



兰州大学

多元统计分析课程论文

论文题目（中文） 基于张量分解的神经影像分类模型

论文题目（英文） Neuroimaging Classification Model

Based on Tensor Decomposition

学生姓名 王一鑫

指导教师 李周平

学 院 萃英学院

专 业 数学

年 级 2022 级

二〇二五年六月

基于张量分解的神经影像分类模型

中文摘要

本文提出了一种基于张量分解的分类模型，并探讨了该模型在高维神经影像数据分析中的应用。首先，介绍了张量的基本概念及其分解方法，包括 Tucker 分解和 PARAFAC 分解，并详细讨论了张量在处理神经影像数据中的优势。然后，本文结合贝叶斯框架，提出了基于不同损失函数（SVM 损失与逻辑回归损失）的贝叶斯张量分类方法，并通过 MCMC 算法进行后验推断。在模拟研究中，本文对比了不同模型的性能，结果表明，提出的贝叶斯张量分类模型在系数估计和分类性能上优于传统的 Lasso 逻辑回归和 L1-norm SVM 方法。此外，本文还在脑肿瘤 MRI 图像分类任务中验证了该模型的有效性，结果表明，BT-SVM 模型表现最佳，展示了该模型在医学影像分析中的潜力。

关键词：张量分解；贝叶斯建模；神经影像；MCMC 算法；支持向量机；逻辑回归

Neuroimaging Classification Model Based on Tensor Decomposition

Abstract

This essay explores the history of studies in analytical philosophy in China since the beginning of the last century, by dividing into three phases. It shows that, in these phases, analytic philosophy was always at a disadvantage in confronting serious challenges coming from both Chinese traditional philosophy and modern philosophical trends. The authors argue that Chinese philosophers have both done preliminary studies and offered their own analyses of various problems as well as some new applications of analytic philosophy especially in the latest period. Meanwhile, Chinese traditional philosophy was always trying to adjust its cultural mentality in the struggle with analytic philosophy, and accommodated in its own way the rationalistic spirit and scientific method represented in analytic philosophy.

Keywords: analytical philosophy; Chinese philosophers; philosophical analysis; dialogue in philosophy.

目 录

中文摘要	2
英文摘要	3
第一章 绪 论	1
1.1 二级标题	1
第二章 建模方法	2
2.1 张量及其分解	2
2.2 基于不同损失函数的分类模型	3
2.2.1 支持向量机 (SVM)	4
2.2.2 逻辑回归 (Logistic Regression)	5
2.3 先验估计	6
2.4 后验推断的 MCMC 算法	7
2.4.1 MCMC 算法简介 ^[g]	7
2.4.2 吉布斯采样 (Gibbs Sampling) 算法简介 ^[g]	8
2.4.3 基于 SVM 损失的贝叶斯张量模型	9
2.4.4 基于 Logistic 损失的贝叶斯张量模型	11
2.4.5 参数设置	11
第三章 模拟数据研究	13
3.1 数据生成	13
3.2 模型评估	15
3.3 MRI 脑肿瘤分类	20
参考文献	22
附 录	22

图 目 录

图 2.1	标量, 向量, 矩阵和三阶张量.....	2
图 2.2	Tucker 分解.....	3
图 2.3	PARAFAC 分解	3
图 2.4	Hinge Loss	4
图 2.5	Logistic Loss	5
图 3.1	Scene 1	15
图 3.2	Scene 2	15
图 3.3	Scene 3	15
图 3.4	Scene 4	15
图 3.5	估计的张量系数 (第一排由 BT-SVM 模型估计第二排由 BT-LR 模型估计)	20
图 3.6	残差图	21

表 目 录

表 3.1	标签由 SVM 损失生成时不同模型在每种场景下的表现.....	18
表 3.2	标签由逻辑损失生成时不同模型在每种场景下的表现.....	19
表 3.3	不同模型对脑肿瘤影像的分类性能	21

