CHAPTER

Tensor network algorithms for image classification

8

Cong Chen[a], **Kim Batselier**[b], **and Ngai Wong**[a]
[a]*Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong*
[b]*Delft Center for Systems and Control, Delft University of Technology, Delft, the Netherlands*

CONTENTS

8.1 Introduction

Classification algorithm design has been an important topic in machine learning, pattern recognition, and computer vision for decades. Support vector machine (SVM) [1] and logistic regression (LR) [2] are two representative and popular classifiers, which achieve enormous success in pattern classification. However, standard SVM and LR models are based on vector inputs and cannot directly deal with matrices or higher-dimensional data structures (namely, tensors) which are very common in real-life applications. For example, a grayscale picture is stored as a matrix, which is a second-order tensor, while color pictures have a color axis and are naturally third-order

Tensors for Data Processing. https://doi.org/10.1016/B978-0-12-824447-0.00014-5

tensors. Therefore, tensorial data need to be vectorized before being fed into SVM or LR. However, this breaks the original data structure information and results in loss of the spatial relationship of nearby pixels [8], which may lead to poor classification performance. Moreover, this vectorization can easily result in a very high-dimensional vector, which results in the notorious curse of dimensionality problem. Consequently, when the number of training samples is relatively small compared to the feature vector dimension, the classifier may easily produce poor classification performance due to overfitting [5–7]. To alleviate overfitting, researchers have attempted to add sparseness constraints during the classifier training procedure. For example, [44] proposed a Bayesian LR approach to solve the overfitting problem. Specifically, this method assumes a prior probability distribution that favors sparseness in the trained classifier. However, the useful tensorial data structure is still disregarded. To solve the above two issues, researchers have focused on extending the traditional SVM and LR into their tensor-formatted counterparts. By doing so, the tensorial versions take the underlying data structural prior into account and at the same time reduce the number of model parameters. In fact, it can be observed that the tensor-based approaches generally outperform the traditional ones [4,29].

In terms of SVM, Ref. [4] proposes a supervised tensor learning (STL) scheme by replacing the vector inputs with tensor inputs and decomposing the corresponding weight vector into a rank-1 tensor, which is trained by the alternating projection optimization method. Based on this learning scheme, [4] extends the standard linear SVM to a general tensor form called the support tensor machine (STM). Although STM lifts the overfitting problem in traditional SVMs, the expressive power of a rank-1 weight tensor is limited, which may not be powerful enough to classify complicated data. In [15], the rank-1 weight tensor of STM is generalized to canonical polyadic (CP) forms for stronger model expressive power. However, the determination of a good CP rank is NP-complete [3]. In [29], an STM is generalized to a support Tucker machine (STuM), which replaces the rank-1 tensor in STM with a Tucker-decomposed tensor. Nevertheless, the number of parameters in the Tucker form is exponentially large, which still suffers from the curse of dimensionality. To alleviate the issue in STuM, [31] proposed to reformulate SVM with a more scalable tensor network called the tensor train (TT). The aforementioned tensorial extensions of SVM 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. Therefore, [37] further proposed the kernelized support TT machine (K-STTM), which employs the customized kernel functions for the TT structure in order to handle nonlinear tensorial classification problems.

As for LR, the tensorial extensions are similar to the cases in SVM. Ref. [38] replaces the weight vector into a rank-1 tensor, and the authors argue that this setting retains the structure of the spatial and spectral band information, which is a very important aspect for hyperspectral data classification. However, this method suffers from the same problems as in an STM. A more generalized tensor-based LR is proposed by [39], in which the weight vector in LR is replaced by a CP-decomposed tensor.

The rest of this chapter is organized as follows. Some background knowledge of this chapter is introduced in Section 8.2. In Section 8.3, the supervised learning scheme and different tensor-based SVM versions are demonstrated in detail. The tensor-based LR models are briefly introduced in Section 8.4, which are quite similar to the extension schemes in SVM. Lastly, the conclusion is drawn in Section 8.5.

8.2 Background

In this section, some basic tensor computation operations and decomposition methods are introduced. The ideas of traditional SVM and LR are also succinctly reviewed.

8.2.1 Tensor basics

Tensors are multidimensional arrays that are higher-order generalizations of vectors (first-order tensors) and matrices (second-order tensors). A d-th-order or d-way tensor is denoted as $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and the element of $\boldsymbol{\mathcal{A}}$ by $\boldsymbol{\mathcal{A}}(i_1, i_2, \ldots, i_d)$, 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 $\boldsymbol{\mathcal{A}}$. We use boldface capital calligraphic letters $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$, ... 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. $\boldsymbol{A}^T$ and $\boldsymbol{a}^T$ are the transpose of a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{a}$, respectively. The unit matrix of order n is denoted $\boldsymbol{I}_n$. An intuitive and useful graphical representation of scalars, vectors, matrices, and tensors is depicted in Fig. 8.1. The edges, also called free legs, are the indices of the array. Therefore scalars have no free edge, while matrices have two free edges. We will mainly employ these graphical representations to visualize the tensor networks and operations in the following sections whenever possible and refer to [18] for more details. We now briefly introduce some important tensor operations.

Definition 8.1. (Tensor k-mode product) The k-mode product of a d-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_k \times \cdots \times I_d}$ with a matrix $\boldsymbol{U} \in \mathbb{R}^{P_k \times I_k}$ is denoted as $\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{A}} \times_k \boldsymbol{U}$ and defined by

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

where $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_{k-1} \times P_k \times I_{k+1} \times \cdots \times I_d}$.

The graphical representation of a three-mode product between a third-order tensor $\boldsymbol{\mathcal{A}}$ and a matrix $\boldsymbol{U}$ is shown in Fig. 8.2, where the summation over the i_3 index is indicated by the connected edge.

Definition 8.2. (Matricization) The matricization (also known as unfolding or flattening of a tensor) is the reordering of the tensor elements into a matrix.

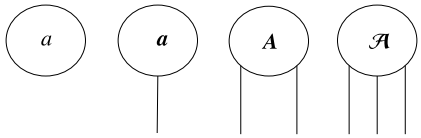

FIGURE 8.1

Graphical representation of a scalar a, vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$, and third-order tensor $\boldsymbol{\mathcal{A}}$.

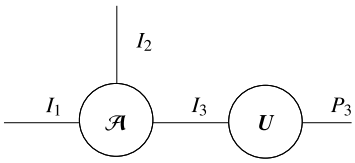

FIGURE 8.2

Three-mode product between a three-way tensor $\boldsymbol{\mathcal{A}}$ and matrix $\boldsymbol{U}$.

The k-th mode matricization of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_k \times \cdots \times I_d}$, denoted by $\boldsymbol{A}_{(k)} \in \mathbb{R}^{I_k \times (\prod_{k \neq i} I_i)}$, arranges the k-th mode fibers to become the columns of the final matrix. The element $(i_1, i_2, \ldots, i_d)$ in tensor $\boldsymbol{\mathcal{A}}$ is mapped to the matrix element (i_k, j), where

$$j = 1 + \sum_{t=1, t \neq k}^{d} (i_t - 1) J_t \;\; with \;\; J_t = \prod_{l=1, l \neq k}^{t-1} I_l.$$

Definition 8.3. (Reshaping) Employing MATLAB® notation, "reshape($\boldsymbol{\mathcal{A}}$, [J_1, J_2, …, J_d])" reshapes the tensor $\boldsymbol{\mathcal{A}}$ into another tensor with dimensions $J_1, J_2, \ldots, J_d$. The total number of elements of the tensor $\boldsymbol{\mathcal{A}}$ must be $\prod_{k=1}^{d} J_k$.

Definition 8.4. (Vectorization) Vectorization is a special reshaping operation that reshapes a tensor $\boldsymbol{\mathcal{A}}$ into a column vector, denoted as $vec(\boldsymbol{\mathcal{A}})$.

Definition 8.5. (Tensor inner product) For two tensors $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, their inner product $\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \rangle$ is defined as

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

Definition 8.6. (Frobenius norm) The Frobenius norm of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is defined as $||\boldsymbol{\mathcal{A}}||_F = \sqrt{\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}} \rangle}$.

8.2.2 Tensor decompositions

Here we introduce four related tensor decomposition methods in this chapter, namely rank-1 tensor decomposition, CP decomposition, Tucker decomposition, and TT decomposition.

8.2.2.1 Rank-1 tensor decomposition

A d-way tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is rank-1 if it can be written as the outer product of d vectors

$$\mathcal{A} = \boldsymbol{u}^{(1)} \circ \boldsymbol{u}^{(2)} \circ \cdots \circ \boldsymbol{u}^{(d)}, \tag{8.1}$$

where $\circ$ denotes the vector outer product and each element in $\mathcal{A}$ is the product of the corresponding vector elements:

$$\mathcal{A}(i_1, \ldots, i_d) = \boldsymbol{u}^{(1)}(i_1)\boldsymbol{u}^{(2)}(i_2) \cdots \boldsymbol{u}^{(d)}(i_d).$$

Storing the component vectors $\boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(d)}$ instead of the whole tensor $\mathcal{A}$ significantly reduces the required number of storage elements. However, a rank-1 tensor is rare in real-world applications, so that a rank-1 approximation to a general tensor usually results in unacceptably large approximation errors. This calls for a more general and powerful tensor approximation, for which the following CP decomposition serves as a more suitable choice.

8.2.2.2 Canonical polyadic decomposition

The CP decomposition factorizes a tensor into a sum of constituent rank-1 tensors. For example, a d-way tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ can be written as

$$\mathcal{A} = \sum_{r=1}^{R} \boldsymbol{u}_r^{(1)} \circ \boldsymbol{u}_r^{(2)} \circ \cdots \circ \boldsymbol{u}_r^{(d)}, \tag{8.2}$$

where R is a positive integer and $\boldsymbol{u}_r^{(1)} \in \mathbb{R}^{I_1}$, $\boldsymbol{u}_r^{(2)} \in \mathbb{R}^{I_2}$, ..., $\boldsymbol{u}_r^{(d)} \in \mathbb{R}^{I_d}$, for $r = 1, \ldots, R$. If the mode j components $\boldsymbol{u}_r^{(j)}$, $r = 1, \ldots, R$, are stacked together in a matrix $\boldsymbol{U}^{(j)}$ as

$$\boldsymbol{U}^{(j)} = \left[\boldsymbol{u}_1^{(j)}, \boldsymbol{u}_2^{(j)}, \ldots, \boldsymbol{u}_R^{(j)}\right], \quad j = 1, \ldots, d, \tag{8.3}$$

then the j-th mode matricization of $\mathcal{A}$ can be computed as

$$\boldsymbol{A}_{(j)} = \boldsymbol{U}^{(j)}\left[\boldsymbol{U}^{(d)} \odot \cdots \odot \boldsymbol{U}^{(j+1)} \odot \boldsymbol{U}^{(j-1)} \cdots \odot \boldsymbol{U}^{(1)}\right]^T. \tag{8.4}$$

If we define

$$\boldsymbol{U}^{(-j)} = \boldsymbol{U}^{(d)} \odot \cdots \odot \boldsymbol{U}^{(j+1)} \odot \boldsymbol{U}^{(j-1)} \cdots \odot \boldsymbol{U}^{(1)}, \tag{8.5}$$

Eq. (8.4) can be written as

$$\boldsymbol{A}_{(j)} = \boldsymbol{U}^{(j)}\big[\boldsymbol{U}^{(-j)}\big]^T. \tag{8.6}$$

The inner product between $\boldsymbol{\mathcal{A}}$ and itself can then be written as

$$\langle \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{A}} \rangle = Tr\big[\boldsymbol{A}_{(j)}\boldsymbol{A}_{(j)}^T\big] = vec(\boldsymbol{A}_{(j)})^T vec(\boldsymbol{A}_{(j)}), \tag{8.7}$$

where $Tr[\cdot]$ denotes the trace of a matrix.

8.2.2.3 Tucker decomposition

The Tucker decomposition represents a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ as

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}} \times_1 \boldsymbol{P}^{(1)} \times_2 \boldsymbol{P}^{(2)} \cdots \times_d \boldsymbol{P}^{(d)}, \tag{8.8}$$

where $\boldsymbol{\mathcal{G}}$ is the Tucker core tensor and $\boldsymbol{P}^{(1)}, \ldots, \boldsymbol{P}^{(d)}$ are a set of (factor) matrices that are multiplied to the core tensor $\boldsymbol{\mathcal{G}}$ along each tensor mode. The Kronecker product of all d matrices is defined as

$$\boldsymbol{P}_{\otimes} = \boldsymbol{P}^{(d)} \otimes \cdots \otimes \boldsymbol{P}^{(1)}, \tag{8.9}$$

while the Kronecker product of $d-1$ matrices is defined as

$$\boldsymbol{P}_{\otimes}^{(-j)} = \boldsymbol{P}^{(d)} \otimes \cdots \otimes \boldsymbol{P}^{(j+1)} \otimes \boldsymbol{P}^{(j-1)} \otimes \cdots \otimes \boldsymbol{P}^{(1)}. \tag{8.10}$$

Therefore, the j-th mode matricization of $\boldsymbol{\mathcal{A}}$ is

$$\begin{aligned} \boldsymbol{A}_{(j)} &= \boldsymbol{P}^{(j)}\boldsymbol{G}_{(j)}\big(\boldsymbol{P}^{(d)} \otimes \cdots \otimes \boldsymbol{P}^{(j+1)} \otimes \boldsymbol{P}^{(j-1)} \otimes \cdots \otimes \boldsymbol{P}^{(1)}\big)^T \\ &= \boldsymbol{P}^{(j)}\boldsymbol{G}_{(j)}\big(\boldsymbol{P}_{\otimes}^{(-j)}\big)^T. \end{aligned} \tag{8.11}$$

Eq. (8.11) can be further rewritten into the vectorized version when $j = 1$,

$$vec(\boldsymbol{A}_{(1)}) = \boldsymbol{P}_{\otimes} vec(\boldsymbol{G}_{(1)}). \tag{8.12}$$

8.2.2.4 Tensor train decomposition

A TT decomposition [33] represents a d-way tensor $\boldsymbol{\mathcal{A}}$ as d third-order tensors $\boldsymbol{\mathcal{A}}^{(1)}$, $\boldsymbol{\mathcal{A}}^{(2)}$, $\ldots$, $\boldsymbol{\mathcal{A}}^{(d)}$ such that a particular entry of $\boldsymbol{\mathcal{A}}$ is written as the following matrix product:

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d) = \boldsymbol{\mathcal{A}}^{(1)}(:, i_1, :) \cdots \boldsymbol{\mathcal{A}}^{(d)}(:, i_d, :). \tag{8.13}$$

Each tensor $\boldsymbol{\mathcal{A}}^{(k)}$, $k = 1, \ldots, d$, 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 (8.13) 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 three-way tensor $\boldsymbol{\mathcal{A}}$ is shown

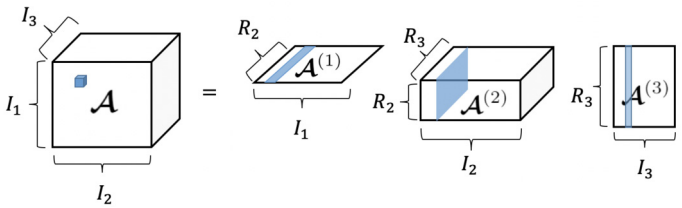

FIGURE 8.3

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

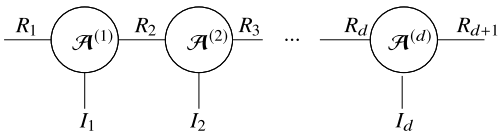

FIGURE 8.4

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

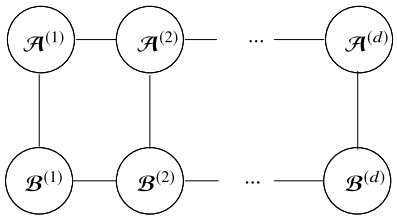

FIGURE 8.5

The inner product between two d-way tensor trains.

in Fig. 8.3. 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. 8.4 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 (8.13). The simplifying notation $TT(\mathcal{A})$ denotes a TT decomposition of a d-way tensor $\mathcal{A}$ with user-specified TT ranks of (8.13).

