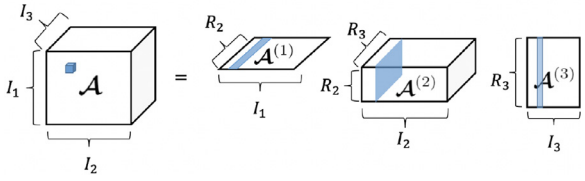


Fig. 2. The TT cores of a 3-way tensor $\mathcal{A}$ are two matrices $\mathcal{A}^{(1)}, \mathcal{A}^{(3)}$ and a 3-way tensor $\mathcal{A}^{(2)}$.

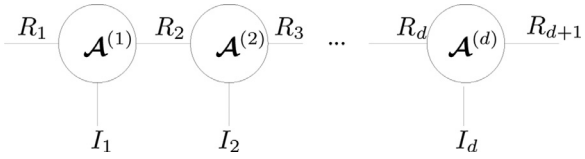


Fig. 3. Tensor train decomposition of a d-way tensor $\mathcal{A}$ into d 3-way tensors $\mathcal{A}^{(1)}, \mathcal{A}^{(2)} \ldots, \mathcal{A}^{(d)}$.

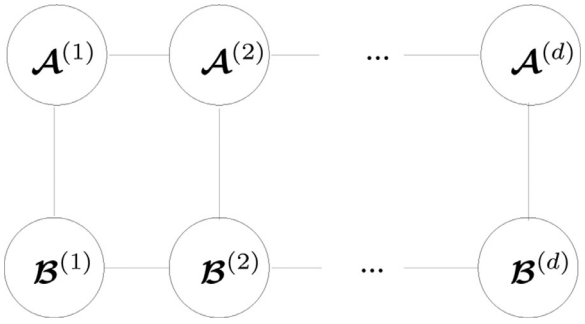


Fig. 4. The inner product between two d-way tensor trains.

Definition 5 (TT inner product)**.** The inner product between two tensor trains $TT(\mathcal{A})$ and $TT(\mathcal{B})$ is denoted as $\langle TT(\mathcal{A}), TT(\mathcal{B}) \rangle$.

The tensor network diagram of the inner product of two TTs is shown in Fig. 4. The lack of unconnected edges in Fig. 4 implies that $\langle TT(\mathcal{A}), TT(\mathcal{B}) \rangle$ is a scalar.

3.3. Support vector machines

Since this work is based on traditional SVM, we therefore briefly review the main idea of an SVM. Assume we have a dataset $D = \{\boldsymbol{x_i}, y_i\}_{i=1}^{M}$ of M labeled samples, where $\boldsymbol{x}_i \in \mathbb{R}^n$ are the vectorized data samples with labels $y_i \in \{-1, 1\}$. The goal of an SVM is to find a discriminant hyperplane

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b \tag{2}$$

that maximizes the margin between the two classes where $\boldsymbol{w}$ and b are the weight vector and bias, respectively. However, an SVM is very sensitive to noise since it requires all the training samples to meet the hard margin constraint. In that case, the trained model tends to overfit. To solve this, slack variables $\xi_1, \ldots, \xi_M$ are introduced to allow some certain samples to be misclassified, thus enhancing the robustness of the trained model. We can express the learning problem as a quadratic optimization problem

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \tfrac{1}{2}\|\boldsymbol{w}\|_F^2 + C\textstyle\sum_{i=1}^M \xi_i \\ \text{subject to} \quad & y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, M. \end{aligned} \tag{3}$$

The parameter C controls the trade-off between the size of the weight vector $\boldsymbol{w}$ and the size of the slack variables. It is more common to solve the dual problem of (3), especially when the feature size n is larger than the sample size M. The dual problem format of (3) is

$$\begin{aligned} \min_{\alpha_1,\alpha_2,\ldots,\alpha_M} \quad & \textstyle\sum_{i=1}^M \alpha_i - \frac{1}{2}\sum_{i,j=1}^M \alpha_i\alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle \\ \text{subject to} \quad & \textstyle\sum_{i=1}^M \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C,\ i = 1, \ldots, M, \end{aligned} \tag{4}$$

where $\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$ represents the inner product between vector $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and $\alpha_i\ (i = 1, \ldots, M)$ are the Lagrange multipliers.

To solve a nonlinear classification problem with SVM, a nonlinear mapping function ϕ is introduced that projects the original vectorial data onto a much higher-dimensional feature space. In the feature space, the data generally become more (linearly) separable. By doing so, the optimization in (4) is transformed into

$$\min_{\alpha_1,\alpha_2,\ldots,\alpha_M} \quad \sum_{i=1}^M \alpha_i - \frac{1}{2}\sum_{i,j=1}^M \alpha_i\alpha_j y_i y_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle \tag{5}$$

with the same constraints as in (4). The kernel trick allows us to compute the inner product term $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle$ with a kernel function $\boldsymbol{k}(\boldsymbol{x}_i, \boldsymbol{x}_j)$, thus avoiding the explicit construction of the possibly infinite-dimensional $\phi(\boldsymbol{x}_i)$ vectors.

4. Kernelized support tensor train machines

In this section, we first demonstrate the tensor-based kernel learning problem and then introduce the proposed K-STTM.

4.1. Problem statement

Given M tensorial training data and their labels, i.e., dataset $D = \{\mathcal{X}_i, y_i\}_{i=1}^M$, where $\mathcal{X}_i \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_d}$ and $y_i \in \{-1, 1\}$, we want to find a hyperplane

$$f(\mathcal{X}) = \langle \mathcal{W}, \mathcal{X}\rangle + b \tag{6}$$

that separates the tensorial data into two classes. $\mathcal{W}$ is the hyperplane weight tensor with the same dimensions as $\mathcal{X}_i$ and b is the bias. Similar to the primal problem in SVM, we can derive the corresponding primal optimization problem for (6)

$$\begin{aligned} \min_{\mathcal{W},b,\xi} \quad & \tfrac{1}{2}\|\mathcal{W}\|_F^2 + C\textstyle\sum_{i=1}^M \xi_i \\ \text{subject to} \quad & y_i(\langle \mathcal{W}, \mathcal{X}_i\rangle + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, M. \end{aligned} \tag{7}$$

Following the scheme of the kernel trick for conventional SVMs, we introduce a nonlinear feature mapping function $\Phi(\cdot)$. Then, given a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\ldots\times I_d}$, we assume it is mapped into the Hilbert space $\mathcal{H}$ by

$$\Phi(\cdot) : \mathbb{R}^{I_1\times I_2\times\ldots\times I_d} \to \mathbb{R}^{H_1\times H_2\times\ldots\times H_d}. \tag{8}$$

We need to mention that the dimension of projected tensor $\Phi(\mathcal{X})$ can be infinite depending on the feature mapping function $\Phi(\cdot)$. The resulting Hilbert space is then called the *tensor feature space* and we can further develop the following model

$$\begin{aligned} \min_{\mathcal{W},b,\xi} \quad & \tfrac{1}{2}\|\mathcal{W}\|_F^2 + C\textstyle\sum_{i=1}^M \xi_i \\ \text{subject to} \quad & y_i(\langle \mathcal{W}, \Phi(\mathcal{X}_i)\rangle + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, M, \end{aligned} \tag{9}$$

with parameter tensor $\mathcal{W} \in \mathbb{R}^{H_1\times H_2\times\ldots\times H_d}$. To obtain the tensor-based kernel optimization model, we need to transfer model (9) into its dual, namely

$$\begin{aligned} \min_{\alpha_1,\alpha_2,\ldots,\alpha_M} \quad & \textstyle\sum_{i=1}^M \alpha_i - \frac{1}{2}\sum_{i,j=1}^M \alpha_i\alpha_j y_i y_j \langle \Phi(\mathcal{X}_i), \Phi\left(\mathcal{X}_j\right)\rangle \\ \text{subject to} \quad & \textstyle\sum_{i=1}^M \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C,\ i = 1, \ldots, M, \end{aligned} \tag{10}$$

where α_i are the Lagrange multipliers. The key task we need to solve is to define a tensorial kernel function $\mathcal{K}(\mathcal{X}_i, \mathcal{X}_j)$ that computes the inner product $\langle \Phi(\mathcal{X}_i), \Phi(\mathcal{X}_j)\rangle$ in the original data space instead of the feature space.