第一章 绪 论

这里是绪论，也可以说是引言，在 LZUThesis.clc 里面改，引言写什么呢，先凑字数，注意啊，段落在 latex 里面是要空一行的，不要简单一个回车，编译过程中警告内容无需管

1.1 二级标题

绪论其实也可以有二级标题，要不然，论文要求：“包括毕业论文（设计）的研究目的、意义、范围、研究设想、方法、实验设计、选题依据等；还包括毕业论文（设计）研究领域的历史回顾，文献追溯，理论分析等内容”全部写成一堆不成？

$$E = mc^2 \quad (1)$$

第二章 建模方法

在本文中，我们使用了一种基于数据增强的贝叶斯分类建模方法。该方法使用基于张量的表示方法，根据成像协变量对二元结果进行建模。我们考虑了两种不同的数据增强方案，检验了两种不同的贝叶斯分类器的性能，分别是支持向量机和逻辑回归模型。

虽然这些分类器已在文献中得到广泛应用，但其重点是使用忽略图像中空间结构的非结构化协变量。主要思路是通过对系数张量进行低秩分解，并利用数据增强技术和贝叶斯思想提出一个全新的神经影像分类方法。

2.1 张量及其分解

基于张量的模型具有多方面的优势，已被公认为一种有前途的神经成像数据建模方法。张量天然继承了多维结构，可以表示复杂的数据结构，如大脑区域的空间特征。此外，基于张量的技术还能实现降维，这对神经成像数据特别有用，能解决统计建模中 $p \gg n$ 的难题。

张量是一个多维数组， d 阶张量是一个具有 d 个维度的数组，表示为：

$$\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$$

其中， d 是张量的阶数，而 n_k 是第 k 阶的维数。

张量 \mathcal{X} 的元素通过 (i_1, i_2, \dots, i_d) 的索引来表示，元素表示为：

$$x_{i_1 i_2 \dots i_d} \quad i_k \in \{1, 2, \dots, n_k\} \quad k \in \{1, 2, \dots, d\}$$

图 2.1^a 展示了标量，向量，矩阵和三阶张量。

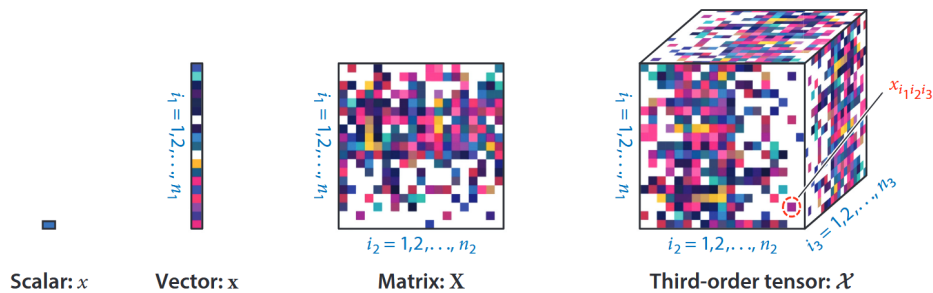


图 2.1 标量，向量，矩阵和三阶张量

张量分解是一种将高维张量分解为若干低维因子组合的技巧，其中一种为 Tucker 分解，它是张量的高阶主成分分析（PCA），将张量分解为核心张量与各模式因子矩阵的乘积。具

体地, 给定一个张量 $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$, Tucker 分解表示为:

$$\mathcal{X} \approx \mathcal{C} \times_1 \mathbf{Q}^1 \times_2 \mathbf{Q}^2 \cdots \times_d \mathbf{Q}^d = \sum_{j_1=1}^{m_1} \sum_{j_2=1}^{m_2} \cdots \sum_{j_d=1}^{m_d} c_{j_1 j_2 \dots j_d} \mathbf{q}_{j_1}^1 \circ \mathbf{q}_{j_2}^2 \circ \cdots \circ \mathbf{q}_{j_d}^d \quad (2)$$