Definition 8.7. (TT inner product) The inner product between two TTs $TT(\mathcal{A})$ 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. 8.5. The absence of free edges in Fig. 8.5 implies that $\langle TT(\mathcal{A}), TT(\mathcal{B})\rangle$ is a scalar.

Definition 8.8. (Left orthogonal and right orthogonal TT cores) A TT core $\mathcal{A}^{(k)}(1 \leq k \leq d)$ is left orthogonal when reshaped into an $R_k I_k \times R_{k+1}$ matrix $\boldsymbol{A}$ we have

$$\boldsymbol{A}^T \boldsymbol{A} = \boldsymbol{I}_{R_{k+1}}.$$

Similarly, a TT core $\mathcal{A}^{(k)}$ is right orthogonal when reshaped into an $R_k \times I_k R_{k+1}$ matrix $\boldsymbol{A}$ we have

$$\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I}_{R_k}.$$

Those two properties facilitate the computation of the Frobenius norm of a TT format tensor, as we will show later.

Definition 8.9. (Site-k-mixed-canonical tensor train) A TT is in site-k-mixed-canonical form [20] when all TT cores $\{\mathcal{A}^{(l)} \mid l = 1, \ldots, k-1\}$ are left orthogonal and $\{\mathcal{A}^{(l)} \mid l = k+1, \ldots, d\}$ are right orthogonal.

Turning a TT into its site-k-mixed-canonical form requires $d-1$ QR decompositions of the reshaped TT cores. Changing k in a site-k-mixed-canonical form to either $k-1$ or $k+1$ requires one QR factorization of $\mathcal{A}^{(k)}$. It can be shown that the Frobenius norm of a tensor $\mathcal{A}$ in a site-k-mixed-canonical form is easily computed from

$$||\mathcal{A}||_F^2 = ||\mathcal{A}^{(k)}||_F^2 = vec\big(\mathcal{A}^{(k)}\big)^T vec\big(\mathcal{A}^{(k)}\big).$$

8.2.3 Support vector machines

We briefly introduce linear SVMs before discussing STMs. 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 samples or feature vectors with labels $y_i \in \{-1, 1\}$. Learning a linear SVM is to find a discriminant hyperplane

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b \tag{8.14}$$

that maximizes the margin between the two classes where $\boldsymbol{w}$ and b are the weight vector and bias, respectively. In practice, the data are seldom linearly separable due to measurement noise. A more robust classifier can then be found by introducing the slack variables $\xi_1, \ldots, \xi_M$ and writing the learning problem as an optimization problem

$$\begin{aligned} \min_{\boldsymbol{w},b,\xi} \quad & \frac{1}{2}||\boldsymbol{w}||_F^2 + C\sum_{i=1}^{M}\xi_i \\ \text{subject to} \quad & y_i\big(\boldsymbol{w}^T\boldsymbol{x}_i + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, M. \end{aligned} \tag{8.15}$$

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 common to solve the dual problem of (8.15) with quadratic programming, especially when the feature size n is larger than the sample size M. The dual problem of (8.15) is

$$\begin{aligned} \min_{\alpha_1,\alpha_2,\cdots,\alpha_M} \quad & \sum_{i=1}^{M}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle \\ \text{subject to} \quad & \sum_{i=1}^{M}\alpha_i y_i = 0, \\ & 0 \le \alpha_i \le C, i = 1, \ldots, M, \end{aligned} \tag{8.16}$$

where $\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$ represents the inner product between vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and α_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 that feature space, the data generally become more (linearly) separable. Specifically, the optimization in (8.16) is transformed into

$$\min_{\alpha_1,\alpha_2,\cdots,\alpha_M} \quad \sum_{i=1}^{M}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle \tag{8.17}$$

with the same constraints as in (8.16). 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.

8.2.4 Logistic regression

LR [2] is a well-known binary classification approach. Given the same training dataset as in Section 8.2.3, the probability that a sample belongs to the positive class is of the form

$$p(y_i = +1|\boldsymbol{w}, b, \boldsymbol{x_i}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T\boldsymbol{x}_i - b)}. \tag{8.18}$$

The model parameter $\boldsymbol{w}$ and b are then estimated by solving the following maximum log-likelihood problem:

$$\min_{\boldsymbol{w},b} \quad \sum_{i=1}^{M}\log\big(1 + \exp\big(-y_i\big(\boldsymbol{w}^T\boldsymbol{x}_i + b\big)\big)\big). \tag{8.19}$$

Based on this, some regularization norms are often applied to alleviate overfitting when implementing model training, e.g., the l_1-norm regularizer is a common choice

to deal with high-dimensional features and obtain a robust and sparse classifier. As a result, the regularized LR optimization function can be written as

$$\min_{\boldsymbol{w},b} \quad \sum_{i=1}^{M} \log\left(1+\exp\left(-y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\right)\right)+\lambda(||\boldsymbol{w}||_1+|b|), \tag{8.20}$$

where $\lambda > 0$ is a tuning parameter used to control the sparsity.

8.3 Tensorial extensions of support vector machine

In this section, the STL [4] scheme is first described. Some typical tensorial extensions of SVM are then introduced.

8.3.1 Supervised tensor learning

In the STL scheme, a general formula for vector-based convex optimization problems is first assumed with a format of

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & f(\boldsymbol{w},b,\boldsymbol{\xi}) \\ \text{subject to} \quad & y_i c_i\left(\boldsymbol{w}^T\boldsymbol{x}_i+b\right)\geq \xi_i, \\ & i=1,\ldots,M, \end{aligned} \tag{8.21}$$

where $f:\mathbb{R}^{n+M+1}\to\mathbb{R}$ is a convex function for classification, $c_i:\mathbb{R}^{n+M+1}\to\mathbb{R}$, $1\leq i\leq M$ are convex constraint function with inputs $\boldsymbol{w}$, b, $\boldsymbol{\xi}$, $\boldsymbol{x}_i\in\mathbb{R}^n$, $1\leq i\leq M$, are training samples and the corresponding labels are represented by $y_i\in\{+1,-1\}$, $\boldsymbol{\xi}=[\xi_1,\xi_2,\ldots,\xi_M]\in\mathbb{R}^M$ are slack variables, and $\boldsymbol{w}$ and b are the classification hyperplane parameters, namely $y(\boldsymbol{x})=\text{sign}(\boldsymbol{w}^T\boldsymbol{x}+b)$.

STL extends the above vector-based optimization problem into tensorial format, in which the training samples are tensors. Specifically, given M training samples $\boldsymbol{\mathcal{X}}_i\in\mathbb{R}^{I_1\times I_2\times\cdots\times I_d}$ and the corresponding labels $y_i\in\{+1,-1\}, 1\leq i\leq M$, STL trains the classifier by solving the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w}^{(k)}|_{k=1}^d,b,\boldsymbol{\xi}} \quad & f\left(\boldsymbol{w}^{(k)}|_{k=1}^d,b,\boldsymbol{\xi}\right) \\ \text{subject to} \quad & y_i c_i\left(\boldsymbol{\mathcal{X}}_i\prod_{k=1}^{d}\times_k\boldsymbol{w}^{(k)}+b\right)\geq\xi_i, \\ & i=1,\ldots,M. \end{aligned} \tag{8.22}$$

Note that the input $\boldsymbol{x}_i$ in (8.21) is extended to $\boldsymbol{\mathcal{X}}_i$ in (8.22), while the model parameter $\boldsymbol{w}$ is replaced by $\boldsymbol{w}^{(k)}$, $1\leq k\leq d$. The trained classification hyperplane function through (8.22) is formed as $y(\boldsymbol{\mathcal{X}}_i)=\text{sign}(\boldsymbol{\mathcal{X}}_i\prod_{k=1}^{d}\times_k\boldsymbol{w}^{(k)}+b)$. The Lagrangian

for (8.22) can be written as

$$\begin{aligned} & L\big(\boldsymbol{w}^{(k)}|_{k=1}^{d}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}\big) \\ &= f\big(\boldsymbol{w}^{(k)}|_{k=1}^{d}, b, \boldsymbol{\xi}\big) - \sum_{i=1}^{M} \alpha_i \left(y_i c_i \left(\boldsymbol{\mathcal{X}}_i \prod_{k=1}^{d} \times_k \boldsymbol{w}^{(k)} + b \right) - \xi_i \right) \\ &= f\big(\boldsymbol{w}^{(k)}|_{k=1}^{d}, b, \boldsymbol{\xi}\big) - \sum_{i=1}^{M} \alpha_i y_i c_i \left(\boldsymbol{\mathcal{X}}_i \prod_{k=1}^{d} \times_k \boldsymbol{w}^{(k)} + b \right) + \boldsymbol{\alpha}^T \boldsymbol{\xi} \end{aligned} \tag{8.23}$$

with Lagrangian multipliers $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_M]^T, \alpha_i \geq 0$. The solution is determined by the saddle point of the Lagrangian, namely,

$$\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}^{(k)}|_{k=1}^{d}, b, \boldsymbol{\xi}} L\big(\boldsymbol{w}^{(k)}|_{k=1}^{d}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}\big). \tag{8.24}$$

The derivatives of $L(\boldsymbol{w}^{(k)}|_{k=1}^{d}, b, \boldsymbol{\xi})$ with respect to $\boldsymbol{w}^{(j)}$ and b can then be derived as

$$\begin{aligned} \partial_{\boldsymbol{w}^{(j)}} L &= \partial_{\boldsymbol{w}^{(j)}} f - \sum_{i=1}^{M} \alpha_i y_i \partial_{\boldsymbol{w}^{(j)}} c_i \left(\boldsymbol{\mathcal{X}}_i \prod_{k=1}^{d} \times_k \boldsymbol{w}^{(k)} + b \right) \\ &= \partial_{\boldsymbol{w}^{(j)}} f - \sum_{i=1}^{M} \alpha_i y_i \frac{dc_i}{dz} \partial_{\boldsymbol{w}^{(j)}} \left(\boldsymbol{\mathcal{X}}_i \prod_{k=1}^{d} \times_k \boldsymbol{w}^{(k)} + b \right) \\ &= \partial_{\boldsymbol{w}^{(j)}} f - \sum_{i=1}^{M} \alpha_i y_i \frac{dc_i}{dz} \big(\boldsymbol{\mathcal{X}}_i \overline{\times}_j \boldsymbol{w}^{(j)} \big) \end{aligned} \tag{8.25}$$

and

$$\begin{aligned} \partial_b L &= \partial_b f - \sum_{i=1}^{M} \alpha_i y_i \partial_{\boldsymbol{w}_j} c_i \left(\boldsymbol{\mathcal{X}}_i \prod_{k=1}^{d} \times_k \boldsymbol{w}^{(k)} + b \right) \\ &= \partial_b f - \sum_{i=1}^{M} \alpha_i y_i \frac{dc_i}{dz} \partial_b \left(\boldsymbol{\mathcal{X}}_i \prod_{k=1}^{d} \times_k \boldsymbol{w}^{(k)} + b \right) \\ &= \partial_b f - \sum_{i=1}^{M} \alpha_i y_i \frac{dc_i}{dz}, \end{aligned} \tag{8.26}$$

where $z = \boldsymbol{\mathcal{X}}_i \prod_{k=1}^{d} \times_k \boldsymbol{w}^{(k)} + b$. Setting $\partial_{\boldsymbol{w}^{(j)}} L = 0$ and $\partial_b L = 0$, we have

$$\partial_{\boldsymbol{w}^{(j)}} L = 0 \Rightarrow \partial_{\boldsymbol{w}^{(j)}} f = \sum_{i=1}^{M} \alpha_i y_i \frac{dc_i}{dz} \big(\boldsymbol{\mathcal{X}}_i \overline{\times}_j \boldsymbol{w}^{(j)} \big), \tag{8.27}$$

Algorithm 1 STL algorithm.

Input: Training dataset $\{\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times \cdots \times I_d}, y_i \in \{-1, 1\}\}_{i=1}^{M}$.

Output: The classification hyperplane parameters $\boldsymbol{w}^{(k)}|_{k=1}^{d}$; The bias b.