4.2. Customized kernel mapping schemes for TT-based data

Although tensor is a natural structure for representing real-world data, there is no guarantee that such a representation works well for kernel learning. Instead of the full tensor, here we employ a TT for data representation due to the following reasons:

1. Real-life data often contain redundant information, which is not useful for kernel learning. The TT decomposition has proven to be efficient for removing the redundant information in the original data and provides a more compact data representation.
2. Compared to the Tucker decomposition whose storage scales exponentially with the core tensor, a TT is more scalable (parameter number grows linearly with the tensor order d), which reduces the computation during kernel learning.
3. Unlike the CP decomposition, determining the TT-rank is easily achieved through a series of singular value decompositions (TT-SVD [13]). Moreover, instead of decomposing many tensorial data sample by sample, it is possible to stack them together and decompose the stacked tensor with TT-SVD in one shot. This naturally leads to a faster data transformation to the TT format.
4. It is convenient to implement different operations on different tensor modes when data is in the TT format. Since a TT decomposition decomposes the original data into many TT cores, it is possible to apply different kernel functions on different TT cores for better classification performance. Furthermore, we can emphasize the importance of different tensor modes by putting different weights on those TT cores during the kernel mapping. For example, a color image is a 3-way (pixel-pixel-color) tensor. The color mode can be treated differently with the two pixel modes since they contain different kinds of information, as will be exemplified later.

In the following, we demonstrate the proposed TT-based feature mapping approach. Specifically, we map all fibers in each TT-core to the feature space through

$$\phi_i(\cdot) : \mathbb{R}^{I_i} \to \mathbb{R}^{H_i}, \quad i = 1, \ldots, d,$$

such that

$$\begin{aligned} & \phi_i\left(\mathcal{X}^{(i)}(r_i, :, r_{i+1})\right) \in \mathbb{R}^{H_i} \\ & 1 \leq r_i \leq R_i, \quad 1 \leq r_{i+1} \leq R_{i+1}, \quad i = 1, \ldots, d, \end{aligned} \tag{11}$$

where $\mathcal{X}^{(i)}$ and R_i are the ith TT-core and TT-rank of $TT(\mathcal{X})$, respectively. The fibers of each TT-core are vectors as the rank indices r_i, r_{i+1} are fixed to a specific value, hence the feature mapping works in the same way as for the conventional SVM. We then represent the resulting high-dimensional TT, which is in the

tensor feature space, as $\Phi(TT(\mathcal{X})) \in \mathbb{R}^{H_1 \times H_2 \times \ldots \times H_d}$. We stress that $\Phi(TT(\mathcal{X}))$ is still in a TT format with the same TT-ranks as $TT(\mathcal{X})$. In this sense, the TT format data structure is preserved after the feature mapping.

After mapping the TT format data into the TT-based high-dimensional feature space, we then propose two approaches for computing the inner product between two mapped TT format data using kernel functions.

4.2.1. K-STTM-Prod

The first method is called K-STTM-Prod since we implement consecutive multiplication operations on d fiber inner products, which is consistent with the result of an inner product between two TTs. Assuming $\Phi(TT(\mathcal{X}))$ and $\Phi(TT(\mathcal{Y})) \in \mathbb{R}^{H_1 \times H_2 \times \ldots \times H_d}$ with TT-ranks R_i and $\hat{R}_i$, $i = 1, 2, \ldots, d$, respectively, their inner product can be computed from

$$\begin{aligned}\langle \Phi(TT(\mathcal{X})), \Phi(TT(\mathcal{Y})) \rangle = \textstyle\sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} \sum_{\hat{r}_1=1}^{\hat{R}_1} \cdots \sum_{\hat{r}_d=1}^{\hat{R}_d} \\ \left(\textstyle\prod_{i=1}^{d} \langle \phi_i\left(\mathcal{X}^{(i)}(r_i, :, r_{i+1})\right), \phi_i\left(\mathcal{Y}^{(i)}\left(\hat{r}_i, :, \hat{r}_{i+1}\right)\right)\rangle\right).\end{aligned} \tag{12}$$

We remark that (12) derives the exact same result as Fig. 4 (assuming $\mathcal{X} = \mathcal{A}$ and $\mathcal{Y} = \mathcal{B}$) when an identity feature mapping function $\Phi(\cdot)$ is used, namely $\Phi(TT(\mathcal{X})) = TT(\mathcal{X})$. What is more, since each fiber of a mapped TT-core is naturally a vector, we have

$$\begin{aligned}&\langle \phi_i(\mathcal{X}^{(i)}(r_i, :, r_{i+1})), \phi_i(\mathcal{Y}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})) \rangle \\ &= \boldsymbol{k}_i(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \mathcal{Y}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})),\end{aligned} \tag{13}$$

where $\boldsymbol{k}_i(\cdot)$ can be any kernel function used for a standard SVM, such as a Gaussian radial basis function (RBF) kernel, polynomial kernel, linear kernel etc. Combining (12) and (13), we obtain the corresponding TT-based kernel function

$$\begin{aligned}\mathcal{K}(TT(\mathcal{X}), TT(\mathcal{Y})) = \sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} \sum_{\hat{r}_1=1}^{\hat{R}_1} \cdots \sum_{\hat{r}_d=1}^{\hat{R}_d} \\ \left(\prod_{i=1}^{d} \boldsymbol{k}_i(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \mathcal{Y}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1}))\right).\end{aligned} \tag{14}$$

As mentioned before, in K-STTM setting, different kernel functions $\boldsymbol{k}_i$ can be applied on different tensor modes i. One possible application is in color image classification, where one could apply Gaussian RBF kernels $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$ on its first two spatial modes, while choosing a linear or polynomial kernel $\boldsymbol{k}_3$ for the color mode. This will be investigated in the experiments.

4.2.2. K-STTM-Sum

The second method we propose to construct a TT kernel function is called K-STTM-Sum. Instead of implementing continuous multiplication operations on d fiber inner products like in K-STTM-Prod, K-STTM-Sum performs consecutive addition operations on them. This idea is inspired by Houthuys and Suykens [21] which argues that the product of inner products can lead to the loss/misinterpretation of information. Take the linear kernel as an example, the inner product between two fibers of the same mode could be negative, which indicates a low similarity between those two fibers. However, by implementing consecutive multiplications of d fiber inner products, highly negative values could result in a large positive value. In that case, the overall similarity is high which is clearly unwanted. This situation also appears when employing Gaussian RBF kernels. A nearly zero value would be assigned to two non-similar fibers, which could influence the final result significantly. To this end, we propose the K-STTM-Sum. Similar to K-STTM-Prod, we can obtain the corresponding kernel function as

$$\begin{aligned}\mathcal{K}(TT(\mathcal{X}), TT(\mathcal{Y})) = \sum_{r_1=1}^{R_1} \cdots \sum_{r_d=1}^{R_d} \sum_{\hat{r}_1=1}^{\hat{R}_1} \cdots \sum_{\hat{r}_d=1}^{\hat{R}_d} \\ \left(\sum_{i=1}^{d} \boldsymbol{k}_i(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \mathcal{Y}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1}))\right).\end{aligned} \tag{15}$$

4.3. Kernel optimization problem

After defining the TT-based kernel function, we can then replace the term $\langle \Phi(\mathcal{X}_i), \Phi(\mathcal{X}_j) \rangle$ in (10) with (14) or (15), and derive our final kernel optimization problem based on the TT structure, namely,

$$\begin{aligned}\min_{\alpha_1, \alpha_2, \ldots, \alpha_M} \textstyle\sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j \mathcal{K}\left(TT(\mathcal{X}_i), TT\left(\mathcal{X}_j\right)\right) \\ \text{subject to} \quad \textstyle\sum_{i=1}^{M} \alpha_i y_i = 0, \\ 0 \le \alpha_i \le C, \ i = 1, \ldots, M.\end{aligned} \tag{16}$$

After solving (16), we can get the unknown model parameters $\alpha_1, \alpha_2, \ldots, \alpha_M$ and the resulting decision function is then represented as

$$f(\mathcal{X}) = \operatorname{sign}\left(\sum_{i=1}^{M} \alpha_i y_i \mathcal{K}(TT(\mathcal{X}_i), TT(\mathcal{X})) + b\right). \tag{17}$$

4.4. Kernel validity

According to Mercer's condition, a kernel function is valid when the constructed kernel matrix is symmetric and positive semi-definite on the given training data. This guarantees that the mapped high-dimensional feature space is truly an inner product space. Therefore, we provide Theorem 1 to show the validity of K-STTM-Prod and K-STTM-Sum. In the actual implementation of K-STTM-Prod and K-STTM-Sum, it is extremely inefficient to use TT decomposition to decompose each tensorial sample one by one. The way we did it is by first stacking all the d-way samples and then compute a TT decomposition on the resulting $(d+1)$-way tensor directly. This procedure is explained in detail in Section 4.5. By doing so, all TT-based training samples have the same TT-ranks. Also in the case where we compute the TT decomposition separately for each sample, we can still set the TT-ranks of all samples to be identical. That means R_i is equal to $\hat{R}_i$, $i = 1, 2, \ldots, d$ for all the TT-based training samples. This setting is also assumed in Theorem 1 and its proof.