其中, $\mathcal{C} \in \mathbb{R}^{m_1 \times m_2 \times \dots \times m_d}$ 是核心张量, $\mathbf{Q}^k \in \mathbb{R}^{n_k \times m_k}$ 是因子矩阵。图 2.2^a 给出了 Tucker 分解的图示。特别地, 当 $m_1 = m_2 = \dots = m_d = R$, 且核心张量限制为对角型时被称为 PARAFAC

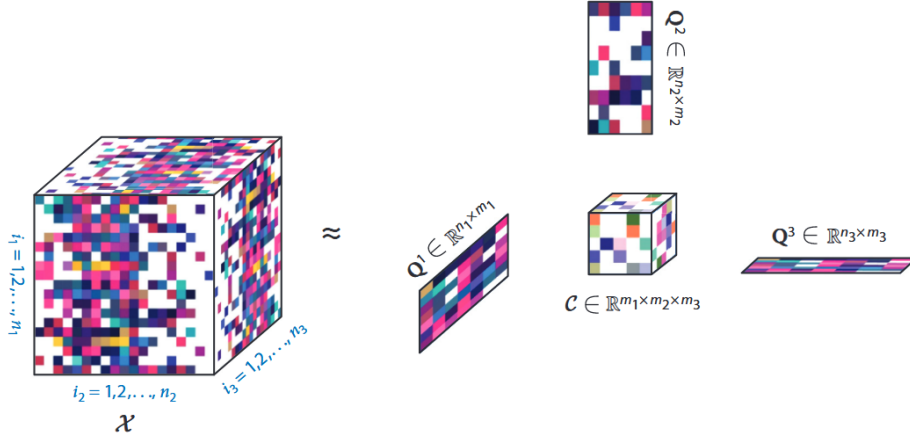


图 2.2 Tucker 分解

分解。它将张量近似为一组秩为 1 的张量的和。具体而言, 给定一个张量 $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$, 其 CP 分解形式为:

$$\mathcal{X} \approx \sum_{r=1}^R \beta_1^{(r)} \circ \beta_2^{(r)} \circ \cdots \circ \beta_d^{(r)} \quad (3)$$

其中 β_1, \dots, β_d 是长度为 p_1, \dots, p_d 的向量。PARAFAC 分解将系数 $p_1 \times \dots \times p_d$ 降低至 $R(p_1 + p_2 + \dots + p_d)$, 提供了有效的降维。

图 2.3^a 提供了 PARAFAC 分解的简单图解。

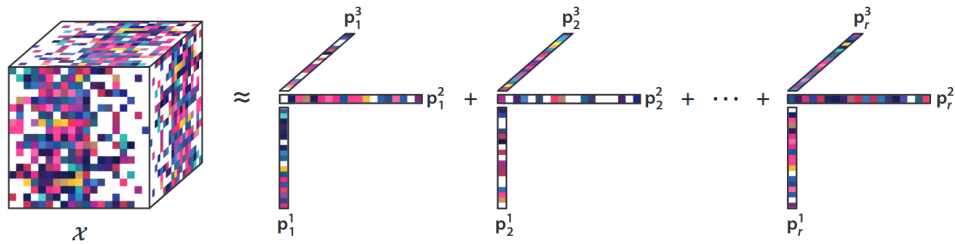


图 2.3 PARAFAC 分解

2.2 基于不同损失函数的分类模型

损失函数就像是我们用来衡量“错误”的工具。在机器学习中, 我们通过这个工具来评判模型训练性能。

贝叶斯方法中，损失函数转化为不同类型的似然（likelihood）。通过损失函数结合模型参数的先验假设和真实观测值来更新参数。通过这种方式，我们可以得到对模型参数的更好估计。

我们使用了两种常用的损失函数，分别是以支持向量机为代表的铰链损失（hinge loss）和逻辑回归损失（logistic regression loss），这两种损失函数都采用高维图像作为分类的协变量。

2.2.1 支持向量机 (SVM)