1: Initialize $\boldsymbol{w}^{(k)} \in \mathbb{R}^{I_k}$ as a random vector for $k = 1, 2, \ldots, d$.
2: Repeat steps 3–5 iteratively until convergence.
3: **for** $j = 1, \ldots, d$ **do**
4: Derive $\boldsymbol{w}^{(j)}$ by optimizing (8.29).
5: **end for**

$$\partial_b L = 0 \Rightarrow \partial_b f = \sum_{i=1}^{M} \alpha_i y_i \frac{dc_i}{dz}. \tag{8.28}$$

From (8.27), we observe that the solution for $\boldsymbol{w}^{(j)}$ depends on $\boldsymbol{w}^{(k)}$, $1 \le k \le d, k \ne j$, and we cannot derive the solution directly. STL proposes to utilize an alternating projection optimization scheme to tackle this. Specifically, STL derives $\boldsymbol{w}^{(j)}$ by assuming $\boldsymbol{w}^{(k)}, 1 \le k \le d, k \ne j$ are known. Therefore, the optimization problem is rewritten as

$$\begin{aligned} \min_{\boldsymbol{w}^{(j)}, b, \boldsymbol{\xi}} \quad & f\left(\boldsymbol{w}^{(j)}, b, \boldsymbol{\xi}\right) \\ \text{subject to} \quad & y_i c_i \left[\left(\boldsymbol{w}^{(j)}\right)^T \left(\boldsymbol{\mathcal{X}}_i \overline{\times}_j \boldsymbol{w}^{(k)}\right) + b\right] \ge \xi_i, \\ & i = 1, \ldots, M. \end{aligned} \tag{8.29}$$

After updating $\boldsymbol{w}^{(j)}$, $\boldsymbol{w}^{(j+1)}$ is then updated in a similar way. The algorithm is summarized in Algorithm 1, whose convergence proof can be found in [4].

8.3.2 Support tensor machines

Base on the STL scheme, we now present the STM.

8.3.2.1 Methodology

Suppose the input samples in the dataset $D=\{\boldsymbol{\mathcal{X}}_i, y_i\}_{i=1}^{M}$ are tensors $\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$. Following the STL scheme, a linear STM extends a linear SVM by defining d weight vectors $\boldsymbol{w}^{(k)} \in \mathbb{R}^{I_i}$ $(k = 1, \ldots, d)$ and rewriting (8.14) as

$$f(\boldsymbol{\mathcal{X}}) = \boldsymbol{\mathcal{X}} \times_1 \boldsymbol{w}^{(1)} \times_2 \cdots \times_d \boldsymbol{w}^{(d)} + b. \tag{8.30}$$

The graphical representation of (8.30) is shown in Fig. 8.6. The tensor $\boldsymbol{\mathcal{X}}$ is contracted along each of its modes with the weight vectors $\boldsymbol{w}^{(1)}, \ldots, \boldsymbol{w}^{(d)}$, resulting in a scalar that is added to the bias b. The weight vectors of the STM are computed by the alternating projection optimization procedure, which comprises d optimization problems. The main idea is to optimize each $\boldsymbol{w}^{(k)}$ in turn by fixing all weight vectors

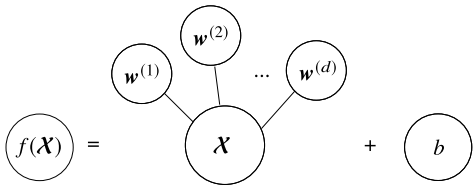

FIGURE 8.6

Graphical representation of an STM hyperplane function.

but $\boldsymbol{w}^{(k)}$. The k-th optimization problem is

$$\begin{aligned} \min_{\boldsymbol{w}^{(k)},b,\xi} \quad & \frac{1}{2}\beta\,||\boldsymbol{w}^{(k)}||_F^2 + C\sum_{i=1}^{M}\xi_i \\ \text{subject to} \quad & y_i\left(\left(\boldsymbol{w}^{(k)}\right)^T\hat{\boldsymbol{x}}_i + b\right) \geq 1-\xi_i, \\ & \xi_i \geq 0,\ i=1,\ldots,M, \end{aligned} \tag{8.31}$$

where

$$\beta = \prod_{\substack{1\leq l\leq d \\ l\neq k}} ||\boldsymbol{w}^{(l)}||_F^2 \quad \text{and} \quad \hat{\boldsymbol{x}}_i = \mathcal{X}_i \prod_{\substack{1\leq l\leq d \\ l\neq k}} \times_l \boldsymbol{w}^{(l)}.$$

The optimization problem (8.31) is equivalent to (8.15) for the linear SVM problem. This implies that any SVM learning algorithm can also be used for the linear STM. Each of the weight vectors of the linear STM is updated consecutively until the loss function of (8.31) converges. Each single optimization problem in learning an STM requires the estimation of only a few weight parameters, which alleviates the overfitting problem when M is relatively small. The weight tensor obtained from the outer product of the weight vectors

$$\mathcal{W} = \boldsymbol{w}^{(1)} \circ \boldsymbol{w}^{(2)} \circ \cdots \circ \boldsymbol{w}^{(d)} \tag{8.32}$$

is per definition rank-1 and allows us to rewrite (8.30) as

$$f(\mathcal{X}) = \langle \mathcal{W}, \mathcal{X} \rangle + b. \tag{8.33}$$

8.3.2.2 Examples

Ref. [9] employs STM for text categorization. Different from traditional text categorization algorithms which consider a text document as a vector in $\mathbb{R}^n$ based on the vector space model, the work [9] considers a document as a second-order tensor in

Table 8.1 Performance comparison in Reuters-21578.

Train & test split	Method	micro F_1	macro F_1
5% Training	SVM	0.8490	0.3355
	STM	**0.8666**	**0.4818**

Table 8.2 Performance comparison in TDT2.

Train & test split	Method	micro F_1	macro F_1
5% Training	SVM	0.8881	0.6477
	STM	**0.9064**	**0.7507**

$\mathbb{R}^{I_1 \times I_2}$, where $I_1 \times I_2 = n$. Two standard document datasets are used in [9], namely Reuters-21578[1] and TDT2[2].

The classification performance is evaluated by comparing the predicted label of each test document with the true label. Specifically, F_1 measure is used here which combines recall (r) and precision (p) with an equal weight in the following form:

$$F_1(r, p) = \frac{2rp}{r + p}.$$

To measure the overall classification performance, Ref. [9] employs the following two metrics. The first is to simply average the F_1 scores of different categories and then derive a mean F_1 score, namely the macro-averaging F_1 score. In the second way, instead of calculating each category's F_1 score, the authors consider the global F_1 score over all the $n \times m$ binary decisions, where n is the total number of test documents and m is the number of categories under consideration, which is known as the micro-averaging score. Both scores are the larger the better.

5% documents are used as the training set for both Reuters-21578 and TDT2. Tables 8.1 and 8.2 show the classification results on two datasets. As can be seen from the results, STM outperforms SVM on both micro-averaged F_1 and macro-averaged F_1 with a large margin. The experimental results demonstrate the greater performance of STM on small training sample cases over SVM.

8.3.2.3 Conclusion

Based on the STL scheme, STM accepts tensors as input and therefore keeps the useful structural information in data. However, the constraint that the weight tensor $\mathcal{W}$ in STM is a rank-1 tensor has a significant impact on the expressive power of the STM, resulting in usually unsatisfactory classification accuracy for more complicated

[1] The Reuters-21578 corpus can be found at http://www.daviddlewis.com/resources/testcollections/reuters21578/.

[2] The Nist Topic Detection and Tracking corpus can be found at http://www.nist.gov/speech/tests/tdt/tdt98/index.html.

data. Therefore, it is necessary to generalize the rank-1 tensor in STM into a more general tensorial format.

8.3.3 Higher-rank support tensor machines

As mentioned, an STM assumes a simple rank-1 tensor as the classifier hyperplane parameter, which might not be powerful enough to model the hyperplane function. In [15], the authors propose a higher-rank STM wherein the classifier parameter is formulated as a sum of rank-1 tensors, namely the general CP tensor format. By doing so, the classification capability is significantly enhanced.

8.3.3.1 Methodology

Consider the following general higher-rank STM optimization problem:

$$\begin{aligned} \min_{\mathcal{W},b,\xi} \quad & \frac{1}{2}\langle \mathcal{W}, \mathcal{W} \rangle + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \left(\langle \mathcal{W}, \mathcal{X}_i \rangle + b \right) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, M, \end{aligned} \tag{8.34}$$

where $\mathcal{W}$ is a general d-way parameter tensor with the same dimensions as the training sample $\mathcal{X}$. Adopting the alternating optimization scheme as in STL, each time only the parameters that are associated with the j-th tensor mode of $\mathcal{W}$ are updated, while fixing all other parameters. According to (8.7), (8.34) can be rewritten into the following format:

$$\begin{aligned} \min_{\boldsymbol{W}_{(j)},b,\xi} \quad & \frac{1}{2} Tr\left[\boldsymbol{W}_{(j)} \boldsymbol{W}_{(j)}^T \right] + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \left(Tr\left[\boldsymbol{W}_{(j)} \boldsymbol{X}_{(j)i}^T \right] + b \right) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, M, \end{aligned} \tag{8.35}$$

where $\boldsymbol{X}_{(j)i}^T$ represents the j-th mode matricization of the i-th training sample. Under the assumption that the weight parameter tensor $\mathcal{W}$ can be decomposed into its CP tensor format as in (8.2), $\boldsymbol{W}_{(j)}$ in (8.35) can be rewritten as $\boldsymbol{U}^{(j)}(\boldsymbol{U}^{(-j)})^T$ according to (8.6). Therefore, (8.35) can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{U}^{(j)},b,\xi} \quad & \frac{1}{2} Tr\left[\boldsymbol{U}^{(j)} \left(\boldsymbol{U}^{(-j)} \right)^T \boldsymbol{U}^{(-j)} \left(\boldsymbol{U}^{(j)} \right)^T \right] + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \left(Tr\left[\boldsymbol{U}^{(j)} \left(\boldsymbol{U}^{(-j)} \right)^T \boldsymbol{X}_{(j)i}^T \right] + b \right) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, M. \end{aligned} \tag{8.36}$$

With the above optimization problem, we can iteratively update $\boldsymbol{U}^{(j)}$, $j = 1, \ldots, d$, employing the idea of STL. To further facilitate the implementation of (8.36), we

Algorithm 2 Higher-rank STM algorithm.

Input: Training dataset $\{\mathcal{X}_i \in \mathbb{R}^{I_1 \times \cdots \times I_d}, y_i \in \{-1, 1\}\}_{i=1}^{M}$.
Output: The classification hyperplane parameters $\mathcal{W}$; The bias b.

1: Initialize $\mathcal{W}$ as a sum of random rank-1 tensors.
2: Repeat steps 3–6 iteratively until convergence.
3: **for** $j = 1, \ldots, d$ **do**
4: Derive $\widetilde{\boldsymbol{U}}^{(j)}$ by optimizing (8.39).
5: Update $\boldsymbol{U}^{(j)} = \widetilde{\boldsymbol{U}}^{(j)} \boldsymbol{B}^{-\frac{1}{2}}$
6: **end for**

transfer it into a classic vector-based SVM format. By doing so, any existing SVM solver can be employed to readily solve the higher-rank STM problem.

Let $\boldsymbol{B} = (\boldsymbol{U}^{(-j)})^T \boldsymbol{U}^{(-j)}$ such that $\boldsymbol{B}$ is a positive-definite matrix. Defining $\widetilde{\boldsymbol{U}}^{(j)} = \boldsymbol{U}^{(j)} \boldsymbol{B}^{\frac{1}{2}}$, we have

$$Tr\big[\boldsymbol{U}^{(j)}\big(\boldsymbol{U}^{(-j)}\big)^T \boldsymbol{U}^{(-j)}\big(\boldsymbol{U}^{(j)}\big)^T\big] = Tr\big[\widetilde{\boldsymbol{U}}^{(j)}\big(\widetilde{\boldsymbol{U}}^{(j)}\big)^T\big] = vec\big(\widetilde{\boldsymbol{U}}^{(j)}\big)^T vec\big(\widetilde{\boldsymbol{U}}^{(j)}\big). \tag{8.37}$$

Also, letting $\widetilde{\boldsymbol{X}}_{(j)i} = \boldsymbol{X}_{(j)i} \boldsymbol{U}^{(-j)} \boldsymbol{B}^{-\frac{1}{2}}$, we have

$$Tr\big[\boldsymbol{U}^{(j)}\big(\boldsymbol{U}^{(-j)}\big)^T \boldsymbol{X}_{(j)i}^T\big] = Tr\big[\widetilde{\boldsymbol{U}}^{(j)} \widetilde{\boldsymbol{X}}_{(j)i}^T\big] = vec\big(\widetilde{\boldsymbol{U}}^{(j)}\big)^T vec(\widetilde{\boldsymbol{X}}_{(j)i}). \tag{8.38}$$

Therefore, (8.36) can be rewritten as

$$\begin{aligned} \min_{\widetilde{\boldsymbol{U}}^{(j)}, b, \xi} \quad & \frac{1}{2} vec\big(\widetilde{\boldsymbol{U}}^{(j)}\big)^T vec\big(\widetilde{\boldsymbol{U}}^{(j)}\big) + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \big(vec\big(\widetilde{\boldsymbol{U}}^{(j)}\big)^T vec(\widetilde{\boldsymbol{X}}_{(j)i}) + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, M. \end{aligned} \tag{8.39}$$

Eq. (8.39) is a classic vector-based SVM problem. After deriving the solution of $\widetilde{\boldsymbol{U}}^{(j)}$ in each iteration, the model parameter $\boldsymbol{U}^{(j)}$ is updated with $\boldsymbol{U}^{(j)} = \widetilde{\boldsymbol{U}}^{(j)} \boldsymbol{B}^{-\frac{1}{2}}$. The overall procedure for implementing higher-rank STM is summarized in Algorithm 2.

8.3.3.2 Complexity analysis

Because the subproblem (8.39) of the higher-rank STM optimization problem is naturally an SVM problem, we analyze the computation complexity of SVM first. When there are M training samples and each sample is in $\mathbb{R}^n$, after constructing the linear kernel matrix, the worst case of the computation complexity of an SVM is $\mathcal{O}(M^3)$. When it comes to higher-rank STM, the overall optimization problem is separated into d subproblems, where d is the mode number of the training sample. Furthermore,

Table 8.3 Accuracies (%) achieved by various methods for the KTH action database.

Method	SVM	STM	Higher-rank STM
Accuracy	88.5	89.3	**93.3**

these subproblems would be solved for p iterations until convergence. Therefore, the complexity of solving a higher-rank STM problem is $\mathcal{O}(dpM^3)$.

8.3.3.3 Examples

Considering the fact that there is no known closed-form solution to determine the rank R of a tensor and that rank determination of a tensor is still an open problem [12,13], in higher-rank STM, the common way to determine the optimal weight tensor rank is by utilizing grid search.