Before we show Theorem 1 and its proof, we first prove the following lemma, which is helpful in the proof of Theorem 1.

Lemma 1. *The summation and Hadamard product between two symmetric and positive semi-definite matrices $\boldsymbol{A}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ still results in a symmetric and positive semi-definite matrix.*

Proof. According to the definition of symmetric and positive semi-definite matrix, we have

$$\begin{cases} \boldsymbol{A} = \boldsymbol{A}^T, & \boldsymbol{u}^T \boldsymbol{A} \boldsymbol{u} \ge 0 \\ \boldsymbol{B} = \boldsymbol{B}^T, & \boldsymbol{u}^T \boldsymbol{B} \boldsymbol{u} \ge 0, \end{cases}$$

for every non-zero column vector $\boldsymbol{u} \in \mathbb{R}^n$.

For the summation case, obviously we can conclude that $(\boldsymbol{A} + \boldsymbol{B})^T = \boldsymbol{A} + \boldsymbol{B}$; $\boldsymbol{u}^T(\boldsymbol{A} + \boldsymbol{B})\boldsymbol{u} \ge 0$, namely $\boldsymbol{A} + \boldsymbol{B}$ is still symmetric and positive semi-definite.

For the Hadamard product case, we refer to the Schur product theorem [22] and we can easily obtain $\boldsymbol{u}^T(\boldsymbol{A} \odot \boldsymbol{B})\boldsymbol{u} \ge 0$, for every non-zero column vector $\boldsymbol{u} \in \mathbb{R}^n$, where $\odot$ is the Hadamard product. It is obvious that $(\boldsymbol{A} \odot \boldsymbol{B})^T = (\boldsymbol{A} \odot \boldsymbol{B})$. Thus $\boldsymbol{A} \odot \boldsymbol{B}$ is still symmetric and positive semi-definite. □

We then demonstrate Theorem 1 and its proof here.

Theorem 1. *Given a tensorial training dataset $\{\mathcal{X}_i\}_{i=1}^{M}$, where $\mathcal{X}_i \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$, and assumed TT-ranks $R_1, \ldots, R_d$, the proposed kernel functions K-STTM-Prod and K-STTM-Sum are valid kernel functions*

and they produce symmetric and positive semi-definite kernel matrices.

Proof. We first demonstrate the kernel function validity of K-STTM-Prod. For any tensor $\mathcal{X}$, $\mathcal{Y} \in \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_M\}$, they are first decomposed into their TT formats, namely $TT(\mathcal{X})$ and $TT(\mathcal{Y})$, after which Eq. (14) is applied. Assuming all the indices over $\sum$ and $\prod$, namely $r_1, \ldots, r_d, \hat{r}_1, \ldots, \hat{r}_d$ and i, are fixed, Eq. (14) can be written as

$$\mathcal{K}(TT(\mathcal{X}), TT(\mathcal{Y})) = \boldsymbol{k}_i(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \mathcal{Y}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})). \tag{18}$$

As we mentioned before, $\boldsymbol{k}_i(\cdot, \cdot)$ can be any valid kernel function used for a standard SVM. Therefore, the kernel matrix constructed by Eq. (18) is symmetric and positive semi-definite. When only the indices over $\sum$, namely $r_1, \ldots, r_d, \hat{r}_1, \ldots, \hat{r}_d$, are fixed, Eq. (14) can be written as the following product kernel

$$\mathcal{K}(TT(\mathcal{X}), TT(\mathcal{Y})) = (\prod_{i=1}^{d} \boldsymbol{k}_i(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \mathcal{Y}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1}))). \tag{19}$$

The kernel matrix constructed by Eq. (19) can be regarded as Hadamard products of the d valid kernel matrices constructed by Eq. (18) when $i = 1, \ldots, d$. Since the matrix constructed by Eq. (18) is symmetric and positive semi-definite, according to Lemma 1, the matrix constructed by Eq. (19) is also symmetric and positive semi-definite.

Similarly, we notice that the kernel matrix constructed by Eq. (14) can be regarded as the summation of $R_1 \times \ldots \times R_d \times \hat{R}_1 \times \ldots \times \hat{R}_d$ kernel matrices constructed by Eq. (19) when $r_1, \ldots, r_d, \hat{r}_1, \ldots, \hat{r}_d$ are varied from 1 to their corresponding maximum values. According to Lemma 1, we conclude that the kernel matrix constructed by Eq. (14) is symmetric and positive semi-definite, namely K-STTM-Prod is a valid kernel function.

The validity proof for K-STTM-Sum is similar to the proof for K-STTM-Prod. The kernel matrix constructed by Eq. (15) can be regarded as the summation of $R_1 \times \ldots \times R_d \times \hat{R}_1 \times \ldots \times \hat{R}_d \times d$ kernel matrices constructed by Eq. (18) when $r_1, \ldots, r_d, \hat{r}_1, \ldots, \hat{r}_d$ and i are varied from 1 to their corresponding maximum values. According to Lemma 1, the kernel matrix constructed by Eq. (15) is symmetric and positive semi-definite. Therefore, K-STTM-Sum is a valid kernel function. □

4.5. Efficient implementation for kernel matrix construction

The main computation bottleneck in K-STTM lies in the kernel matrix construction. In Eqs. (14) and (15), there are both $2d + 1$ times consecutive summation or multiplication operations, which is time-consuming if we use $2d + 1$ for-loops to compute each element in the final kernel matrices of K-STTM-Prod and K-STTM-Sum. Therefore, we propose an efficient implementation. Here we first demonstrate the implementation detail of K-STTM-Prod. The computation of Eq. (14) is first separated into d parts, one for each TT-core. For example, the ith part calculates the following values

$$\boldsymbol{k}_i(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \mathcal{Y}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})), \tag{20}$$

where $1 < r_i < R_i$, $1 < \hat{r}_i < \hat{R}_i$ and $1 < r_{i+1} < R_{i+1}$, $1 < \hat{r}_{i+1} < \hat{R}_{i+1}$. This leads to a matrix with dimensions $R_i R_{i+1} \times \hat{R}_i \hat{R}_{i+1}$. Copies of this matrix are repeated into a matrix $\boldsymbol{X}_i$ of dimensions $(R_1 \ldots R_d) \times (\hat{R}_1 \ldots \hat{R}_d)$. All parts are then combined through

$$\boldsymbol{X}_{\text{prod}} = \boldsymbol{X}_1 \odot \boldsymbol{X}_2 \odot \ldots \odot \boldsymbol{X}_d, \tag{21}$$

where $\odot$ is the hadamard product. We then sum over all the elements in $\boldsymbol{X}_{\text{prod}}$, which generates the results of Eq. (14).

The implementation detail of K-STTM-Sum is similar. The only difference with K-STTM-Prod is that Eq. (21) is replaced by

$$\boldsymbol{X}_{\text{sum}} = \boldsymbol{X}_1 + \boldsymbol{X}_2 + \ldots + \boldsymbol{X}_d. \tag{22}$$

After summing over all the elements in $\boldsymbol{X}_{\text{sum}}$, we obtain the result of Eq. (15).