大多数基于 SVM 的分类方法采用带惩罚项的点估计克服高维协变量的影响，通常有以下形式：

$$\mathcal{L}(y|\boldsymbol{\beta}) = \frac{1}{\sigma^2} \max(1 - yf(\mathbf{x}; \boldsymbol{\beta}), 0) + R \quad (4)$$

这里的 $y \in \{-1, 1\}$ 是二元输出， $f(\cdot)$ 是协变量 \mathbf{x} 的线性或非线形函数， $\boldsymbol{\beta}$ 是需要从数据中估计的参数，以及 σ^2 是调整参数。图 2.4给出了可视化的铰链损失。SVM 并没有明确的

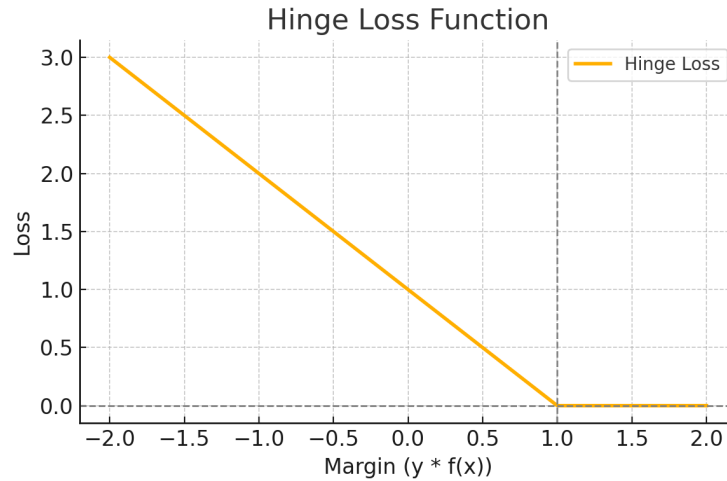


图 2.4 Hinge Loss

似然函数，不能直接在贝叶斯框架下建模。针对该问题，Polson 和 Scott 提出了一种伪似然（pseudo-likelihood）的方法^c。

具体而言，伪似然可以被表示为一种带有潜在变量 ρ 的位置-尺度混合正态分布（location-scale mixture of normals）：

$$\begin{aligned} L &= \prod_{i=1}^n L_i(y_i|\mathbf{x}_i, \boldsymbol{\beta}, \sigma^2) = \prod_{i=1}^n \left\{ \frac{1}{\sigma^2} \exp\left\{-\frac{2}{\sigma^2} \max(1 - y_i f(\mathbf{x}; \boldsymbol{\beta}), 0)\right\} \right\} \\ &= \int_0^\infty \prod_{i=1}^n \frac{1}{\sigma \sqrt{2\pi\rho_i}} \exp\left(-\frac{(1 + \rho_i - y_i f(\mathbf{x}; \boldsymbol{\beta}))^2}{2\rho_i \sigma^2}\right) d\rho_i \end{aligned} \quad (5)$$

这种表示法可以为后验推断提供高效的吉布斯采样器。

2.2.2 逻辑回归 (Logistic Regression)

逻辑损失函数是一种 S 型损失，有如下形式：

$$\mathcal{L}(y=1|\boldsymbol{\beta}) = \exp\{f(\mathbf{x};\boldsymbol{\beta})\}/(1+\exp\{f(\mathbf{x};\boldsymbol{\beta})\}) \quad (6)$$

其中， $f(\mathbf{x};\boldsymbol{\beta})$ 代表协变量对逻辑损失的贡献。 $\boldsymbol{\beta}$ 为待估参数. 图 2.5 给出了可视化的损失函数。