Ref. [15] utilizes the higher-rank STM for human action recognition. Specifically, the KTH [27] dataset is employed to check the performance of higher-rank STM. The KTH Action Dataset depicts 25 subjects performing six different activities, namely, "boxing," "handclapping," "handwaving," "jogging," "running," and "walking." Each training sample is a three-way tensor with modes pixel–pixel–time. For a fair comparison, a linear kernel is applied in traditional SVM considering STM and higher-rank STM are all linear classifiers. The classification results are shown in Table 8.3. We note that the higher-rank STM achieves a 4% accuracy enhancement over STM, and they both perform better than traditional SVM.

Ref. [14] tests the performance of higher-rank STM on human face classification. Specifically, nine second-order face recognition datasets, namely Yale32×32, Yale64×64, ORL32×32, ORL64×64, C05, C07, C09, C27, and C29, from http://www.zjucadcg.cn/dengcai/Data/FaceData.html are employed. Moreover, since an SVM can naturally utilize the kernel trick and empower its classification capability, [14] compares the higher-rank STM with kernel SVM employing a Gaussian radial basis function (RBF) kernel. Table 8.4 shows the experimental results. We observe that in terms of test accuracy, STM outperforms SVM only in one dataset, while higher-rank STM outperforms SVM and STM in all nine datasets.

8.3.3.4 Conclusion

Higher-rank STM assumes the parameters defining the separating hyperplane form a tensor that can be written as a sum of rank-1 terms according to the CP decomposition. This generalizes STM and achieves better classification performance on several complicated tensorial data classification tasks.

8.3.4 Support Tucker machines

Apart from higher-rank STM, STuM [29] is another approach that generalizes STM by employing the Tucker tensor format to replace the rank-1 weight tensor.

Table 8.4 Comparison of the accuracies (%) of SVM, STM, and higher-rank STM on nine experimental datasets.

Dataset	SVM	STM	Higher-rank STM
Yale32 × 32	77.3	74.0	**79.0**
Yale64 × 64	84.3	82.3	**85.3**
ORL32 × 32	97.8	97.0	**98.0**
ORL64 × 64	97.8	76.5	**98.5**
C05	98.6	98.1	**98.8**
C07	96.5	95.4	**96.7**
C09	97.4	96.2	**97.5**
C27	**96.7**	95.1	**96.7**
C29	**96.6**	94.8	**96.6**

8.3.4.1 Methodology

Starting from (8.35), we now assume the weight tensor $\mathcal{W}$ is represented by a Tucker tensor decomposition. According to (8.11), (8.35) can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{P}^{(j)},b,\xi} \quad & \frac{1}{2} Tr\big[\boldsymbol{P}^{(j)} \boldsymbol{G}_{(j)} \big(\boldsymbol{P}_{\otimes}^{(-j)}\big)^T \boldsymbol{P}_{\otimes}^{(-j)} \boldsymbol{G}_{(j)}^T \big(\boldsymbol{P}^{(j)}\big)^T\big] + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \big(Tr\big[\boldsymbol{P}^{(j)} \boldsymbol{G}_{(j)} \big(\boldsymbol{P}_{\otimes}^{(-j)}\big)^T \boldsymbol{X}_{(j)i}^T\big] + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, M. \end{aligned} \tag{8.40}$$

It is desirable to reformulate the above optimization problem such that the classic vector-based SVM implementation can be employed to solve it. We let $\boldsymbol{H}^{(j)} = \boldsymbol{G}_{(j)} (\boldsymbol{P}_{\otimes}^{(-j)})^T$. Now (8.40) can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{P}^{(j)},b,\xi} \quad & \frac{1}{2} Tr\big[\boldsymbol{P}^{(j)} \boldsymbol{H}^{(j)} \big(\boldsymbol{H}^{(j)}\big)^T \big(\boldsymbol{P}^{(j)}\big)^T\big] + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \big(Tr\big[\boldsymbol{P}^{(j)} \boldsymbol{H}^{(j)} \boldsymbol{X}_{(j)i}^T\big] + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, M. \end{aligned} \tag{8.41}$$

We further define $\boldsymbol{K} = \boldsymbol{H}^{(j)} (\boldsymbol{H}^{(j)})^T$. Obviously, $\boldsymbol{K}$ is positive-definite. Also, letting $\widetilde{\boldsymbol{P}}^{(j)} = \boldsymbol{P}^{(j)} \boldsymbol{K}^{\frac{1}{2}}$, we have

$$Tr\big[\boldsymbol{P}^{(j)} \boldsymbol{H}^{(j)} \big(\boldsymbol{H}^{(j)}\big)^T \big(\boldsymbol{P}^{(j)}\big)^T\big] = Tr\big[\widetilde{\boldsymbol{P}}^{(j)} \big(\widetilde{\boldsymbol{P}}^{(j)}\big)^T\big] = vec\big(\widetilde{\boldsymbol{P}}^{(j)}\big)^T vec\big(\widetilde{\boldsymbol{P}}^{(j)}\big). \tag{8.42}$$

Algorithm 3 STuM algorithm.

Input: Training dataset $\{\mathcal{X}_i \in \mathbb{R}^{I_1 \times \cdots \times I_d}, y_i \in \{-1, 1\}\}_{i=1}^{M}$; the given Tucker ranks

Output: The classification hyperplane parameters $\mathcal{G}, \boldsymbol{P}^{(1)}, \boldsymbol{P}^{(2)}, \ldots, \boldsymbol{P}^{(d)}$; The bias b.

1: Initialize $\mathcal{G}, \boldsymbol{P}^{(1)}, \boldsymbol{P}^{(2)}, \ldots, \boldsymbol{P}^{(d)}$ randomly.
2: Repeat steps 3–8 iteratively until convergence.
3: **for** $j = 1, \ldots, d$ **do**
4: Derive $\widetilde{\boldsymbol{P}}^{(j)}$ by optimizing (8.44).
5: Update $\boldsymbol{P}^{(j)} = \widetilde{\boldsymbol{P}}^{(j)} \boldsymbol{K}^{-\frac{1}{2}}$
6: **end for**
7: Derive $\boldsymbol{G}_{(1)}$ by optimizing (8.45)
8: Reshape $\boldsymbol{G}_{(1)}$ back and derive $\mathcal{G}$

By defining $\widetilde{\boldsymbol{X}}_{(j)i} = \boldsymbol{X}_{(j)i}(\boldsymbol{H}^{(j)})^T \boldsymbol{K}^{-\frac{1}{2}}$, we have

$$Tr\big[\boldsymbol{P}^{(j)} \boldsymbol{H}^{(j)} \boldsymbol{X}_{(j)i}^T\big] = Tr\big[\widetilde{\boldsymbol{P}}^{(j)} \widetilde{\boldsymbol{X}}_{(j)i}^T\big] = vec\big(\widetilde{\boldsymbol{P}}^{(j)}\big)^T vec(\widetilde{\boldsymbol{X}}_{(j)i}). \tag{8.43}$$

Therefore, (8.41) can be rewritten as

$$\begin{aligned} \min_{\widetilde{\boldsymbol{P}}^{(j)}, b, \xi} \quad & \frac{1}{2} vec\big(\widetilde{\boldsymbol{P}}^{(j)}\big)^T vec\big(\widetilde{\boldsymbol{P}}^{(j)}\big) + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \big(vec\big(\widetilde{\boldsymbol{P}}^{(j)}\big)^T vec(\widetilde{\boldsymbol{X}}_{(j)i}) + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, M. \end{aligned} \tag{8.44}$$

We note that (8.44) is a classic vector-based SVM problem. By solving it, we can derive $\widetilde{\boldsymbol{P}}^{(j)}$ and update $\boldsymbol{P}^{(j)} = \widetilde{\boldsymbol{P}}^{(j)} \boldsymbol{K}^{-\frac{1}{2}}$. After updating $\boldsymbol{P}^{(1)}, \boldsymbol{P}^{(2)}, \ldots, \boldsymbol{P}^{(d)}$, we then formulate the optimization problem for the Tucker core tensor $\mathcal{G}$. According to (8.7) and (8.12), the optimization problem (8.35) can be rewritten in terms of $\mathcal{G}$ as follows:

$$\begin{aligned} \min_{\boldsymbol{G}_{(1)}, b, \xi} \quad & \frac{1}{2} \big(\boldsymbol{P}_{\otimes} vec(\boldsymbol{G}_{(1)})\big)^T \boldsymbol{P}_{\otimes} vec(\boldsymbol{G}_{(1)}) + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \big(\big(\boldsymbol{P}_{\otimes} vec(\boldsymbol{G}_{(1)})\big)^T vec(\boldsymbol{X}_{(j)i}) + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, M. \end{aligned} \tag{8.45}$$

This problem can be solved by any classic vector-based SVM solver. The overall procedure for STuM is summarized in Algorithm 3.

Table 8.5 Comparison of proposed methods with state-of-the-art (CS-1/CS-5).

Test set	HMM [10]	LTN [25]	GEI [11]	ETG [23]	ETGLDA [23]	SVM	STM	STuM
A	99/**100**	94/99	**100**/**100**	92/96	99/**100**	80/97	92/**100**	99/**100**
B	**89**/**90**	83/85	85/85	85/90	88/**93**	79/**93**	81/90	85/**93**
C	78/**90**	78/83	80/88	76/81	**83**/**88**	68/85	73/88	79/**90**
D	35/65	33/65	30/55	39/55	36/**71**	30/54	47/67	**53**/**71**
E	29/65	24/67	33/55	29/52	29/60	23/46	48/79	**63**/**86**
F	18/60	17/58	21/41	21/58	21/59	24/49	29/49	**42**/**63**
G	24/50	21/48	29/48	21/50	21/60	12/37	31/71	**52**/**87**
Mean	53/74	50/72	54/67	52/69	54/76	45/62	57/68	**68**/**84**

Table 8.6 Accuracies (%) achieved by various methods for the KTH action database.

Method	[22]	[21]	[17]	[16]	SVM	STM	STuM
Accuracy	90.5	91.7	87.7	95.3	88.5	89.3	**92.3**

8.3.4.2 Examples

To prove the superiority of STuM, Ref. [29] implements experiments on gait and human action recognition.

For gait recognition, the USF HumanID Gait Challenge dataset [23] is used. The database consists of 452 sequences of 74 subjects walking in elliptical paths in front of the camera. Therefore, each sequence is a third-order tensor. The datasets are separated into one training set and seven test sets (A–G). The detailed experimental setting can be found in [24]. The results are shown in Table 8.5. Apart from SVM with linear kernel and STM, some common gait recognition methods are also included. The results are evaluated using the Cumulative Scores 1 and 5 (CS-1/CS-5, respectively) similarly to [23,25]. The results of STuM are reported by setting the Tucker core tensor with dimensions $18 \times 18 \times 18$ as in [29]. From Table 8.5, we observe that tensor-based methods largely outperform vector-based ones. We can also see that the STuMs outperform the classical STMs with a large margin.

For the human action recognition experiments, Ref. [29] employed the commonly used KTH dataset [27]. The KTH Action Dataset depicts 25 subjects performing six different activities, namely, "boxing," "handclapping," "handwaving," "jogging," "running," and "walking." The Tucker ranks are set to 6 for all modes in STuM, so that the Tucker core tensor is of size $6 \times 6 \times 6$. The comparison results are listed in Table 8.6. Again, STuMs perform significantly better than other methods.

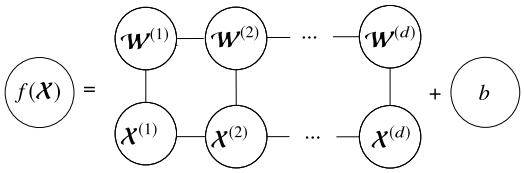

FIGURE 8.7

Tensor graphical representation of an STTM hyperplane function.

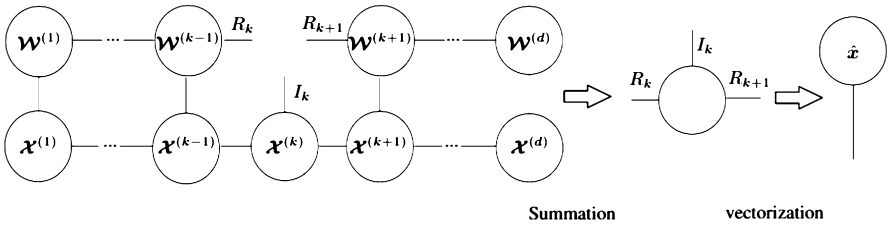

FIGURE 8.8

The computation diagram of $\hat{\boldsymbol{x}}$.

8.3.5 Support tensor train machines

Although higher-rank STM and STuM generalize STM and achieve a better classification performance in high-dimensional data, they still suffer from various issues. First, determining the CP rank in higher-rank STM is NP-complete. Second, the number of model parameters in STuM is still exponentially large since the Tucker core tensor $\mathcal{G}$ has the same tensor modes as the original parameter tensor $\mathcal{W}$. To solve these issues, Ref. [31] proposes a support TT machine (STTM) wherein the rank-1 weight tensor of an STM is replaced by a TT that can approximate any tensor with a scalable number of parameters. Moreover, a TT mixed-canonical form has been exploited to speed up algorithmic convergence.

8.3.5.1 Methodology

As mentioned in Section 8.3.2, an STM suffers from its weak expressive power due to its rank-1 weight tensor $\mathcal{W}$. In [31], the proposed STTM replaces the rank-1 weight tensor by a TT with prescribed TT ranks. Moreover, most real-world data contain redundancies and uninformative parts. Based on this knowledge, STTM also utilizes a TT decomposition to approximate the original data tensor and further alleviates the overfitting problem. The conversion of the training samples into a TT can be done using the TT-SVD algorithm [33, p. 2301], which allows the user to determine the relative error of the approximation. A graphical representation of the STTM hyperplane equation is shown in Fig. 8.7. Both the data tensor $\mathcal{X}$ and the weight tensor $\mathcal{W}$ are represented by TTs, and the summations correspond to computing the inner product $\langle \mathcal{X}, \mathcal{W} \rangle$.

The TT cores $\mathcal{W}^{(1)}$, $\mathcal{W}^{(2)}$, …, $\mathcal{W}^{(d)}$ are also computed using an alternating projection optimization procedure [4], namely, iteratively fixing $d-1$ TT cores and updating the remaining cores until convergence. This updating occurs in a "sweeping" fashion, namely, first update $\mathcal{W}^{(1)}$ and proceed towards $\mathcal{W}^{(d)}$. Once the core $\mathcal{W}^{(d)}$ is updated, the algorithm sweeps back to $\mathcal{W}^{(1)}$ and repeats this procedure until the termination criterion is met.