The above implementations all make use of matrix operations, which leans itself well to software platforms such as MATLAB or hardware acceleration platforms such as a GPU. There are two additional ways to accelerate the construction of the kernel matrix further.

First, assuming the given dataset consists of M samples $\mathcal{X}_i$, we first stack them together into a tensor $\mathcal{X}_{\text{stack}}$ with dimensions $I_1 \times I_2 \times \ldots \times I_d \times M$. We then decompose $\mathcal{X}_{\text{stack}}$ with assumed TT-ranks $R_1, \ldots, R_d$ into d TT cores, namely $\mathcal{X}^{(i)}_{\text{stack}}, i = 1, \ldots, d$, where the last TT-core $\mathcal{X}^{(d)}_{\text{stack}}$ has dimensions $R_d \times I_d \times M$. We note that all the training samples share the same $d - 1$ TT cores $\mathcal{X}^{(i)}_{\text{stack}}, i = 1, \ldots, d - 1$. The dth TT-core of each sample is derived by selecting the corresponding slices of $\mathcal{X}^{(d)}_{\text{stack}}$ through fixing the third dimension, namely $\mathcal{X}^{(d)}_{\text{stack}}(:, :, \text{sampleIndex})$. With this property, $\boldsymbol{X}_i, i = 1, \ldots, d - 1$ in Eqs. (21) and (22) are the same no matter which two training samples are employed to compute the kernel function value. Therefore, we only need to compute the cores $\boldsymbol{X}_i$ once instead of M^2 times.

Second, we can compute the kernel matrix in a parallel manner. Specifically, we compute the kernel function values between one sample and all other training samples in one shot. This can be easily achieved by computing $\mathcal{K}(TT(\mathcal{X}_i), TT(\mathcal{X}_{\text{stack}}))$. The computation is also separated into d parts and each part is computed similarly as we discussed for Eq. (20), but the resulting matrices are repeated to larger-size matrices $\boldsymbol{X}_i, i = 1, \ldots, d$ with dimensions $(R_1 \ldots R_d) \times (\hat{R}_1 \ldots \hat{R}_d M)$. Those $\boldsymbol{X}_i$, matrices are then combined using Eqs. (21) and (22) to generate $\boldsymbol{X}_{\text{prod}}$ or $\boldsymbol{X}_{\text{sum}}$, respectively. We then sum over the elements of $\boldsymbol{X}_{\text{prod}}$ or $\boldsymbol{X}_{\text{sum}}$ block by block with block size $(R_1 \ldots R_d) \times (\hat{R}_1 \ldots \hat{R}_d)$, which leads to a row vector of size $1 \times M$. This produces one row of the final kernel matrix.

After explaining the efficient implementation of kernel matrix construction, we compare the difference in computational complexity in flops (considering multiplication only) between the naive kernel matrix construction (using $2d + 1$ times consecutive summation or multiplication) and our efficient way. Given a tensorial training dataset $\{\mathcal{X}_i\}_{i=1}^{M}$, where $\mathcal{X}_i \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$, and assumed TT-ranks $R_1, \ldots, R_d$, we need $\mathcal{O}(dM^2IR^{2d})$ and $\mathcal{O}(M^2IR^{2d})$ flops for the kernel matrix construction of K-STTM-Prod and K-STTM-Sum respectively when employing the naive way, where I and R are the maximum values of I_i and R_i, $i = 1, 2, \ldots d$, respectively. Using our aforementioned method the computation of $\boldsymbol{X}_i$ costs $\mathcal{O}(dIR^4)$ flops and $\mathcal{O}(M^2R_d^2I_d)$ flops for computing $\boldsymbol{X}_d$. For K-STTM-Sum, the overall computation is then $\mathcal{O}(dIR^4 + M^2R_d^2I_d)$, where I and R are the maximum values of I_i and R_i, $i = 1, 2, \ldots d - 1$. The reduction in computational cost is primarily from the reduction of the R^{2d} factor to R^4. For K-STTM-Prod, the overall computation is $\mathcal{O}(dIR^4 + M^2R_d^2I_d + M^2R^{2d} + MdR^{2d})$, I_i and R_i, $i = 1, 2, \ldots d$. We note that real-world data is commonly low-rank, so the TT-ranks R_i are generally small. Moreover, their dimensions I_i are commonly high. Therefore, the computational cost for the computation of K-STTM-Prod and K-STTM-Sum are similar as they have the same multiplication factor on I, which is much smaller than the case in the naive way. We also investigate this in our Section 5.2, in which K-STTM-Prod and K-STTM-Sum cost similar time on kernel matrix construction.

The actual implementation of the kernel matrix construction can be found in our open-source MATLAB code at: https://github.com/git2cchen/KSTTM.git

4.6. Non-uniqueness of TT-SVD

In this paper, we employ the TT-SVD [13] algorithm to convert the data into the TT format. However, we note that the TT decomposition of a tensor is not unique. This is not an issue for tensor compression, but it affects the classification result when using TT cores. Specifically, referring back to the definition of TT decomposition, namely Eq. (1), we can derive a specific TT decomposition of a d-way tensor $\mathcal{X}$, namely $TT_1(\mathcal{X})$. However, the TT decomposition for $\mathcal{X}$ is not unique. If we apply an invertible linear transformation $\boldsymbol{U} \in \mathbb{R}^{R_{k+1} \times R_{k+1}}$ on the third mode of $\mathcal{X}^{(k)}$, namely $\mathcal{X}^{(k)} \times_3 \boldsymbol{U}^T$, and apply $\boldsymbol{U}^{-1}$ on the first mode of $\mathcal{X}^{(k+1)}$, namely $\mathcal{X}^{(k+1)} \times_1 \boldsymbol{U}^{-1}$, we derive another TT decomposition of $\mathcal{X}$, which we call $TT_2(\mathcal{X})$. $TT_1(\mathcal{X})$ and $TT_2(\mathcal{X})$ are different TT decompositions of the same tensor $\mathcal{X}$. If we compute $\mathcal{K}(TT_1(\mathcal{X}), TT_2(\mathcal{X}))$, the result may be different with $\mathcal{K}(TT_1(\mathcal{X}), TT_1(\mathcal{X}))$. This is because the kernel function in K-STTM is evaluated in the fibers of TT cores, which are different between $TT_1(\mathcal{X})$ and $TT_2(\mathcal{X})$ due to the applied linear transformation. The extreme case occurs for even tensor order d when all TT cores $\mathcal{X}^{(i)}, i = 1, \ldots, d$ in $TT_1(\mathcal{X})$ have opposite signs with the TT cores $\hat{\mathcal{X}}^{(i)}, i = 1, \ldots, d$ in $TT_2(\mathcal{X})$, namely $\mathcal{X}^{(i)} = -\hat{\mathcal{X}}^{(i)}, \quad i = 1, \ldots, d$. In this case, when we compute $\boldsymbol{k}(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \hat{\mathcal{X}}^{(i)}(r_i, :, r_{i+1}))$, we would derive a totally different result with $\boldsymbol{k}(\mathcal{X}^{(i)}(r_i, :, r_{i+1}), \mathcal{X}^{(i)}(r_i, :, r_{i+1}))$, which is unwanted since we may get different predicted labels for two identical tensors in the classification task just because of the non-uniqueness of TT-SVD. We note that this issue also appears in other kernelized support tensor machines, such as DuSK [17], because the CP decomposition of a tensor is only unique up to an arbitrary scale indeterminacy. However, this problem was not discussed in He et al. [17] and no solution was provided either.