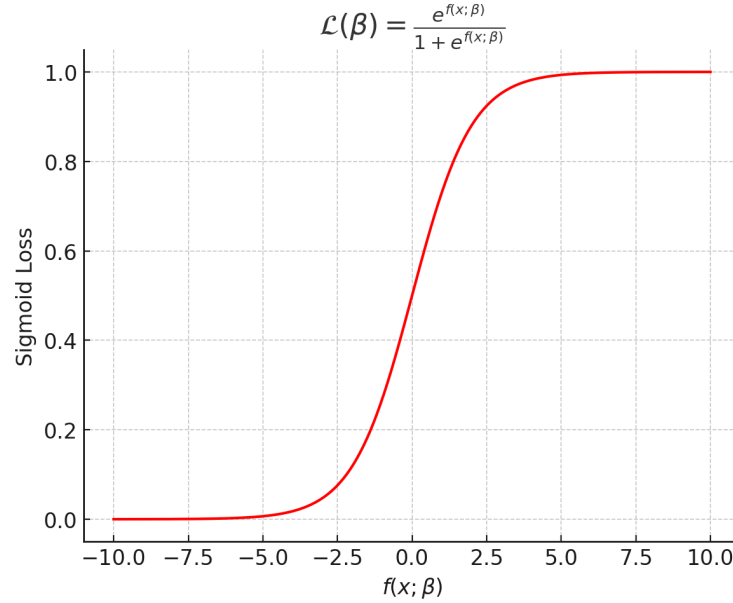


图 2.5 Logistic Loss

在贝叶斯框架中，逻辑回归的似然函数在分析上并不方便处理，这使得直接从后验分布中采样变得困难。针对此问题，我们通常使用 Polya-Gamma 潜变量来实现^d。

若随机变量 X 有如下形式

$$X \stackrel{D}{=} \frac{1}{2\pi^2} \sum_{k=1}^{\infty} \frac{g_k}{(k-1/2)^2 + c^2/(4\pi^2)} \quad (7)$$

则称 X 服从参数为 $b > 0, c \in \mathbb{R}$ 的 Polya-Gamma 分布，记作 $X \sim PG(b, c)$. 其中 g_k 服从 Gamma 分布 $Ga(b, 1)$ ，相互独立。 $\stackrel{D}{=}$ 表示在分布意义下相等。

通过引入 Polya-Gamma 潜在变量可以将对数优势比参数化的二项式似然表示为关于 Polya-Gamma 分布的高斯混合^d。

逻辑损失函数可以通过对潜在的 Polya-Gamma 变量进行边缘化处理而得到，其关系如下所示：

$$\frac{(e^{f(\cdot)})^y}{(1+e^{f(\cdot)})^b} = 2^{-b} e^{\kappa\psi} \int_0^{\infty} e^{-\omega\psi^2/2} p(\omega) d\omega, \quad b > 0, \quad \kappa = y - \frac{b}{2} \quad (8)$$

其中 $\omega \sim PG(b, 0)$ ， $p(\omega)$ 表示 Polya-Gamma 分布的密度函数。

在该等式基础上，完整的数据增强似然函数为

$$L = \prod_{i=1}^n \frac{(e^{f_i})^{y_i}}{1 + e^{f_i}} = \prod_{i=1}^n 2^{-1} e^{\kappa_i \psi_i} \int_0^\infty e^{-\omega_i f_i^2/2} p(\omega_i) d\omega_i \quad (9)$$

其中 $\kappa_i = y_i - \frac{1}{2}$, $b = 1$, $\omega_i \sim \text{PG}(1, 0)$.

2.3 先验估计

我们采用广泛应用的线性预测模型

$$f_i = \langle \mathbf{X}_i, \mathbf{B} \rangle + \mathbf{z}_i' \boldsymbol{\gamma} \quad (10)$$

这里的 \mathbf{X}_i 和 \mathbf{z}_i 分别表示第 i 个样本的影像预测变量与其他特征，例如人口统计学特征。符号 $\langle \cdot, \cdot \rangle$ 表示内积算子。

张量系数矩阵 \mathbf{B} 用于量化图像在分类模型中的作用， $\boldsymbol{\gamma}$ 是一个维度为 $p_z + 1$ 的向量，用以捕捉补充协变量的影响。