Suppose we want to update $\mathcal{W}^{(k)}$. First, the TT of the weight tensor $\mathcal{W}$ is brought into site-k-mixed-canonical form (see Definition 8.9). From Section 8.2.2.4, the norm of the whole weight tensor is located in the $\mathcal{W}^{(k)}$ TT core. To reformulate the optimization problem (8.31) in terms of the unknown core $\mathcal{W}^{(k)}$, the inner product $\langle \mathcal{X}, \mathcal{W} \rangle$ is rewritten as $vec(\mathcal{W}^{(k)})^T \hat{\boldsymbol{x}}$. The vector $\hat{\boldsymbol{x}}$ is obtained by summing over the tensor network for $\langle \mathcal{W}, \mathcal{X} \rangle$ depicted in Fig. 8.7 with the TT core $\mathcal{W}^{(k)}$ removed and vectorizing the resulting three-way tensor. These two computational steps to compute $\hat{\boldsymbol{x}}$ are graphically depicted in Fig. 8.8. The STTM hyperplane function can then be rewritten as $vec(\mathcal{W}^{(k)})^T \hat{\boldsymbol{x}} + b$, so that $\mathcal{W}^{(k)}$ can be updated from the following optimization problem:

$$\begin{aligned} \min_{\mathcal{W}^{(k)}, b, \xi} \quad & \frac{1}{2} ||vec\big(\mathcal{W}^{(k)}\big)||_F^2 + C \sum_{i=1}^{M} \xi_i \\ \text{subject to} \quad & y_i \big(vec\big(\mathcal{W}^{(k)}\big)^T \hat{\boldsymbol{x}}_i + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ i = 1, \ldots, M, \end{aligned} \tag{8.46}$$

using any computational method for standard SVMs. Suppose now that the next TT core to be updated is $\mathcal{W}^{(k+1)}$. The new TT for $\mathcal{W}$ then needs to be put into site-$(k+1)$-mixed-canonical form, which can be achieved by reshaping the new $\mathcal{W}^{(k)}$ into an $R_k I_k \times R_{k+1}$ matrix $\boldsymbol{W}^{(k)}$ and computing its thin QR decomposition

$$\boldsymbol{W}^{(k)} = \boldsymbol{Q}\boldsymbol{R},$$

where $\boldsymbol{Q}$ is an $R_k I_k \times R_{k+1}$ matrix with orthogonal columns and $\boldsymbol{R}$ is an $R_{k+1} \times R_{k+1}$ upper triangular matrix. Updating the tensors $\mathcal{W}^{(k)}$ and $\mathcal{W}^{(k+1)}$ as

$$\begin{aligned} \mathcal{W}^{(k)} &:= \text{reshape}\big(\boldsymbol{Q}, [R_k, I_k, R_{k+1}]\big), \\ \mathcal{W}^{(k+1)} &:= \mathcal{W}^{(k+1)} \times_1 \boldsymbol{R} \end{aligned}$$

results in a site-$(k+1)$-mixed-canonical form for $\mathcal{W}$. An optimization problem similar to (8.46) can then be derived for $\mathcal{W}^{(k+1)}$.

The training algorithm of the STTM is summarized as pseudo-codes in Algorithm 4. The TT cores for the weight tensor $\mathcal{W}$ are initialized randomly. Bringing this TT into site-1-mixed-canonical form can then be done by applying the QR decomposition step starting from $\mathcal{W}^{(d)}$ and proceeding towards $\mathcal{W}^{(2)}$. The final $\boldsymbol{R}$ factor is absorbed into $\mathcal{W}^{(1)}$, which brings the TT into site-1-mixed-canonical form. The termination criterion in line 4 can be a maximum number of loops and/or when the training error falls below a prescribed threshold.

Algorithm 4 STTM algorithm.

Input: TT ranks $R_2, \ldots, R_d$ of $\mathcal{W}^{(1)}, \mathcal{W}^{(2)}, \ldots, \mathcal{W}^{(d)}$; Training dataset $\{\mathcal{X}_i \in \mathbb{R}^{I_1 \times \cdots \times I_d}, y_i \in \{-1, 1\}\}_{i=1}^{M}$; Relative error ϵ of TT approximation of $\mathcal{X}$.

Output: The TT cores $\mathcal{W}^{(1)}, \mathcal{W}^{(2)}, \ldots, \mathcal{W}^{(d)}$; The bias b.

Initialize $\mathcal{W}^{(k)} \in \mathbb{R}^{R_k \times I_k \times R_{k+1}}$ as a random/prescribed three-way tensor for $k = 1, 2, \ldots, d$.
Compute the TT approximation of training samples $\{\mathcal{X}_i\}_{i=1}^{M}$ with relative error ϵ using TT-SVD.
Cast $\mathcal{W}$ into the site-1-mixed-canonical TT form.
while termination criterion not satisfied **do**
  for $k = 1, \ldots, d-1$ **do**
    $\mathcal{W}^{(k)}, b \leftarrow$ Solve optimization problem (8.46).
    $\boldsymbol{W}^{(k)} \leftarrow \text{reshape}(\mathcal{W}^{(k)}, [R_k I_k, R_{k+1}])$.
    Compute thin QR decomposition $\boldsymbol{W}^{(k)} = \boldsymbol{Q}\boldsymbol{R}$.
    $\mathcal{W}^{(k)} \leftarrow \text{reshape}(\boldsymbol{Q}, [R_k, I_k, R_{k+1}])$.
    $\mathcal{W}^{(k+1)} \leftarrow \mathcal{W}^{(k+1)} \times_1 \boldsymbol{R}$.
  end for
  for $k = d, \ldots, 2$ **do**
    $\mathcal{W}^{(k)}, b \leftarrow$ Solve optimization problem (8.46).
    $\boldsymbol{W}^{(k)} \leftarrow \text{reshape}(\mathcal{W}^{(k)}, [R_k, I_k R_{k+1}])$.
    Compute thin QR decomposition $\boldsymbol{W}^{(k)^T} = \boldsymbol{Q}\boldsymbol{R}$.
    $\mathcal{W}^{(k)} \leftarrow \text{reshape}(\boldsymbol{Q}^T, [R_k, I_k, R_{k+1}])$.
    $\mathcal{W}^{(k-1)} \leftarrow \mathcal{W}^{(k-1)} \times_3 \boldsymbol{R}^T$.
  end for
end while

8.3.5.2 Complexity analysis

Assume that the tensorial training data $D=\{\mathcal{X}_i\ y_i\}_{i=1}^{M}$ are given, where tensors $\mathcal{X}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ are in TT format and their ranks are $R_2, \ldots, R_d$. With $I := \max\{I_1, \ldots, I_d\}$ and $R := \max\{R_2, \ldots, R_d\}$, the computation complexity of forming the small-size SVM optimization problem (8.46) from the overall STTM optimization problem is $\mathcal{O}(MdIR^2)$. The complexity is linear to the tensorial data order d due to the TT structure. Moreover, real-world tensorial data often exhibit the low rank property, namely, R is often small, which indicates the overall complexity is also low. For data storage, the traditional SVM calls for $\mathcal{O}(MI^d)$ space, while that of the STTM is $\mathcal{O}(MIR^2)$. This again shows a great reduction especially when the data order d is large.

8.3.5.3 Effect of TT ranks on STTM classification

This section demonstrates the effect of TT ranks on tensor-based classification. Specifically, the CIFAR-10 database [26] is employed to evaluate the influence. The images of CIFAR-10 are of dimensions $32 \times 32 \times 3$. The TT ranks of the weight TT

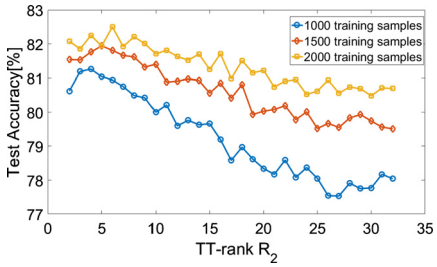

FIGURE 8.9

Test accuracy of STTM on different TT ranks R_2.

were fixed to $R_1 = R_4 = 1$, $R_3 = 3$ and different experiment runs were performed with R_2 varied from 2 to 32. The detailed experiment setting can be found in [31]. Fig. 8.9 shows the STTM test accuracy for different TT ranks when the training sample number is equal to $2k$, $3k$, and $4k$, respectively. The maximal test accuracy for these three sizes is achieved when R_2 is 4, 5, and 6, respectively. A downward trend of all three curves can be observed for TT ranks larger than the optimal value, indicating that higher TT ranks may lead to overfitting. On the other hand, decreasing the TT rank from its optimal value also decreases the test accuracy down to the rank-1 STM case.

8.3.5.4 Updating in site-k-mixed-canonical form

The authors of [31] investigated the effect of keeping the TT of $\mathcal{W}$ in a site-k-mixed-canonical form when updating $\mathcal{W}^{(k)}$. According to the setting of [31], two image classes of CIFAR-10 are chosen for this investigation, namely `airplane` and `automobile`. Three thousand samples of both classes were used for the STTM training. Fig. 8.10 shows the training accuracy for each TT core update iteration in Algorithm 4, with and without the site-k-mixed-canonical form. Updating without the site-k-mixed-canonical form implies that lines 3, 8–10, and 15–17 of Algorithm 4 are not executed, which results in an oscillatory training accuracy ranging between 50% and 89% without any overall convergence. Updating the TT cores $\mathcal{W}^{(k)}$ in a site-k-mixed-canonical form, however, displays a very fast convergence of the training accuracy to around 92%.

8.3.5.5 Examples

The performance of STTM is checked on several image datasets in [31]. Here we present the results on two of those datasets, namely MNIST [30] and ORL[3].

MNIST has a training set of 60k samples and a test set of 10k samples. Each sample is a 28×28 grayscale picture which is reshaped into a $7 \times 4 \times 7 \times 4$ tensor, as this provides more flexibility to choose TT ranks when applying Algorithm 4. For

[3] http://www.zjucadcg.cn/dengcai/Data/FaceData.html.

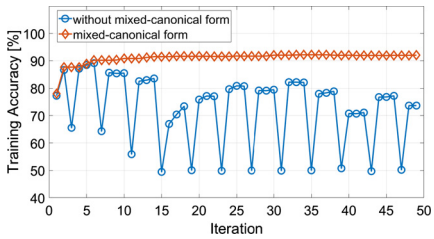

FIGURE 8.10

Comparison of training accuracy of STTMs trained with and without site-k-mixed-canonical form.

Table 8.7 Experiment settings for the four methods.

Method	Input structure	Tensor ranks
SVM	784×1 vector	NA
STM	28×28 matrix	1
STuM	$7 \times 4 \times 7 \times 4$ tensor	$4, 4, 4, 4$
STTM	$7 \times 4 \times 7 \times 4$ tensor	$1, 5, 5, 4, 1$

the STM initialization, the SVM weight vector is reshaped into a 28×28 matrix from which the best rank-1 approximation is used. For the STuM and STTM initialization, the SVM weight vector is reshaped into a $7 \times 4 \times 7 \times 4$ tensor and then converted into its Tucker and TT forms, respectively, with prescribed tensor ranks. Table 8.7 shows the experiment setting for those four methods. All classifiers were trained for training sample batch sizes of $10k$, $20k$, $30k$, and $60k$ in four different experiments. The test accuracy of the different methods for different batch sizes is listed in Table 8.8. STTM achieves the best classification performance for all sizes. The STM performs worse than the standard SVM due to the restrictive expressive power of the rank-1 weight matrix. The test accuracies of STuM are not posted when the training sample sizes are 30k and 60k since they cost much more time (more than 60 h) than the other three methods. This observation indicates that an STuM may not work well when the training sample size is large.

For the ORL database, it contains 400 grayscale face images, and the detailed information about ORL datasets is listed in Table 8.9. The dataset is separated into training and test sets. The detailed experiment settings and classification accuracies (average value of five repeated tests) for ORL32x32 and ORL64x64 when employing different methods are listed in Table 8.10 and Table 8.11. STTM achieves a similar classification performance compared with that of SVM, and they both perform better than STM and STuM.

Table 8.8 Test accuracy (%) under different training sample sizes.

Method	Training sample size			
	$10k$	$20k$	$30k$	$60k$
SVM	91.64	92.84	93.28	93.99
STM	88.36	89.96	89.82	90.54
STuM	90.45	92.28	–	–
STTM	**92.27**	**93.71**	**93.86**	**94.12**

Table 8.9 Detailed information of experimental datasets.

Datasets	Number of samples	Number of classes	Size
ORL 32x32	400	40	32x32
ORL 64x64	400	40	64x64

Table 8.10 Experimental settings and classification accuracy (%) of four methods for ORL32x32.

Method	Input structure	Tensor ranks	Test accuracy
SVM	1024x1 vector	NA	**96.25**
STM	32x32 matrix	1	93.75
STuM	8x4x8x4 tensor	4, 4, 4, 4	93.50
STTM	8x4x8x4 tensor	1, 4, 4, 4, 1	**96.25**

Table 8.11 Experimental settings and classification accuracy (%) of four methods for ORL64x64.

Method	Input structure	Tensor ranks	Test accuracy
SVM	4096x1 vector	NA	**96.25**
STM	64x64 matrix	1	92.71
STuM	8x8x8x8 tensor	4, 4, 4, 4	94.40
STTM	8x8x8x8 tensor	1, 4, 4, 4, 1	**96.25**

8.3.5.6 Conclusion

STTM employs a more general TT structure to largely escalate the model expressive power, which leads to a better classification accuracy than STM. Moreover, the tensor model in STTM is more scalable than that in STuM, which achieves faster training when the training sample size is large. The necessity of keeping the weight TT of STTM in a site-k-mixed-canonical form during model training is empirically confirmed through experiments, in which the algorithm convergence speed can be accelerated significantly.

8.3.6 Kernelized support tensor train machines

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. In [37], the authors proposed a K-STTM to deal with nonlinear tensorial classification problems. Firstly, the TT decomposition [33] is employed to decompose the given tensor data so that a more compact and informative representation of it can be derived. Secondly, the authors define a TT-based feature mapping strategy to derive a high-dimensional TT in the feature space. This strategy enables one to apply different feature mappings on different data modes, which naturally provides a way to leverage the multimode nature of tensorial data. Thirdly, the authors propose two ways to build the kernel matrix 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.