To solve the above issue, we stack all the training tensorial samples together into $\mathcal{X}_{\text{stack}}$ and decompose it into its TT format by TT-SVD in one shot. By doing so, all the samples have the same first $d-1$ TT cores, thus would not suffer the above mentioned issues. Apart from the training data, we also need to constrain that the TT format of validation and testing data employ the same first $d-1$ TT cores as the TT format of training data. Specifically, assuming the stacked training tensor is $\mathcal{X}_{\text{trainStack}}$, its TT format $TT(\mathcal{X}_{\text{trainStack}})$ can then be derived. We contract its first $d-1$ TT cores and reshape it into a matrix $\boldsymbol{P}$ with dimension $(I_1 \ldots I_{d-1}) \times R_d$. $\boldsymbol{P}$ can also be regarded as a projection matrix, which projects the training, validation and testing data into the same tensorial space. We further reshape the stacked validation and testing tensorial data as $\boldsymbol{X}_{\text{validStack}}$ and $\boldsymbol{X}_{\text{testStack}}$ with dimensions $(I_1 \ldots I_{d-1}) \times (I_d N)$ and $(I_1 \ldots I_{d-1}) \times (I_d O)$, respectively, where N and O are the number of validation and testing samples, respectively. The dth TT-core of validation and test data are then computed as follows

$$\mathcal{X}^{(d)}_{\text{validStack}} = \text{reshape}(\boldsymbol{P}^{\dagger}\boldsymbol{X}_{\text{validStack}}, [R_d, I_d, N]), \tag{23}$$

$$\mathcal{X}^{(d)}_{\text{testStack}} = \text{reshape}(\boldsymbol{P}^{\dagger}\boldsymbol{X}_{\text{testStack}}, [R_d, I_d, O]), \tag{24}$$

where $\boldsymbol{P}^{\dagger}$ is the pseudo-inverse of $\boldsymbol{P}$. Apart from the last TT-core, the first $d-1$ TT cores are identical to the ones in $TT(\mathcal{X}_{\text{trainStack}})$, namely $\mathcal{X}^{(i)}_{\text{validStack}} = \mathcal{X}^{(i)}_{\text{testStack}} = \mathcal{X}^{(i)}_{\text{trainStack}}, i = 1, \ldots, d-1$.

We note that the TT decomposition for $\mathcal{X}_{\text{trainStack}}$ is not unique for the same reason as mentioned above. Therefore, with the same training samples, we may still get different trained K-STTMs. However, as long as we constrain the training, validation and test samples to employ the same first $d-1$ TT cores, similar tensors (which may belong to the same classification category) would be decomposed into similar TTs. This ensures the success of TT-based kernel learning tasks, which is empirically confirmed through experiments.

The training algorithm of the K-STTM-Prod/Sum is described as pseudo-code in Algorithm 1 using a Gaussian RBF kernel as an example. Hyperparameters can be tuned through a grid search or through cross-validation. If other kernel functions are employed, the grid search for σ in step 5 can be replaced accordingly. Generalizing the binary classification to multi-classification can be easily achieved by utilizing an one-vs-one or one-vs-all strategy, namely, we can build several binary classifiers to do multi-class classification.

Algorithm 1 K-STTM-Prod/Sum algorithm.

Input: Training dataset $\{\mathcal{X}_i \in \mathbb{R}^{I_1 \times \ldots \times I_d}, y_i \in \{-1, 1\}\}_{i=1}^{M}$; Validation dataset $\{\mathcal{X}_j \in \mathbb{R}^{I_1 \times \ldots \times I_d}, y_j \in \{-1, 1\}\}_{j=1}^{N}$; The preset TT-ranks $R_1, R_2, \ldots, R_{d+1}$; The range of the performance trade-off parameter C and kernel width parameter σ, namely $[C_{\min}, C_{\max}]$, and $[\sigma_{\min}, \sigma_{\max}]$.

Output: The Lagrange multipliers $\alpha_1, \alpha_2, \ldots, \alpha_M$; The bias b.

1: Stack the tensors in training dataset together as $\mathcal{X}_{\text{trainStack}}$; stack the tensors in validation dataset together as $\mathcal{X}_{\text{validStack}}$.
2: Compute the TT approximation $TT(\mathcal{X}_{\text{trainStack}})$ with the given TT-ranks using TT-SVD, which produces $\mathcal{X}^{(i)}_{\text{trainStack}}, i = 1, \ldots, d$, and where the last TT-core $\mathcal{X}^{(d)}_{\text{trainStack}}$ has dimensions $R_d \times I_d \times M$.
3: Compute the TT approximation $TT(\mathcal{X}_{\text{validStack}})$, in which the first $d-1$ TT-core are same as the first $d-1$ cores in $TT(\mathcal{X}_{\text{trainStack}})$, and the last TT-core is computed with ~(23).
4: **for** C from $C_{\min}$ to $C_{\max}$ **do**
5: **for** σ from $\sigma_{\min}$ to $\sigma_{\max}$ **do**
6: Construct the K-STTM-Prod kernel matrix~(14) or K-STTM-sum kernel matrix~(15).
7: Solve~(16) using the resulting kernel matrix.
8: Compute the classification accuracy on validation set.
9: **end for**
10: **end for**
11: Find the best C and σ according to the classification accuracy on validation set.
12: Train the K-STTM with the best C and σ by implementing step 6 and 7. Thus the Lagrange multipliers $\alpha_1, \alpha_2, \ldots, \alpha_M$ and the bias b are obtained.

4.7. Convergence and complexity

In this section we discuss convergence of our proposed methods and compare the storage and computation complexity with the standard SVM.

For the convergence rate analysis, it is same as it in the standard SVM problem [23,24]. We already show the kernel validity of (14) and (15) in Theorem 1. With a valid kernel matrix, we can solve a quadratic programming problem to get the Lagrange multipliers α_i and bias b, which is same as the procedure in the standard SVM. Consequently, the convergence analysis is exactly same as it in standard SVM.

For the storage complexity analysis, the original tensorial sample storage is $O(MI^d)$, where I is the maximum value of $I_i, \ i = 1, 2, \ldots, d$. After representing the original tensorial data as TTs, the data storage becomes to $O(dIR^2 + MI_dR_d)$, where R is the maximum TT-rank of $I_i, i = 1, \ldots, d-1$. This shows a great reduction especially when the data order d is large.

The computational complexity of constructing the kernel matrix in standard SVM is $O(M^2I^d)$, where n is the maximum dimension of I_i, $i = 1, 2, \ldots d$. As for the computational complexity of K-STTM-Prod and K-STTM-Sum, the overall results of them are similar as we discussed in Section 4.5. When applying the above mentioned accelerating implementation of K-STTM-Sum, its kernel matrix computation complexity is $O(dIR^4 + M^2I_dR_d^2)$, where I and R are the maximum values of I_i and R_i, $i = 1, 2, \ldots d-1$, respectively. Therefore our proposed method is more efficient than its vector counterpart since the computation complexity is reduced from exponential to polynomial.

5. Experiments

We evaluate the effectiveness of the two proposed schemes, K-STTM-Prod and K-STTM-Sum, on real-world small-size tensorial datasets. We note that the classification model may not be well-trained with a very small number of training samples. However, we focus on the efficiency of the classification method itself. Therefore, data-level improvement skills, such as data augmentation, are not considered. We also do not consider the recently popular transfer learning and few-shot learning methods [12]. The reason is that they commonly need a pre-trained deep model for feature extraction or parameter initialization, and the pre-trained model is trained on a large number of data. To summarize, the compared methods we consider use only the given training data and are trained from scratch. We list the 7 compared methods as follows.

- **SVM**: SVM [4] is one of the most widely used vector-based method for classification. What is more, the proposed K-STTM is a tensorial extension of SVM, so SVM is selected as a baseline. We employ the widely used convex optimization solver CVX[1] to solve the quadratic programming problem.
- **STM**: STM [5] is the first method which extends SVM to the tensorial format, which employs alternating optimization scheme to update the weight tensors and outperforms kernel SVM in some tasks.
- **STuM**: STuM [15] is a linear support tensor machine and it is based on the Tucker decomposition. Its training procedure is similar to the one in STM.
- **STTM**: STTM [16] assumes the weight tensor is a scalable tensor train, which enables STTM to deal with high-dimensional data classification. STM, STuM, and STTM are all tensor-based linear classifiers. In the very small sample size problem, sometimes linear classifiers are observed to achieve a better classification accuracy than nonlinear classifier [17] since a linear classifier is commonly less complex and more stable, thus can be better trained than nonlinear classifiers.
- **DuSK**: DuSK [17] is a kernelized support tensor machine using the CP decomposition. Through introducing the kernel trick, it can deal with nonlinear classification tasks.
- **3D CNN**: CNN is one of the most powerful structure for image classification. The 3D CNN we employ here is an extension of the 2D version in Gupta et al. [25]. We replace the 2D convolutional kernels with 3D ones and keep other settings the same. Though 3D CNN is a relatively simple CNN model, it has an advantage in dealing with small sample size problems since it can be trained better than the complicated CNN model.
- **TT Classifier**: As an updated tensor classification method, TT classifier [26] trains a TT as a polynomial classifier and achieves good results on tensorial image classification tasks.