对张量 \mathbf{B} 进行 PARAFAC 分解，这里的 $\mathbf{B} \in \bigotimes_{j=1}^d \mathbb{R}^{p_j}$ ，有如下分解形式

$$\mathbf{B} \approx \sum_{r=1}^R \boldsymbol{\beta}_1^{(r)} \circ \boldsymbol{\beta}_2^{(r)} \circ \dots \circ \boldsymbol{\beta}_d^{(r)} \quad (11)$$

在贝叶斯框架下需要有先验假设，对这里 $\boldsymbol{\beta}_j^{(r)}$ 的先验选择，我们可以采用多向 Dirichlet 广义双帕累托（multiway Dirichlet generalized double Pareto，M-DGDP）分布^e。

Guhaniyogi 等人证明了^e 使用该方法作为先验可以在贝叶斯张量回归中实现自动稀疏性控制、低秩建模与不失大信号的精确建模，同时具备对称性和后验一致性的理论保证。

具体地，该先验在各组分之间以可交换方式诱导收缩效应，其中全局尺度参数为 $\tau \sim \text{Ga}(a_\tau, b_\tau)$ ，并在每个组分中进行调整为 $\tau_r = \phi_r \tau$ ， $r = 1, \dots, R$ ，其中

$$\boldsymbol{\Phi} = (\phi_1, \dots, \phi_R) \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_R) \quad (12)$$

其作用是鼓励在假设的 PARAFAC 分解中向低秩方向收缩。

此外，令 $\mathbf{W}_{jr} = \text{diag}(w_{jr,1}, \dots, w_{jr,p_j})$ ， $j = 1, \dots, d$ 且 $r = 1, \dots, R$ 表示边缘特异的尺度参数。

层次结构的边缘层先验给定如下：

$$\boldsymbol{\beta}_j^{(r)} \sim \mathcal{N}(0, (\phi_r \tau) \mathbf{W}_{jr}), \quad w_{jr,k} \sim \text{Exp}(\lambda_{jr}^2/2), \quad \lambda_{jr} \sim \text{Ga}(a_\lambda, b_\lambda). \quad (13)$$

对元素特异尺度进行边缘化后，可得：

$$\boldsymbol{\beta}_{j,k}^{(r)} \mid \lambda_{jr}, \phi_r, \tau \stackrel{\text{iid}}{\sim} \text{DE}(\lambda_{jr} / \sqrt{\phi_r \tau}), \quad 1 \leq k \leq p_j. \quad (14)$$

式 (13) 在单个边缘系数上诱导了 GDP（广义双帕累托）先验，从而具有自适应 Lasso 惩罚的形式。

在估计集合 $\mathbf{B}_r = \{\boldsymbol{\beta}_j^{(r)}; 1 \leq j \leq D\}$ 时，该模型通过引入边缘内异质性建模方式进行适应性调整，即引入元素特异的尺度参数 $w_{jr,k}$ 。其中共享的速率参数 λ_{jr} 可在边缘内的多个元素间传递信息，从而在局部尺度上诱导收缩。

最后我们假设 γ 的先验是 $\mathcal{N}(0, \Sigma_{0\gamma})$ ，完成了所有参数的先验估计。

2.4 后验推断的 MCMC 算法

2.4.1 MCMC 算法简介^g

蒙特卡罗法（Monte Carlo method）是通过从概率模型的随机抽样进行近似数值计算的方法。马尔可夫链蒙特卡罗法（Markov Chain Monte Carlo, MCMC），则是以马尔可夫链（Markov chain）为概率模型的蒙特卡罗法。马尔可夫链蒙特卡罗法构建一个马尔可夫链，使其平稳分布就是要进行抽样的分布，首先基于该马尔可夫链进行随机游走，产生样本的序列，之后使用该平稳分布的样本进行近似数值计算。

假设多元随机变量 x ，满足 $x \in \mathcal{X}$ ，其概率密度函数为 $p(x)$ ， $f(x)$ 为定义在 $x \in \mathcal{X}$ 上的函数，目标是获得概率分布 $p(x)$ 的样本集合，以及求函数 $f(x)$ 的数学期望 $\mathbb{E}_{p(x)}[f(x)]$ 。