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.

8.3.6.1 Methodology

Given M tensorial training data and their labels, i.e., dataset $D = \{\boldsymbol{\mathcal{X}}_i, y_i\}_{i=1}^{M}$, where $\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and $y_i \in \{-1, 1\}$, the hyperplane can be defined as

$$f(\boldsymbol{\mathcal{X}}) = \langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}} \rangle + b \tag{8.47}$$

that separates the tensorial data into two classes. Here, $\boldsymbol{\mathcal{W}}$ is the hyperplane weight tensor with the same dimensions as $\boldsymbol{\mathcal{X}}_i$ and b is the bias. Similar to the primal problem in SVM, the corresponding primal optimization problem for (8.47) is derived as

$$\begin{aligned} \min_{\boldsymbol{\mathcal{W}}, b, \xi} \quad & \frac{1}{2}||\boldsymbol{\mathcal{W}}||_F^2 + C\sum_{i=1}^{M}\xi_i \\ \text{subject to} \quad & y_i\big(\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}_i \rangle + b\big) \geq 1 - \xi_i, \\ & \xi_i \geq 0,\ i = 1, \ldots, M. \end{aligned} \tag{8.48}$$

Following the scheme of the kernel trick for conventional SVMs, it is necessary to define a nonlinear feature mapping function $\Phi(\cdot)$ for tensors. Specifically, given a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, it is mapped into the Hilbert space $\mathcal{H}$ by

$$\Phi(\cdot) : \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d} \rightarrow \mathbb{R}^{H_1 \times H_2 \times \cdots \times H_d}. \tag{8.49}$$

The dimension of the projected tensor $\Phi(\boldsymbol{\mathcal{X}})$ can be infinite depending on the feature mapping function $\Phi(\cdot)$. The resulting Hilbert space is then called the *tensor feature*

space and the following model is further developed:

$$\begin{aligned} \min_{\mathcal{W},b,\xi} \quad & \frac{1}{2}||\mathcal{W}||_F^2 + C\sum_{i=1}^{M}\xi_i \\ \text{subject to} \quad & y_i\left(\langle \mathcal{W}, \Phi(\mathcal{X}_i)\rangle + b\right) \geq 1-\xi_i, \\ & \xi_i \geq 0,\ i=1,\ldots,M, \end{aligned} \tag{8.50}$$

with parameter tensor $\mathcal{W} \in \mathbb{R}^{H_1\times H_2\times\cdots\times H_d}$. This model is naturally a linear classifier on the tensor feature space. However, when the classifier is mapped back to the original data space, it is a nonlinear classifier. To obtain the tensor-based kernel optimization model, the dual format of (8.50) is derived, namely

$$\begin{aligned} \min_{\alpha_1,\alpha_2,\cdots,\alpha_M} \quad & \sum_{i=1}^{M}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{M}\alpha_i\alpha_j y_i y_j \langle \Phi(\mathcal{X}_i), \Phi(\mathcal{X}_j)\rangle \\ \text{subject to} \quad & \sum_{i=1}^{M}\alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C,\ i=1,\ldots,M, \end{aligned} \tag{8.51}$$

where α_i are the Lagrange multipliers. The key task 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.

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, K-STTM employs 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 proved 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 [33]). 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 are 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, it is possible to 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 three-way (pixel–pixel–color) tensor. The color mode can be treated differently from the two pixel modes since they contain different kinds of information, as will be exemplified later.

To this end, a TT-based feature mapping approach is proposed in K-STTM. Specifically, all fibers in each TT core are mapped to the feature space through

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

such that

$$\begin{gathered} \phi_i\left(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1})\right) \in \mathbb{R}^{H_i} \\ 1 \le r_i \le R_i, \quad 1 \le r_{i+1} \le R_{i+1}, \quad i = 1, \ldots, d, \end{gathered} \tag{8.52}$$

where $\boldsymbol{\mathcal{X}}^{(i)}$ and R_i are the i-th TT core and TT rank of $TT(\boldsymbol{\mathcal{X}})$, respectively. The fibers of each TT core are vectors as the rank indices (r_i, r_{i+1}) are fixed to specific values, and hence the feature mapping works in the same way as for the conventional SVM. The resulting high-dimensional TT, which is in the tensor feature space, is then represented as $\Phi(TT(\boldsymbol{\mathcal{X}})) \in \mathbb{R}^{H_1 \times H_2 \times \cdots \times H_d}$. Note that $\Phi(TT(\boldsymbol{\mathcal{X}}))$ is still in a TT format with the same TT ranks as $TT(\boldsymbol{\mathcal{X}})$. 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, [37] proposes two approaches for computing the inner product between two mapped TT format data using kernel functions. The first method is called K-STTM-Prod since consecutive multiplication operations are implemented on d fiber inner products, which is consistent with the result of an inner product between two TTs. Assuming $\Phi(TT(\boldsymbol{\mathcal{X}}))$ and $\Phi(TT(\boldsymbol{\mathcal{Y}})) \in \mathbb{R}^{H_1 \times H_2 \times \cdots \times H_d}$ with TT ranks R_i and $\hat{R}_i$, $i = 1, 2, \ldots, d+1$, respectively, their inner product can be computed from

$$\begin{aligned} \left\langle \Phi\left(TT(\boldsymbol{\mathcal{X}})\right), \Phi\left(TT(\boldsymbol{\mathcal{Y}})\right) \right\rangle = & \sum_{r_1=1}^{R_1} \cdots \sum_{r_{d+1}=1}^{R_{d+1}} \sum_{\hat{r}_1=1}^{\hat{R}_1} \cdots \sum_{\hat{r}_{d+1}=1}^{\hat{R}_{d+1}} \\ & \left(\prod_{i=1}^{d} \left\langle \phi_i\left(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1})\right), \phi_i\left(\boldsymbol{\mathcal{Y}}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})\right) \right\rangle \right). \end{aligned} \tag{8.53}$$

Note that (8.53) derives the exact same result as Fig. 8.5 (assuming $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{B}}$) when an identity feature mapping function $\Phi(\cdot)$ is used, namely $\Phi(TT(\boldsymbol{\mathcal{X}}))$ $= TT(\boldsymbol{\mathcal{X}})$. Moreover, since each fiber of a mapped TT core is naturally a vector, the following equation is derived:

$$\left\langle \phi_i\left(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1})\right), \phi_i\left(\boldsymbol{\mathcal{Y}}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})\right) \right\rangle = \boldsymbol{k}_i\left(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1}), \boldsymbol{\mathcal{Y}}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})\right), \tag{8.54}$$

where $\boldsymbol{k}_i(\cdot)$ can be any kernel function used for a standard SVM, such as a Gaussian RBF kernel, polynomial kernel, linear kernel, etc. Combining (8.53) and (8.54), the corresponding TT-based kernel function is obtained:

$$\mathcal{K}\big(TT(\boldsymbol{\mathcal{X}}), TT(\boldsymbol{\mathcal{Y}})\big) = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{d+1}=1}^{R_{d+1}} \sum_{\hat{r}_1=1}^{\hat{R}_1} \cdots \sum_{\hat{r}_{d+1}=1}^{\hat{R}_{d+1}} \left(\prod_{i=1}^{d} \boldsymbol{k}_i\big(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1}), \boldsymbol{\mathcal{Y}}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})\big) \right). \tag{8.55}$$

As mentioned before, in the 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 two spatial modes, while choosing a linear or polynomial kernel $\boldsymbol{k}_3$ for the color mode.

The second method proposed in [37] for constructing a TT kernel function is called the 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. The authors mentioned that this idea is inspired by [32], 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 nonsimilar fibers, which could influence the final result significantly. To this end, the K-STTM-Sum is proposed. Similar to K-STTM-Prod, the corresponding kernel function is obtained as

$$\mathcal{K}\big(TT(\boldsymbol{\mathcal{X}}), TT(\boldsymbol{\mathcal{Y}})\big) = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{d+1}=1}^{R_{d+1}} \sum_{\hat{r}_1=1}^{\hat{R}_1} \cdots \sum_{\hat{r}_{d+1}=1}^{\hat{R}_{d+1}} \left(\sum_{i=1}^{d} \boldsymbol{k}_i\big(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1}), \boldsymbol{\mathcal{Y}}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})\big) \right). \tag{8.56}$$

After defining the TT-based kernel function, the term $\langle \Phi(\boldsymbol{\mathcal{X}}_i), \Phi(\boldsymbol{\mathcal{X}}_j) \rangle$ in (8.51) can be replaced with (8.55) or (8.56), and the final kernel optimization problem based on the TT structure can be derived as

$$\min_{\alpha_1, \alpha_2, \cdots, \alpha_M} \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j \mathcal{K}\big(TT(\boldsymbol{\mathcal{X}}_i), TT(\boldsymbol{\mathcal{X}}_j)\big)$$

$$\text{subject to} \quad \sum_{i=1}^{M} \alpha_i y_i = 0,$$
$$0 \leq \alpha_i \leq C, i = 1, \ldots, M. \tag{8.57}$$

After solving (8.57), the unknown model parameters $\alpha_1, \alpha_2, \ldots, \alpha_M$ can be obtained and the resulting decision function is then represented as

$$f(\boldsymbol{\mathcal{X}}) = \text{sign}\left(\sum_{i=1}^{M} \alpha_i y_i \mathcal{K}\big(TT(\boldsymbol{\mathcal{X}}_i), TT(\boldsymbol{\mathcal{X}})\big) + b\right). \tag{8.58}$$

The training algorithm of the K-STTM-Prod/Sum is described by pseudo-codes in Algorithm 5 using a Gaussian RBF kernel as an example. Hyperparameters can be tuned through a grid search or through cross-validation.

Algorithm 5 K-STTM-Prod/Sum algorithm.

Input: Training dataset $\{\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times \cdots \times I_d}, \ y_i \in \{-1, 1\}\}_{i=1}^{M}$; Validation dataset $\{\boldsymbol{\mathcal{X}}_j \in \mathbb{R}^{I_1 \times \cdots \times I_d}, y_j \in \{-1, 1\}\}_{j=1}^{N}$; 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 $\boldsymbol{\mathcal{X}}_{\text{trainStack}}$; stack the tensors in validation dataset together as $\boldsymbol{\mathcal{X}}_{\text{validStack}}$.
2: Compute the TT approximation $TT(\boldsymbol{\mathcal{X}}_{\text{trainStack}})$ and $TT(\boldsymbol{\mathcal{X}}_{\text{validStack}})$ with the given TT ranks using TT-SVD.
3: **for** C from C_{min} to C_{max} **do**
4: **for** σ from σ_{min} to σ_{max} **do**
5: Construct the K-STTM-Prod kernel matrix (8.55) or the K-STTM-Sum kernel matrix (8.56).
6: Solve (8.57) using the resulting kernel matrix.
7: Compute the classification accuracy on the validation set.
8: **end for**
9: **end for**
10: Find the best C and σ according to the classification accuracy on the validation set.
11: Train the K-STTM with the best C and σ by implementing steps 6 and 7. Thus the Lagrange multipliers $\alpha_1, \alpha_2, \ldots, \alpha_M$ and the bias b are obtained.

We note that the key task for kernelized tensor learning is to define a suitable tensor mapping scheme for the corresponding tensor decomposition. Apart from the demonstrated K-STTM, Ref. [35] proposed a customized kernel mapping scheme for CP decomposition, named DuSK. We therefore introduce it for comparison.

Let the CP decomposition of two tensorial samples $\mathcal{X}, \mathcal{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ be $\mathcal{X} = \sum_{r=1}^{R} \prod_{i=1}^{d} \circ \boldsymbol{x}_r^{(i)}$ and $\mathcal{Y} = \sum_{r=1}^{R} \prod_{i=1}^{d} \circ \boldsymbol{y}_r^{(i)}$, respectively. Ref. [35] defines a kernel mapping scheme $\hat{\Phi}(\cdot)$ for CP format tensors, such that $\hat{\Phi}(CP(\mathcal{X})) \in \mathbb{R}^{H_1 \times H_2 \times \cdots \times H_d}$. Specifically, this is achieved by mapping all vectors $\boldsymbol{x}_r^{(i)} \in \mathbb{R}^{I_i}$ into a high-dimensional feature space $\phi(\boldsymbol{x}_r^{(i)}) \in \mathbb{R}^{H_i}$; therefore,

$$\hat{\Phi} : \sum_{r=1}^{R} \prod_{i=1}^{d} \circ \boldsymbol{x}_r^{(i)} \rightarrow \sum_{r=1}^{R} \prod_{i=1}^{d} \circ \phi\left(\boldsymbol{x}_r^{(i)}\right). \tag{8.59}$$

After mapping the CP factorization of the tensorial sample into the high-dimensional tensorial feature space, the kernel function is simply the standard inner product of tensors in that feature space, and it can be written as

$$\begin{aligned} \hat{\mathcal{K}}\left(CP(\mathcal{X}), CP(\mathcal{Y})\right) &= \hat{\mathcal{K}}\left(\sum_{r=1}^{R} \prod_{i=1}^{d} \circ \boldsymbol{x}_r^{(i)}, \sum_{r=1}^{R} \prod_{i=1}^{d} \circ \boldsymbol{y}_r^{(i)}\right) \\ &= \sum_{r_1}^{R} \sum_{r_2}^{R} \prod_{i=1}^{d} \boldsymbol{k}\left(\boldsymbol{x}_{r_1}^{(i)}, \boldsymbol{y}_{r_2}^{(i)}\right), \end{aligned} \tag{8.60}$$

where $\boldsymbol{k}(\cdot)$ can be any vector-based kernel function. With (8.60), we can derive the kernel optimization problem based on CP decomposition as follows:

$$\begin{aligned} \min_{\alpha_1, \alpha_2, \cdots, \alpha_M} & \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j \hat{\mathcal{K}}\left(CP(\mathcal{X}_i), CP(\mathcal{X}_j)\right) \\ \text{subject to} \quad & \sum_{i=1}^{M} \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C, \, i = 1, \ldots, M. \end{aligned} \tag{8.61}$$

The comparison between DuSK and K-STTM is implemented in the K-STTM paper [37], and we also showcase the results in the example part of this section.