For simplicity, all of the kernel-based methods, i.e., SVM, DuSK, and K-STTM, employ the Gaussian RBF kernel. The optimal parameters, namely the performance trade-off parameter C, RBF kernel parameter σ, hidden layer size in 3D CNN, plus the corresponding tensor rank R_i in STuM, STTM, DuSK, TT classifier and K-STTM, are determined through a grid search. We provide a wide search range of C and σ such that it can fulfill the requirement for all methods. Specifically, we select C and σ from $\{10^{-6}, 10^{-5}, \ldots, 10^{8}, 10^{9}\}$. We select the hidden layer size of 3D-CNN from $\{10, 30, 50, 100, 150, 200\}$.

As for the tensor rank R_i, though STM is a tensor-based method, it assumes its weight tensor to be rank-one, thus there is no need to determine the tensor rank. STuM and the polynomial TT classifier assume the same rank over all tensor modes, namely $R_1 = R_2 = \ldots = R_d = R$. Therefore, the upper bound of the Tucker rank in STuM is limited to the smallest data dimension. In the first two fMRI datasets, the smallest dimensions are 8 and 23 respectively. Thus the tensor rank search range of STuM in the first two fMRI datasets is relatively smaller than other methods. As for the polynomial TT classifier, although it is based on the scalable TT structure, we could not set a high tensor rank since this increases the memory consumption a lot and makes the training speed extremely slow due to their model setting. We summarize the tensor rank search range of different methods on different datasets (details presented later) in Table 1. For more details about hyper-parameter search, we refer the reader to the supplementary material.

Due to the very small number of training samples, we implement a 5-fold cross-validation on all experiments. We repeat this process 50 times for all methods and report the average classification accuracy of each method for stable learning.

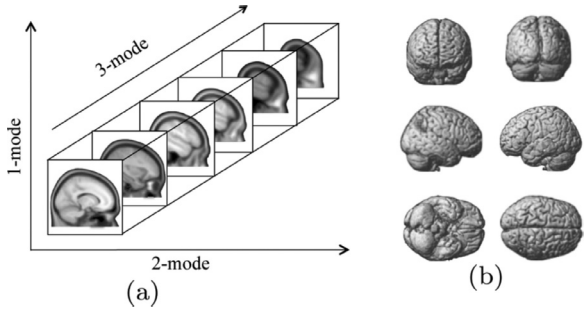


Fig. 5. fMRI images from [17]. (a) An illustration of a 3-way tensor (fMRI image), (b) Visualization of an fMRI image.

5.1. fMRI datasets

Here we consider three high-dimensional functional magnetic resonance imaging (fMRI) datasets, namely the StarPlus fMRI dataset,[2] the CMU Science 2008 fMRI dataset (CMU2008) [27] and the ADNI fMRI dataset[3] to evaluate the classification performance of different models. An fMRI image is essentially a 3-way tensor. Fig. 5 from [17] illustrates the tensorial structure of the fMRI image.

5.1.1. StarPlus fMRI dataset

The fMRI images in the StarPlus dataset are with dimensions $64 \times 64 \times 8$ that contains 25 to 30 anatomically defined regions (called "Regions of Interest", or ROIs). To achieve a better classification accuracy, we only consider the following ROIs: 'CALC' 'LIPL' 'LT' 'LTRIA' 'LOPER' 'LIPS' 'LDLPFC'. After extracting those ROIs, we further normalize the data of each subject. StarPlus fMRI dataset

[1] http://cvxr.com/cvx/

[2] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-81/www/

[3] http://adni.loni.usc.edu/

Table 1
Tensor ranks search range of different methods on different datasets.

Dataset	StarPlus	CMU2008	ANDI	Caltech-101
STuM	R: {2, 3, ..., 8}	R: {2, 3, ..., 23}	R: {10, 20, ..., 60}	R: {2, 3}
STTM	R_2: {60, 80, ..., 200}	R_2: {10, 20, ..., 100}	R_2: {10, 20, ..., 100}	R_2: {10, 20, ..., 100}
	R_3: {2, 4, 6, 8}	R_3: {2, 4, ..., 20}	R_3: {10, 20, ..., 60}	R_3: {3}
DuSK	R: {60, 80, ..., 200}	R: {10, 20, ..., 90}	R: {10, 20, ..., 100}	R: {10, 20, ..., 90}
TT classifier	R: {2, 3,..., 20 }	R: {2, 3,..., 20 }	R: {2, 3,..., 20 }	R: {2, 3,..., 20}
K-STTM- Prod/Sum	R_3: {60, 100, ..., 200}	R_2: {10, 20, ..., 100}	R_2: {10, 20, ..., 60}	R_2: {10, 20, ..., 100}
	R_2: {2, 4, 6, 8}	R_3: {2, 4, ..., 20}	R_3: {10, 20, ..., 100}	R_3: {3}

Table 2
Classification accuracy of different methods for different subjects in StarPlus fMRI datasets.

Subject	SVM	STM	STuM	STTM	DuSK	3D CNN	TT classifier	K-STTM-Prod	K-STTM-Sum
04799	50.00%	37.14%	34.97%	38.01%	47.88%	51.79%	56.13%	**68.28%**	66.55%
04820	50.00%	42.97%	32.66%	46.55%	45.65%	44.57%	53.81%	**70.65%**	64.87%
04847	50.00%	38.30%	17.71%	48.40%	53.99%	54.69%	60.48%	65.46%	**66.02%**
05675	50.00%	38.56%	29.78%	34.38%	56.58%	48.43%	54.98%	**60.13%**	59.57%
05680	50.00%	37.49%	39.39%	41.64%	62.95%	68.74%	60.21%	72.01%	**76.25%**
05710	50.00%	39.11%	31.54%	44.45%	55.63%	48.70%	54.12%	**58.31%**	58.13%

contains the brain images of 6 human subjects. The data of each human subject is partitioned into trials, and each subject has 40 effective trials. Here we only use the first 4 s of each trial since the subject was shown one kind of stimulus (sentence or picture) during the whole period. The fMRI images were collected every 500 ms, thus we can utilize 8 fMRI images in each trial. Overall, we have 320 fMRI images: one half of them were collected when the subject was shown a picture, the other half were collected when the subject was shown a sentence.

The classification results are listed in Table 2. Due to the very high-dimensional and sparse data, SVM fails to find a good hyperparameter setting and classifies all test samples into one class, thus can not do classification. Since fMRI data are very complicated, those linear classifiers, namely STM, STuM and STTM, can not achieve acceptable performance, and the classification accuracies of them are all lower than 50%. The classification result of TT classifier is poor on several subjects. DuSK also performs poorly on subjects '04799' and '04820'. Due to the small number of training samples and high-dimensional data size, the 3D CNN overfits and can not be well trained, while our proposed two methods still achieve the highest classification accuracy on all human subjects.

5.1.2. CMU2008

The second fMRI dataset we consider is CMU2008. It shows the brain activities associated with the meanings of nouns. During the data collection period, the subjects were asked to view 60 different word-picture from 12 semantic categories. There are 5 pictures in each category and each image is shown to the subject for 6 times. Therefore, we can get 30 fMRI images for each semantic category, and each fMRI image is with dimensions $51 \times 61 \times 23$. In this experiment, we consider all the ROIs thus the classified fMRI images are relatively denser than the images we classified in the StarPlus example. Considering the extremely small number of samples in each category, we therefore follow the experiment settings in Kampa et al. [28], which combines two similar categories into an integrated class. Specifically, we combine categories *animal* and *insect* as class **Animals**, and categories *tool* and *furniture* as class **Tools**. By doing so, we have 60 samples in both **Animals** and **Tools** classes.