应用马尔可夫链蒙特卡罗法解决这个问题。基本想法是：在随机变量 x 的状态空间 \mathcal{S} 上定义一个满足遍历定理的马尔可夫链 $X = \{X_0, X_1, \dots, X_t, \dots\}$ ，使其平稳分布就是抽样的目标分布 $p(x)$ 。然后在这个马尔可夫链上进行随机游走，每个时刻得到一个样本。根据遍历定理，当时间趋于无穷时，样本的分布趋近于平稳分布，样本的函数均值逼近函数的数学期望。所以，当时间足够长时（时刻大于某个正整数 m ），在之后的时间（时刻小于等于某个正整数 n , $n > m$ ）里随机游走得到的样本集合 $\{x_{m+1}, x_{m+2}, \dots, x_n\}$ 就是目标概率分布的抽样结果，得到的函数均值（遍历均值）就是要计算的数学期望值：

$$\hat{f} = \frac{1}{n-m} \sum_{i=m+1}^n f(x_i) \quad (15)$$

到时刻 m 为止的时间段称为燃烧期。

MCMC 在贝叶斯学习中起重要的作用。假设观测数据由随机变量 $y \in \mathcal{Y}$ 表示，模型由随机变量 $x \in \mathcal{X}$ 表示，贝叶斯学习通过贝叶斯定理计算给定数据条件下模型的后验概率，并选择后验概率最大的模型。

后验概率有如下计算公式：

$$p(x|y) = \frac{p(x)p(y|x)}{\int_{\mathcal{X}} p(y|x')p(x')dx'} \quad (16)$$

贝叶斯学习中经常需要进行三种积分运算：归一化（normalization）、边缘化（marginal-

ization)、数学期望 (expectation)。

后验概率计算中需要归一化计算：

$$\int_{\mathcal{X}} p(y|x')p(x')dx' \quad (17)$$

如果有隐变量 $z \in \mathcal{Z}$ ，后验概率的计算需要边缘化计算：

$$p(x|y) = \int_{\mathcal{Z}} p(x, z|y)dz \quad (18)$$

如果有一个函数 $f(x)$ ，可以计算该函数的关于后验概率分布的数学期望：

$$\mathbb{E}_{P(x|y)}[f(x)] = \int_{\mathcal{X}} f(x)p(x|y)dx \quad (19)$$

当观测数据和模型都很复杂的时候，以上的积分计算变得困难。马尔可夫链蒙特卡罗法为这些计算提供了一个通用的有效解决方案。

2.4.2 吉布斯采样 (Gibbs Sampling) 算法简介⁸

常用的 MCMC 算法有 Metropolis-Hastings 算法、吉布斯采样法。

算法 1 Gibbs Sampling (吉布斯抽样)

输入：目标分布的密度函数 $p(x)$ ，函数 $f(x)$ ；收敛步数 m ，迭代步数 n

输出：随机样本 $\{x_{m+1}, \dots, x_n\}$ 及函数样本均值 f_{mn}

- 1: 初始化：设初始样本 $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_k^{(0)})^\top$
- 2: **for** $i = 1$ to n **do**
- 3: 设上一步样本为 $x^{(i-1)} = (x_1^{(i-1)}, x_2^{(i-1)}, \dots, x_k^{(i-1)})^\top$
- 4: **for** $j = 1$ to k **do**
- 5: 从条件分布 $p(x_j|x_1^{(i)}, \dots, x_{j-1}^{(i)}, x_{j+1}^{(i-1)}, \dots, x_k^{(i-1)})$ 中采样，得到 $x_j^{(i)}$
- 6: **end for**
- 7: 得到当前样本 $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)})^\top$
- 8: **end for**
- 9: 抽取后 $n - m$ 个样本组成样本集合 $\{x^{(m+1)}, \dots, x^{(n)}\}$
- 10: 计算函数样本均值：

$$f_{mn} = \frac{1}{n - m} \sum_{i=m+1}^n f(x^{(i)})$$

吉布斯抽样 (Gibbs sampling) 用于多元变量联合分布的抽样和估计。其基本做法是，从联合概率分布定义满足条件概率分布，依次对满足条件概率分布进行抽样，得到样本的序列。可以证明这样的抽样过程是在一个马尔可夫链上的随机游走，每一个样本对应着马尔