8.3.6.2 Kernel validity of K-STTM

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, it is necessary to show the validity of K-STTM-Prod and K-STTM-Sum. In this section, we provide Theorem 8.1 for this purpose. 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 K-STTM does it is by first stacking all the d-way samples and then computing a TT decomposition on the resulting $(d+1)$-way tensor directly. By doing so, all

TT-based training samples have the same TT ranks. That means R_i is equal to $\hat{R}_i$, $i = 1, \ldots, d+1$, for all the TT-based training samples, which is also assumed in Theorem 8.1 and its proof.

Before proving Theorem 8.1, we first give the necessary lemma below.

Lemma 8.1. *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 result in a symmetric and positive-semi-definite matrix.*

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

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

for every nonzero column vector $\boldsymbol{u} \in \mathbb{R}^n$. For the summation case, obviously we can conclude that

$$(\boldsymbol{A} + \boldsymbol{B})^T = \boldsymbol{A} + \boldsymbol{B}, \quad \boldsymbol{u}^T (\boldsymbol{A} + \boldsymbol{B})\boldsymbol{u} \geq 0,$$

namely $\boldsymbol{A} + \boldsymbol{B}$ is still symmetric and positive-semi-definite. For the Hadamard product case, we refer to the Schur product theorem [28] and we can easily obtain

$$\boldsymbol{u}^T (\boldsymbol{A} \odot \boldsymbol{B})\boldsymbol{u} \geq 0,$$

for every nonzero 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 depict Theorem 8.1 and its proof here.

Theorem 8.1. *Given a tensorial training dataset $\{\boldsymbol{\mathcal{X}}_i\}_{i=1}^M$, where $\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, and assumed TT ranks $R_1, \ldots, R_{d+1}$, 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 show the kernel function validity of K-STTM-Prod. For any tensor $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} \in \{\boldsymbol{\mathcal{X}}_1, \boldsymbol{\mathcal{X}}_2, \ldots, \boldsymbol{\mathcal{X}}_M\}$, they are first decomposed into their TT formats, namely $TT(\boldsymbol{\mathcal{X}})$ and $TT(\boldsymbol{\mathcal{Y}})$, after which Eq. (8.55) is applied. Assuming all the indices over $\sum$ and $\prod$, namely $r_1, \ldots, r_{d+1}, \hat{r}_1, \ldots, \hat{r}_{d+1}$ and i, are fixed, (8.55) can be written as

$$\boldsymbol{\mathcal{K}}\big(TT(\boldsymbol{\mathcal{X}}), TT(\boldsymbol{\mathcal{Y}})\big) = \boldsymbol{k}_i\big(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1}), \boldsymbol{\mathcal{Y}}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})\big). \tag{8.62}$$

As mentioned before, $\boldsymbol{k}_i(\cdot, \cdot)$ can be any valid kernel function used for a standard SVM. Therefore, the kernel matrix constructed by (8.62) is symmetric and positive-semi-definite. When only the indices over $\sum$, namely $r_1, \ldots, r_{d+1}, \hat{r}_1, \ldots, \hat{r}_{d+1}$, are fixed, (8.55) can be written as the following product kernel:

$$\boldsymbol{\mathcal{K}}\big(TT(\boldsymbol{\mathcal{X}}), TT(\boldsymbol{\mathcal{Y}})\big) = \left(\prod_{i=1}^{d} \boldsymbol{k}_i\big(\boldsymbol{\mathcal{X}}^{(i)}(r_i, :, r_{i+1}), \boldsymbol{\mathcal{Y}}^{(i)}(\hat{r}_i, :, \hat{r}_{i+1})\big)\right). \tag{8.63}$$

The kernel matrix constructed by (8.63) can be regarded as Hadamard products of the d valid kernel matrices constructed by (8.62) when $i = 1, \ldots, d$. Since the matrix constructed by (8.62) is symmetric and positive-semi-definite, according to Lemma 8.1, the matrix constructed by (8.63) is also symmetric and positive-semi-definite.

Similarly, we note that the kernel matrix constructed by (8.55) can be regarded as the summation of $R_1 \times \ldots \times R_{d+1} \times \hat{R}_1 \times \ldots \times \hat{R}_{d+1}$ kernel matrices constructed by (8.63) when $r_1, \ldots, r_{d+1}, \hat{r}_1, \ldots, \hat{r}_{d+1}$ are varied from 1 to their corresponding maximum values. According to Lemma 8.1, we conclude that the kernel matrix constructed by (8.55) 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 (8.56) can be regarded as the summation of $R_1 \times \ldots \times R_{d+1} \times \hat{R}_1 \times \ldots \times \hat{R}_{d+1} \times d$ kernel matrices constructed by (8.62) when $r_1, \ldots, r_{d+1}, \hat{r}_1, \ldots, \hat{r}_{d+1}$ and i are varied from 1 to their corresponding maximum values. According to Lemma 8.1, the kernel matrix constructed by (8.56) is symmetric and positive-semi-definite. Therefore, K-STTM-Sum is a valid kernel function. □

8.3.6.3 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 $O(dIR^2 + MI_dR_d)$, where R is the maximum TT rank of $R_i, i = 1, \ldots, d-1$. 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, $i = 1, 2, ...d$. As for the computational complexity of K-STTM-Prod and K-STTM-Sum, the overall results of them are similar if neglecting low-order terms. And their kernel matrix computation complexities are $O(dIR^4 + M^2I_dR_d^2)$ if we employ the accelerated computation approach as proposed in [37], where I and R are the maximum values of I_i and R_i, $i = 1, 2, \ldots, d-1$, respectively. Therefore, the K-STTM algorithm is more efficient than its vector counterpart as the computation complexity is reduced from exponential to polynomial.

8.3.6.4 Examples

Ref. [37] considers three high-dimensional fMRI datasets, namely, the StarPlus fMRI dataset[4], the CMU Science 2008 fMRI dataset (CMU2008) [34], and the ADNI fMRI dataset[5], to evaluate the classification performance of K-STTM. An fMRI image is essentially a three-way tensor. For the compared methods, apart from SVM, STM, STuM, and STTM, another kernelized STM method called DuSK [35] is further in-

[4] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-81/www/.
[5] http://adni.loni.usc.edu/.

Table 8.12 Classification accuracies (%) of different methods for different subjects in StarPlus fMRI datasets.

Subject	SVM	STM	STuM	STTM	DuSK	K-STTM-Prod	K-STTM-Sum
04799	50.00	36.67	35.83	39.61	47.50	**68.33**	66.67
04820	50.00	43.33	35.00	45.83	46.67	**70.00**	62.50
04847	50.00	38.33	17.50	47.50	53.33	**65.00**	**65.00**
05675	50.00	37.50	30.83	35.00	55.00	**60.00**	**60.00**
05680	50.00	38.33	39.17	40.00	64.17	73.33	**75.00**
05710	50.00	40.00	30.00	43.33	54.16	**59.17**	58.33

cluded. DuSK is a kernelized STM using the CP decomposition. Through introducing the kernel trick, it can deal with nonlinear classification tasks.

For the StarPlus dataset, each fMRI image is of dimensions $64 \times 64 \times 8$. The whole dataset contains the brain images of six human subjects, and each subject has 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. These fMRI images are randomly separated into training, validation, and test sets. The classification results are listed in Table 8.12. Due to the very high-dimensional and sparse data, SVM fails to find a good hyperparameter setting and classifies all test samples wrongly into one class. Since fMRI data are very complicated, those linear classifiers, namely STM, STuM, and STTM, cannot achieve acceptable performance, and the classification accuracies of them are all lower than 50%. The two K-STTM methods still achieve the highest classification accuracy on all human subjects.

The CMU2008 dataset shows the brain activities associated with the meanings of nouns. During the data collection period, the subjects were asked to view 60 different word-pictures from 12 semantic categories. There are five pictures in each category and each image is shown to the subject six times. Therefore, 30 fMRI images are collected for each semantic category, and each fMRI image is of dimensions $51 \times 61 \times 23$. Considering the extremely small number of samples in each category, [37] used the experimental settings in [36] which combine two similar categories into an integrated class. Specifically, [37] combines categories *animal* and *insect* as class **Animals** and categories *tool* and *furniture* as class **Tools**. By doing so, there are 60 samples in both **Animals** and **Tools** classes. Table 8.13 shows the binary classification results of different models. We note that the classification accuracies of SVM on four subjects are lower than 50%. The linear models, namely STM, STuM, STTM, and TT classifier, can only achieve acceptable performance on a few subjects. Because the dimensions of the data are very high, DuSK fails to find a good CP rank in an acceptable time and cannot achieve a good classification accuracy. The two K-STTM methods still achieve the best classification results in all subjects.

The ANDI fMRI dataset is collected from the Alzheimer's Disease Neuroimaging Initiative. It contains the resting-state fMRI images of 33 subjects. The subjects include patients (with mild cognitive impairment [MCI] or Alzheimer's disease [AD])

Table 8.13 Classification accuracies (%) of different methods for different subjects in CMU2008 fMRI datasets.

Subject	SVM	STM	STuM	STTM	DuSK	K-STTM-Prod	K-STTM-Sum
#1	68.00	66.00	68.00	66.00	48.00	**70.00**	**70.00**
#2	52.00	50.00	58.00	68.00	54.00	74.00	**84.00**
#3	50.00	60.00	58.00	64.00	58.00	66.00	**70.00**
#4	50.00	60.00	58.00	56.00	52.00	**76.00**	72.00
#5	56.00	58.00	64.00	66.00	44.00	**72.00**	**72.00**
#6	44.00	60.00	46.00	46.00	54.00	**70.00**	**70.00**
#7	50.00	52.00	48.00	52.00	52.00	68.00	**72.00**

Table 8.14 Classification accuracies (%) of different methods in the ADNI fMRI dataset.

	SVM	STM	STuM	STTM	DuSK	K-STTM-Prod	K-STTM-Sum
ADNI fMRI	49.33	50.00	38.00	53.67	54.94	**64.00**	62.33

and normal controls. Overall there are 33 fMRI images and each image has dimensions $61 \times 73 \times 61$. These fMRI images are separated into two classes. The positive class includes normal controls, while the negative class includes patients with MCI or AD. Table 8.14 lists the classification results of different methods. We can observe that the performance of SVM, STM, and STuM is still not good. STTM and DuSK achieve a slightly better performance than random classification. The two K-STTM methods achieve the best accuracy of all compared methods.

8.3.6.5 Conclusion

Ref. [37] has proposed a TT-based kernel trick for the first time and devised a K-STTM. Assuming a low-rank TT as the prior structure of multidimensional data, the authors first define a corresponding feature mapping scheme that keeps the TT structure in the feature space. Furthermore, two kernel function construction schemes are proposed with consideration of consistency with the TT inner product and the preservation of information, respectively. Moreover, it is possible to apply different kernel mappings on the tensor modes with different characteristics.

8.4 Tensorial extension of logistic regression

The idea for extending vector-based LR is similar to the case in SVM. In this section, we briefly introduce two tensorial extensions of LR, namely rank-1 LR [38] and logistic tensor regression [39]. Specifically, rank-1 LR replaces the weight vector in LR with a rank-1 tensor, while logistic tensor regression assumes a general CP format weight tensor.

Algorithm 6 Rank-1 LR algorithm.

Input: Training dataset $\{\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times \cdots \times I_d}, y_i \in \{-1, 1\}\}_{i=1}^{M}$.
Output: LR model parameters $\boldsymbol{w}^{(j)}|_{l=1}^{d}$ and bias b.

1: Initialize $\boldsymbol{w}|_{l=1}^{d}$ randomly.
2: Repeat steps 3–5 iteratively until convergence.
3: **for** $j = 1, \ldots, d$ **do**
4: Derive $\boldsymbol{w}^{(j)}$ by optimizing (8.67).
5: **end for**

8.4.1 Rank-1 logistic regression

Consider a training dataset $D=\{\boldsymbol{\mathcal{X}}_i, y_i\}_{i=1}^{M}$, where tensors $\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and $y_i \in \{-1, +1\}$. The general tensorized LR model is based on the following expression for the conditional probabilities:

$$p(y_i = +1|\boldsymbol{\mathcal{W}}, b, \boldsymbol{\mathcal{X}}_i) = \frac{1}{1 + \exp(-\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}_i \rangle - b)}, \tag{8.64}$$

where $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and $b \in \mathbb{R}$ are the parameter tensor and the bias of the regression model. The corresponding maximum log-likelihood problem can be written as

$$\min_{\boldsymbol{\mathcal{W}}, b} \quad \sum_{i=1}^{M} \log\big(1 + \exp\big(-y_i\big(\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}_i \rangle + b\big)\big)\big). \tag{8.65}$$

In rank-1 LR models, the parameter tensor weight $\boldsymbol{\mathcal{W}}$ is assumed to be a rank-1 tensor, namely, (8.65) can be rewritten as

$$\min_{\boldsymbol{w}^{(l)}|_{l=1}^{d}, b} \quad \sum_{i=1}^{M} \log\left(1 + \exp\left(-y_i\left(\boldsymbol{\mathcal{X}}_i \prod_{1 \le l \le d} \times_l \boldsymbol{w}^{(l)} + b\right)\right)\right), \tag{8.66}$$

where $\boldsymbol{\mathcal{W}} = \boldsymbol{w}^{(1)} \circ \boldsymbol{w}^{(2)} \circ \cdots \circ \boldsymbol{w}^{(d)}$. The optimization scheme is still an alternating optimization procedure. In particular, we solve for $\boldsymbol{w}^{(j)}$ at each iteration while keeping the parameters $\boldsymbol{w}^{(k)}|_{k=1,k\neq j}^{d}$ fixed. Therefore, each suboptimization problem is written as

$$\min_{\boldsymbol{w}^{(j)}, b} \quad \sum_{i=1}^{M} \log\big(1 + \exp\big(-y_i\big(\big(\boldsymbol{w}^{(j)}\big)^T \hat{\boldsymbol{x}} + b\big)\big)\big), \tag{8.67}$$

where $\hat{\boldsymbol{x}} = \boldsymbol{\mathcal{X}}_i \prod_{1 \le l \le d}^{l \neq j} \times_l \boldsymbol{w}^{(l)}$. We note that (8.67) has the same optimization format as traditional vector-based LR. The overall optimization procedure is summarized in Algorithm 6.

Table 8.15 Test accuracy (%) under different training sample sizes of the Indian Pines dataset.

Method	Training sample size			
	50	100	150	200
Linear SVM	68.25	75.39	80.36	79.84
LR	63.44	67.64	72.64	73.52
Rank-1 LR	**75.22**	**80.41**	**82.65**	**83.76**

Table 8.16 Test accuracy (%) under different training sample sizes of the Pavia University dataset.