Table 3 shows the binary classification results of different models. We notice that SVM can perform classification on this dataset since we include all ROIs, which facilitates the hyperparameter searching procedure. However, its classification accuracies on two subjects are lower than 50%. The linear and polynomial models, namely STM, STuM, STTM, and TT classifier, can only achieve acceptable performance on a few subjects. Due to the high-dimensional data size, DuSK fails to find a good CP-rank in acceptable time and can not achieve a good classification accuracy. 3D CNN still performs poorly due to the very few training samples and high-dimensional feature size. Our proposed two methods still achieve the best classification results on all subjects.

5.1.3. ANDI fMRI dataset

ANDI fMRI dataset is collected from the Alzheimer's Disease Neuroimaging Initiative. It contains the resting-state fMRI images of 33 subjects. The subject includes patients (with Mild Cognitive Impairment (MCI) or Alzheimer's Disease (AD)) and normal controls. Overall we have 33 fMRI images and each image is with dimensions $61 \times 73 \times 61$. We further separate them into two classes. The positive class includes normal controls, while the negative class includes patients with MCI or AD. Since the number of samples is very small, we run all the experiment 50 times and the results are reported by averaging the accuracy over these runs.

Table 4 lists the classification results of the seven compared methods and the proposed K-STTM-Prod/Sum. We can observe that the performance of SVM, STM, STuM and TT classifier is still not good. STTM and DuSK achieve slightly better performance than random classification. The proposed K-STTM-Prod/Sum achieves the best accuracy over all compared methods.

5.2. Caltech-101 binary-classification

In this experiment, we use the Caltech-101 dataset [29] to investigate the fourth claim in Section 4.2, namely, we can perform different kernel functions on different tensor modes. Caltech-101 is an image dataset, which includes 101 object categories. However, the number of images in each category differs a lot, about 40 to 800 images per category. We note that this paper cares more about small sample size classification problems. We therefore select 5 class pairs to implement binary-classification experiments and each category includes 50 colorful images. Since the size of each image differs also, we resize all the images into $200 \times 300 \times 3$. We note that each color image is naturally a three-way tensor (pixel-pixel-color), and the first two tensor modes are related to pixel intensity, therefore we utilize the same Gaussian RBF kernel for the first two tensor modes and try a different kernel (linear or polynomial) for the third mode when implementing the classification with the proposed K-STTM-Prod/Sum. The parameters c, d in the polynomial kernel $\boldsymbol{k}(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y} + c)^d$ were empirically set to

Table 3
Classification accuracy of different methods for different subjects in CMU2008 fMRI datasets.

Subject	SVM	STM	STuM	STTM	DuSK	3D CNN	TT classifier	K-STTM-Prod	K-STTM-Sum
#1	65.55%	66.30%	67.31%	64.93%	45.77%	49.76%	29.40%	69.06%	**71.54**%
#2	52.35%	50.88%	58.77%	68.62%	55.48%	57.48%	43.60%	75.41%	**83.72**%
#3	50.24%	60.40%	57.90%	65.05%	58.09%	55.33%	61.13%	66.43%	**68.98**%
#4	50.62%	59.79%	58.29%	56.31%	53.26%	58.05%	56.38%	**76.11**%	70.62%
#5	56.72%	59.01%	65.10%	66.04%	44.82%	58.11%	56.55%	**72.62**%	72.17%
#6	43.57%	59.03%	46.27%	45.78%	53.86%	55.70%	49.81%	**69.30**%	67.65%
#7	49.75%	52.26%	48.83%	51.66%	50.81%	60.61%	59.84%	67.40%	**71.56**%

Table 4
Classification accuracy of different methods in ADNI fMRI dataset.

SVM	STM	STuM	STTM	DuSK	3D CNN	TT classifier	K-STTM-Prod	K-STTM-Sum
50.63%	57.78%	42.43%	55.90%	50.37%	60.08%	51.25%	**71.03**%	64.83%

Table 5
Classification accuracy of SVM, STM, STuM, STTM, DuSK, 3D CNN, TT classiifer for different Caltech-101 class pairs.

Class pair	SVM	STM	STuM	STTM	DuSK	3D CNN	TT classifier
brain, cup	**70.74**%	50.00%	70.26%	67.47%	70.05%	42.39%	45.28%
brain, soccer_ball	50.04%	50.00%	35.36%	42.35%	50.65%	**54.62**%	51.18%
butterfly, watch	82.76%	50.00%	57.43%	75.81%	84.34%	**89.78**%	42.58%
cup, soccer_ball	61.90%	50.00%	**64.65**%	60.19%	54.33%	56.69%	49.54%
soccer_ball, umbrella	60.80%	50.00%	39.57%	57.16%	**63.19**%	58.19%	47.17%

Table 6
Classification accuracy of K-STTM-Prod and K-STTM-Sum with different kernel functions for different Caltech-101 class pairs.

Class pair	K-STTM-Prod			K-STTM-Sum		
	RBF-RBF-RBF	RBF-RBF-Poly	RBF-RBF-Linear	RBF-RBF-RBF	RBF-RBF-Poly	RBF-RBF-Linear
brain, cup	81.00%	67.24%	**91.22**%	77.72%	78.49%	79.13%
brain, soccer_ball	71.35%	65.92%	**81.04**%	72.41%	77.49%	77.20%
butterfly, watch	89.94%	87.85%	86.63%	90.28%	90.96%	**91.75**%
cup, soccer_ball	69.89%	69.10%	75.69%	72.52%	77.01%	**80.82**%
soccer_ball, umbrella	72.09%	67.15%	75.27%	**83.74**%	72.48%	82.86%

$c = 1$ and $d = 2$. The baseline case is when the Gaussian RBF kernel is applied to all tensor modes.

Tables 5 and 6 lists the classification results of the seven compared methods and the proposed K-STTM-Prod/Sum respectively. We observe that K-STTM-Prod/Sum almost achieves the best accuracy on all class pairs. And by applying a linear kernel on the color mode, the classification accuracy of K-STTM-Prod/Sum achieves similar or better performance than the baseline case (RBF-RBF-RBF) on all class pairs, which demonstrates the potential benefit of employing different kernel functions on different tensor modes when they contain different kind of information. As for applying the polynomial kernel on the color mode, the classification accuracy decreases a little compared with RBF-RBF-RBF case in K-STTM-Prod. This indicates that RBF-RBF-Linear may be a better setting when classifying color images.

Apart from the classification accuracy, we further investigate the kernel matrix construction time of three kernel-based methods, namely SVM, DuSK and K-STTM, to show the efficiency of the proposed accelerating implementation (i.e., Section 4.5) of constructing kernel matrix for K-STTM. For fair comparison, we set the CP-rank as 30 for DuSK, and TT-rank as $R_2 = 10$ and $R_3 = 3$ for K-STTM. By doing so, the overall computation of kernel matrix construction for DuSK and K-STTM is similar, which can be observed by comparing their kernel matrix construction equations. Moreover, we also compare the time consumption for tensorial data preparation in DuSK and K-STTM, in which DuSK decomposes the tensorial data into their CP format sample by sample, while K-STTM stacks those data together and decomposes them into their TT format in one shot.

Table 7
Time consumption with respect to kernel matrix construction and tensorial data preparation.

Time consumption (s)	SVM	DuSK	K-STTM-Prod/Sum
tensorial data preparation	–	1517.04	1.16/1.15
kernel matrix construction	6.62	7.86	0.21/0.28

Table 7 lists the result of time consumption with respect to kernel matrix construction and tensorial data preparation in binary classification for *brain* and *cup*. K-STTM-Prod and K-STTM-Sum cost a similar time on the two metrics. We notice that DuSK consumes 1300× more time on tensorial data preparation than the proposed K-STTM-Prod/Sum. What is more, the proposed K-STTM-Prod/Sum costs 23× and 28× less time on kernel construction than SVM and DuSK, respectively. This indicates the high efficiency of K-STTM.

6. Conclusions and future works

This paper has proposed a tensor train (TT)-based kernel trick and devised a kernelized support tensor train machine (K-STTM). Extensive experiments have demonstrated the superiority of K-STTM for tensorial data classification in few-sample size scenarios. By employing TT structure, we reduce the storage and computation complexity from exponential to polynomial. Theoretical proof has been given to demonstrate the validity of the proposed tensor-based kernel mapping scheme for the first time. Moreover, as a common issue in tensor-based kernel learning, the non-uniqueness of tensor decomposition is well explained and addressed herein.