Method	Training sample size			
	50	100	150	200
Linear SVM	79.62	83.90	87.00	87.04
LR	71.82	77.89	82.19	82.27
Rank-1 LR	**84.16**	**86.69**	**88.04**	**88.76**

8.4.1.1 Examples

Ref. [38] employs rank-1 LR to classify hyperspectral data. A hyperspectral image is represented as a 3D tensor of dimensions $p_1 \times p_1 \times p_3$, where p_1 and p_2 correspond to the height and width of the image and p_3 to the spectral bands. Specifically, the North-western Indiana and Pavia University datasets are used [38]. To evaluate the performance of rank-1 LR under different number of training samples, 50, 100, 150, and 200 samples are randomly selected from each class, respectively, to train the model and the remaining data are used to test the performance. The rank-1 LR is compared with traditional vector-based LR and linear SVM.

Tables 8.15 and 8.16 list the classification accuracies on the two datasets. In both datasets and in all cases, the tensor-based model outperforms linear SVMs and vector-based LR, despite the fact that it employs the smallest number of parameters.

8.4.2 Logistic tensor regression

The issue in rank-1 LR is similar to the case in STM, namely the setting of rank-1 tensor weight is oversimplified such that the trained model is not powerful enough to classify some complicated data. Therefore, assuming a general tensor format for weight parameter is a better choice. Here we demonstrate logistic tensor regression (LTR) [39], which employs the CP tensor format to replace the rank-1 weight tensor setting in rank-1 LR.

Assume the weight tensor $\mathcal{W}$ can be represented as a CP tensor format, namely we have (8.2). Based on (8.65), we obtain the following equations according to the

Algorithm 7 LTR algorithm.

Input: Training dataset $\{\boldsymbol{\mathcal{X}}_i \in \mathbb{R}^{I_1 \times \cdots \times I_d}, y_i \in \{-1, 1\}\}_{i=1}^{M}$, CP rank for weight tensor.

Output: LTR model parameters $\boldsymbol{U}^{(j)}|_{l=1}^{d}$ and bias b.

1: Initialize $\boldsymbol{U}^{(j)}|_{l=1}^{d}$ randomly.
2: Repeat steps 3–5 iteratively until convergence.
3: **for** $j = 1, \ldots, d$ **do**
4: derive $\boldsymbol{U}^{(j)}$ by optimizing (8.70)
5: **end for**

derivation from (8.34) to (8.36):

$$\langle \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{X}}_i \rangle = Tr\big[\boldsymbol{W}_{(j)} \boldsymbol{X}_{(j)i}^T\big] = Tr\big[\boldsymbol{U}^{(j)} \big(\boldsymbol{U}^{(-j)}\big)^T \boldsymbol{X}_{(j)i}^T\big]. \tag{8.68}$$

Therefore, (8.65) can be rewritten as

$$\min_{\boldsymbol{U}^{(j)}, b} \quad \sum_{i=1}^{M} \log\big(1 + \exp\big(-y_i \big(Tr\big[\boldsymbol{U}^{(j)} \big(\boldsymbol{U}^{(-j)}\big)^T \boldsymbol{X}_{(j)i}^T\big] + b\big)\big)\big). \tag{8.69}$$

Let $\widetilde{\boldsymbol{X}}_{(j)i} = \boldsymbol{X}_{(j)i} \boldsymbol{U}^{(-j)}$. Since $Tr[\boldsymbol{U}^{(j)} \widetilde{\boldsymbol{X}}_{(j)i}^T] = (vec(\boldsymbol{U}^{(j)}))^T vec(\widetilde{\boldsymbol{X}}_{(j)i})$, we can rewrite (8.69) into its vector format as follows:

$$\min_{\boldsymbol{U}^{(j)}, b} \quad \sum_{i=1}^{M} \log\big(1 + \exp\big(-y_i \big(\big(vec\big(\boldsymbol{U}^{(j)}\big)\big)^T vec(\widetilde{\boldsymbol{X}}_{(j)i}) + b\big)\big)\big). \tag{8.70}$$

By doing so, any optimization method applied to traditional LR can also be employed to train an LTR. The overall training scheme is still an alternating optimization procedure, as depicted in Algorithm 7. In each iteration, we solve for $\boldsymbol{U}^{(j)}$ while keeping all other $\boldsymbol{U}^{(j)}|_{k=1,k\neq j}^{d}$ fixed.

8.4.2.1 Examples

In [39], the performance of an LTR is evaluated through two public datasets, namely the FG-NET facial images dataset [40] and the Carnegie Mellon University's Graphics Lab human motion capture database [41]. Two metrics are employed to evaluate the LTR classification performance. The first one is the area under the ROC curve (AUC) [42]. More specifically, the MacroAUC (average on AUC of all the classes) and the MicroAUC (the global calculation of AUC regardless of classes) are used. The second one is the harmonic mean of precision and recall, called F_1 score [42]. The Macro F_1 (average on F_1 scores of all the classes) and the Micro F_1 (the global calculation of F_1 regardless of classes) are presented.

The rank of the weight tensor is determined by grid search. Tables 8.17 and 8.18 show the classification performance of LTR and other methods. It is observed that

Table 8.17 Comparison on facial images.

Method	MacroAUC	MicroAUC	Macro F_1	Micro F_1
LTR	**0.7482**	**0.8692**	**0.5697**	**0.5717**
SVR [45]	0.4956	0.6731	0.4073	0.3631
BLR [44]	0.5702	0.7984	0.4417	0.5231
hrTRR [43]	0.6153	0.8152	0.4568	0.4321
hrSTR [43]	0.6916	0.8250	0.5213	0.5037
orTRR [43]	0.5780	0.7756	0.3513	0.3231
orSTR [43]	0.5184	0.7555	0.3913	0.3114

Table 8.18 Comparison on motion data.

Method	MacroAUC	MicroAUC	Macro F_1	Micro F_1
LTR	**0.6976**	**0.9080**	**0.5218**	**0.5031**
SVR [45]	0.5413	0.7231	0.4006	0.4268
BLR [44]	0.5609	0.7527	0.4332	0.4367
hrTRR [43]	0.6708	0.8344	0.4359	0.4135
hrSTR [43]	0.6684	0.8686	0.4965	0.4631
orTRR [43]	0.6594	0.8894	0.5063	0.4792
orSTR [43]	0.5517	0.7932	0.3993	0.4010

all of the tensorial approaches outperform their vector-based counterparts in terms of the two kinds of evaluation metrics. Moreover, LTR outperforms the linear tensor regression methods in the task of classification.

8.5 Conclusion

Many real-world data appear in a matrix or tensor format. In such circumstances, extending the vector-based machine learning algorithms to their tensorial format has recently attracted significant interest in the machine learning and data mining communities since tensor algorithms can naturally utilize the multiway structure of the original tensor data, which is believed to be useful in many machine learning applications. In this chapter, we have reviewed some tensorial extensions of two traditional classifiers, namely SVM and LR. The classification performance enhancement is observed in various tensorial data classification tasks. These advantages would be more obvious in dealing with small-sample high-dimensional tensor classification problems due to the fact that collecting labeled data can be extremely expensive and time consuming in many practical scenarios.

There are still some open problems in this area. First, for the determination of tensor ranks, grid search is still the mainstream scheme for finding suitable tensor ranks, which is computationally demanding and does not guarantee optimality. Some works [46] have tried to explore the possibility of employing a probabilistic model to determine the tensor ranks automatically. Second, the training algorithms for tensor-based classifiers are mostly based on the alternating optimization procedure. Although the algorithmic convergence is guaranteed, it is still time consuming. A more efficient training scheme is still highly desired.

References

[1] Vladimir Vapnik, The Nature of Statistical Learning Theory, Springer Science & Business Media, 2013.

[2] Raymond E. Wright, Logistic regression, 1995.

[3] Johan Håstad, Tensor rank is NP-complete, in: International Colloquium on Automata, Languages, and Programming, Springer, Berlin, Heidelberg, 1989.

[4] Dacheng Tao, et al., Supervised tensor learning, in: Fifth IEEE International Conference on Data Mining (ICDM'05), IEEE, 2005.

[5] Shuicheng Yan, et al., Multilinear discriminant analysis for face recognition, IEEE Transactions on Image Processing 16 (1) (2006) 212–220.

[6] Dacheng Tao, et al., Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (7) (2006) 1088–1099.

[7] Jing Li, et al., Multitraining support vector machine for image retrieval, IEEE Transactions on Image Processing 15 (11) (2006) 3597–3601.

[8] Lior Wolf, Hueihan Jhuang, Tamir Hazan, Modeling appearances with low-rank SVM, in: 2007 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2007.

[9] Deng Cai, et al., Support tensor machines for text categorization, 2006.

[10] A. Kale, A. Sundaresan, A.N. Rajagopalan, N.P. Cuntoor, A.K. Roy-Chowdhury, V. Kruger, R. Chellappa, Identification of humans using gait, IEEE Transactions on Image Processing 13 (9) (2004) 1163–1173.

[11] Jinguang Han, Bir Bhanu, Individual recognition using gait energy image, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (2) (2005) 316–322.

[12] Vin De Silva, Lek-Heng Lim, Tensor rank and the ill-posedness of the best low-rank approximation problem, SIAM Journal on Matrix Analysis and Applications 30 (3) (2008) 1084–1127.

[13] Carla D. Martin, The rank of a $2\times2\times2$ tensor, Linear and Multilinear Algebra 59 (8) (2011) 943–950.

[14] Zhifeng Hao, et al., A linear support higher-order tensor machine for classification, IEEE Transactions on Image Processing 22 (7) (2013) 2911–2920.

[15] Irene Kotsia, Weiwei Guo, Ioannis Patras, Higher rank support tensor machines for visual recognition, Pattern Recognition 45 (12) (2012) 4192–4203.

[16] Tae-Kyun Kim, Roberto Cipolla, Canonical correlation analysis of video volume tensors for action categorization and detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (8) (2008) 1415–1428.

[17] Saad Ali, Mubarak Shah, Human action recognition in videos using kinematic features and multiple instance learning, IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (2) (2008) 288–303.
[18] Andrzej Cichocki, et al., Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: perspectives and challenges part 1, arXiv preprint, arXiv:1609.00893, 2016.
[19] Yiming Yang, Xin Liu, A re-examination of text categorization methods, in: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 1999.
[20] Ulrich Schollwöck, The density-matrix renormalization group in the age of matrix product states, Annals of Physics 326 (1) (2011) 96–192.
[21] Konstantinos Rapantzikos, Yannis Avrithis, Stefanos Kollias, Dense saliency-based spatiotemporal feature points for action recognition, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2009.
[22] Alireza Fathi, Greg Mori, Action Recognition by Learning Mid-Level Motion Features, 2008 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2008.
[23] Haiping Lu, Konstantinos N. Plataniotis, Anastasios N. Venetsanopoulos, MPCA: Multilinear principal component analysis of tensor objects, IEEE Transactions on Neural Networks 19 (1) (2008) 18–39.
[24] Sudeep Sarkar, et al., The humanID gait challenge problem: data sets, performance, and analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (2) (2005) 162–177.
[25] Nikolaos V. Boulgouris, Konstantinos N. Plataniotis, Dimitrios Hatzinakos, Gait recognition using linear time normalization, Pattern Recognition 39 (5) (2006) 969–979.
[26] Alex Krizhevsky, Geoffrey Hinton, Learning multiple layers of features from tiny images, 2009, p. 7.
[27] Christian Schuldt, Ivan Laptev, Barbara Caputo, Recognizing human actions: a local SVM approach, in: Proceedings of the 17th International Conference on Pattern Recognition, vol. 3, IEEE, 2004.
[28] Jssai Schur, Bemerkungen zur Theorie der beschränkten Bilinearformen mit unendlich vielen Veränderlichen, Journal für die reine und angewandte Mathematik (Crelles Journal) 1911 (140) (1911) 1–28.
[29] Irene Kotsia, Ioannis Patras, Support Tucker machines, in: CVPR 2011, IEEE, 2011.
[30] Yann LeCun, et al., Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[31] Cong Chen, et al., A support tensor train machine, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019.
[32] Lynn Houthuys, Johan AK Suykens, Tensor learning in multi-view kernel PCA, in: International Conference on Artificial Neural Networks, Springer, Cham, 2018.
[33] Ivan V. Oseledets , SIAM Journal on Scientific Computing 33 (5) (2011) 2295–2317.
[34] Tom M. Mitchell, et al., Predicting human brain activity associated with the meanings of nouns, Science 320 (5880) (2008) 1191–1195.
[35] Lifang He, et al., Dusk: a dual structure-preserving kernel for supervised tensor learning with applications to neuroimages, in: Proceedings of the 2014 SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics, 2014.
[36] Kittipat Kampa, et al., Sparse optimization in feature selection: application in neuroimaging, Journal of Global Optimization 59.2–3 (2014) 439–457.
[37] Cong Chen, et al., Kernelized support tensor train machines, arXiv preprint, arXiv:2001.00360, 2020.

[38] Konstantinos Makantasis, et al., Tensor-based classification models for hyperspectral data analysis, IEEE Transactions on Geoscience and Remote Sensing 56 (12) (2018) 6884–6898.
[39] Tan Xu, et al., Logistic tensor regression for classification, in: International Conference on Intelligent Science and Intelligent Data Engineering, Springer, Berlin, Heidelberg, 2012.
[40] A. Agarwal, B. Triggs, I. Rhone-Alpes, F. Montbonnot, The FG-NET aging database, http://www.fgnet.rsunit.com, 2010.
[41] Tommaso Piazza, et al., Predicting missing markers in real-time optical motion capture, in: 3D Physiological Human Workshop, Springer, Berlin, Heidelberg, 2009.
[42] Tom Fawcett, An introduction to ROC analysis, Pattern Recognition Letters 27 (8) (2006) 861–874.
[43] Weiwei Guo, Irene Kotsia, Ioannis Patras, Tensor learning for regression, IEEE Transactions on Image Processing 21 (2) (2011) 816–827.
[44] Sean M. O'brien, David B. Dunson, Bayesian multivariate logistic regression, Biometrics 60 (3) (2004) 739–746.
[45] Alex J. Smola, Bernhard Schölkopf, A tutorial on support vector regression, Statistics and Computing 14 (3) (2004) 199–222.
[46] Le Xu, et al., Learning tensor train representation with automatic rank determination from incomplete noisy data, arXiv preprint, arXiv:2010.06564, 2020.