Applying different kernel functions on different tensor modes is also investigated empirically, and we observe a consistent improvement compared with the baselines in which all modes employ the same kernel function.

We further envision three future research directions based on the K-STTM framework. Firstly, in our paper, we treat each fMRI image as an independent sample. However, those fMRI images are sampled in a continuous time period and therefore they have some particular correlation on the time axis. In our future work, we would investigate the short-time series analysis on fMRI images and address the possible time bias issue in it. Secondly, instead of constructing a kernel matrix in the K-STTM formula, we will consider building a kernel tensor. We believe that the kernel matrix constructed for each mode can contain different information. Simply multiplying or adding this information may not be the best solution. Subsequently, we propose to stack this information into a 3-way kernel tensor and develop a better way to exploit information in each of the modes. Thirdly, we will embed the proposed kernel mapping trick into other kernel-based methods such as LSSVM [30], kernel PCA [31] etc., such that these methods can directly deal with tensorial data and achieve potentially better performance.

Declaration of Competing Interest

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

Acknowledgments

This work is partially supported by the Hong Kong Research Grants Council under Project 17246416, the University Research Committee of The University of Hong Kong, Tsinghua University Initiative Scientific Research Program, and NSFC under grant no. 61872206.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.patcog.2021.108337.

References

[1] C. Dou, S. Zhang, H. Wang, L. Sun, Y. Huang, W. Yue, ADHD fMRI short-time analysis method for edge computing based on multi-instance learning, J. Syst. Archit. 111 (2020) 101834.
[2] G. Luca, P. Gennaro, R. Pierluigi, T. Francesco, V. Mario, Trends in IoT based solutions for health care: moving AI to the edge, Pattern Recognit. Lett. 135 (2020) 346–353.
[3] G. Dai, D.-Y. Yeung, Tensor embedding methods, Association for the Advancement of Artificial Intelligence (2006) 330–335.
[4] B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in: The Fifth Annual Workshop on Computational Learning Theory, 1992, pp. 144–152.
[5] D. Tao, X. Li, W. Hu, S. Maybank, X. Wu, Supervised tensor learning, in: IEEE International Conference on Data Mining, 2005, p. 8.
[6] T.D. Nguyen, T. Tran, D. Phung, S. Venkatesh, Tensor-variate restricted Boltzmann machines, Association for the Advancement of Artificial Intelligence, 2015.
[7] B. Zongwen, L. Ying, W. Marcin, Z. Meili, L. Di, Decomvqanet: decomposing visual question answering deep network via tensor decomposition and regression, Pattern Recognit. 110 (2021) 107538.
[8] M. Yang, W. Ping, L. Liangfu, Z. Xuyun, Q. Lianyong, Weighted tensor nuclear norm minimization for tensor completion using tensor-SVD, Pattern Recognit. Lett. (2020) 4–11.
[9] A. Soheil, R. Mansoor, Generalized low-rank approximation of matrices based on multiple transformation pairs, Pattern Recognit. 108 (2020) 107545.
[10] L. Jiani, Z. Ce, L. Zhen, H. Huyan, L. Yipeng, Low-rank tensor ring learning for multi-linear regression, Pattern Recognit. 113 (2021) 107753.
[11] K. Piotr, W. Lei, C. Anoop, Tensor representations for action recognition, IEEE Trans. Pattern Anal. Mach. Intell. (2021) 1.
[12] A. Riaz, M. Asad, S.M.R. Al Arif, E. Alonso, D. Dima, P. Corr, G. Slabaugh, Deep fMRI: an end-to-end deep network for classification of fMRI data, in: IEEE International Symposium on Biomedical Imaging, 2018, pp. 1419–1422.
[13] I.V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput. 33 (5) (2011) 2295–2317.
[14] I. Kotsia, W. Guo, I. Patras, Higher rank support tensor machines for visual recognition, Pattern Recognit. 45 (12) (2012) 4192–4203.
[15] I. Kotsia, I. Patras, Support Tucker machines, in: IEEE Conference on Computer Vision and Pattern Recognition, 2011, pp. 633–640.
[16] C. Chen, K. Batselier, C.-Y. Ko, N. Wong, A support tensor train machine, in: International Joint Conference on Neural Networks, 2019, pp. 1–8.
[17] L. He, X. Kong, P.S. Yu, X. Yang, A.B. Ragin, Z. Hao, Dusk: a dual structure-preserving kernel for supervised tensor learning with applications to neuroimages, in: SIAM International Conference on Data Mining, 2014, pp. 127–135.
[18] L. He, C.-T. Lu, H. Ding, S. Wang, L. Shen, P.S. Yu, A.B. Ragin, Multi-way multi-level kernel modeling for neuroimaging classification, in: IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 356–364.
[19] L. He, C.-T. Lu, G. Ma, S. Wang, L. Shen, P.S. Yu, A.B. Ragin, Kernelized support tensor machines, in: International Conference on Machine Learning, 2017, pp. 1442–1451.
[20] R. Orús, A practical introduction to tensor networks: matrix product states and projected entangled pair states, Ann. Phys. 349 (2014) 117–158.
[21] L. Houthuys, J.A.K. Suykens, Tensor learning in multi-view kernel PCA, in: International Conference on Artificial Neural Networks, 2018, pp. 205–215.
[22] J. Schur, Bemerkungen zur theorie der beschränkten bilinearformen mit unendlich vielen veränderlichen, J. Reine Angew. Math. 1911 (140) (1911) 1–28.
[23] I. Steinwart, Support vector machines are universally consistent, J. Complex. 18 (3) (2002) 768–791.
[24] I. Steinwart, C. Scovel, et al., Fast rates for support vector machines using Gaussian kernels, Ann. Stat. 35 (2) (2007) 575–607.
[25] A. Gupta, M. Ayhan, A. Maida, Natural image bases to represent neuroimaging data, in: International Conference on Machine Learning, 2013, pp. 987–994.
[26] Z. Chen, K. Batselier, J.A. Suykens, N. Wong, Parallelized tensor train learning of polynomial classifiers, IEEE Trans. Neural Netw. Learn. Syst. 29 (10) (2017) 4621–4632.
[27] T.M. Mitchell, S.V. Shinkareva, A. Carlson, K.-M. Chang, V.L. Malave, R.A. Mason, M.A. Just, Predicting human brain activity associated with the meanings of nouns, Science 320 (5880) (2008) 1191–1195.
[28] K. Kampa, S. Mehta, C.-A. Chou, W.A. Chaovalitwongse, T.J. Grabowski, Sparse optimization in feature selection: application in neuroimaging, J. Global Optim. 59 (2–3) (2014) 439–457.
[29] L. Fei-Fei, R. Fergus, P. Perona, Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories, in: IEEE Conference on Computer Vision and Pattern Recognition Workshop, 2004, p. 178.
[30] J.A. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Process. Lett. 9 (3) (1999) 293–300.
[31] B. Schölkopf, A. Smola, K.-R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, Neural Comput. 10 (5) (1998) 1299–1319.

Cong Chen received the M.S. degree in Electrical and Electronic Engineering from The University of Hong Kong, Hong Kong, in 2018, where he is currently pursuing the Ph.D. degree. His current research interests include tensor computation, image completion and model compression.

Kim Batselier received the M.S. degree in Electro-Mechanical Engineering and the Ph.D. Degree in Engineering Science from the KULeuven, Belgium, in 2005 and 2013 respectively. He is currently an Assistant Professor at Delft University of Technology. His current research interests include linear and nonlinear system theory/identification, algebraic geometry, tensors, and numerical algorithms.

Wenjian Yu received the B.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 1999 and 2003, respectively. He joined Tsinghua University, in 2003, where he is an Associate Professor with the Department of Computer Science and Technology. His current research interests include modeling and simulation of complex systems (integrated circuits and others), numerical algorithms, and their applications.

Ngai Wong received his B.Eng. and Ph.D. degrees in Electrical and Electronic Engineering from The University of Hong Kong, Hong Kong, in 1999 and 2003, respectively. He is currently an Associate Professor with the Department of Electrical and Electronic Engineering, The University of Hong Kong. His current research interests include linear and nonlinear circuit modeling and simulation, model order reduction, passivity test and enforcement, and tensor-based numerical algorithms in electronic design automation (EDA).