可夫链的状态，平稳分布就是目标的联合分布。整体成为一个马尔可夫链蒙特卡罗法，燃烧期之后的样本就是联合分布的随机样本。算法 1 中列出了具体的过程。

2.4.3 基于 SVM 损失的贝叶斯张量模型

令 $y \in \mathbb{R}$ 表示一个响应值, $\mathbf{z} \in \mathbb{R}^p$, $\mathbf{X} \in \bigotimes_{j=1}^d \mathbb{R}^{p_j}$, 我们有如下的张量回归模型:

$$\begin{aligned} y|\boldsymbol{\gamma}, \mathbf{B}, \sigma &\sim \mathcal{N}(\mathbf{z}'\boldsymbol{\gamma} + \langle \mathbf{X}, \mathbf{B} \rangle, \sigma^2) \\ \mathbf{B} &= \sum_{r=1}^R \mathbf{B}_r, \quad \mathbf{B}_r = \beta_1^{(r)} \circ \dots \circ \beta_d^{(r)} \\ \boldsymbol{\gamma} &\sim \pi_{\boldsymbol{\gamma}}, \quad \beta_j^{(r)} \sim \pi_{\beta} \end{aligned} \tag{20}$$

先前提出的多向先验(13) 可为张量回归模型 (20) 的大多数参数提供吉布斯采样方案。我们依赖于边缘化与分块策略来减少以下参数的自相关性:

$$\left\{ \left(\beta_j^{(r)}, w_{jr}; 1 \leq j \leq d, 1 \leq r \leq R \right), (\boldsymbol{\Phi}, \tau), \boldsymbol{\gamma} \right\}$$

从 $[\alpha, \boldsymbol{\Phi}, \tau | \mathbf{B}, \mathbf{W}]$, $[\mathbf{B}, \mathbf{W} | \boldsymbol{\Phi}, \tau, \boldsymbol{\gamma}, \mathbf{y}]$, 以及 $[\boldsymbol{\gamma} | \mathbf{B}, \mathbf{y}]$ 依次进行采样, 具体见算法 2。

算法 2 BT-SVM 算法

- 1: 从逆高斯分布更新 ρ_i : $\rho_i^{-1} \sim IN(\mu_i, \lambda_i)$, 其中 $\mu_i = |1 - y_i(< X_i, B > + z'_i \gamma)|^{-1}$ 且 $\lambda_i = 1/\sigma^2$.
- 2: 对 $[\alpha, \Phi, \tau | B, W]$ 组合采样为 $[\alpha | B, W][\Phi, \tau | \alpha, B, W]$:
 - (a) 使用 griddy-Gibbs 采样 $[\alpha | B, W]$: 对每个 $\alpha \in \mathcal{A}$, 通过从 $[\Phi, \tau | \alpha, B, W]$ 采样 M 次, 构建参考集. 令 $w_{j,l} = \pi(B | \alpha, \Phi_l, \tau_l, W) \pi(\Phi_l, \tau_l | \alpha)$, 其中 $1 \leq l \leq M$, $p(\alpha | B, W) = \pi(\alpha) \sum_{l=1}^M w_{j,l} / M$, 以及

$$\Pr(\alpha = \alpha_j | -) = \frac{p(\alpha_j | B, W)}{\sum_{\alpha \in \mathcal{A}} p(\alpha | B, W)}$$

- (b) 对特定组分采样尺度参数 $[\Phi, \tau | \alpha^*, B, W] = [\Phi | B, W][\tau | \Phi, B, W]$; 设 $p_0 = \sum_j p_j$, $a_\tau = R\alpha$, $b_\tau = \alpha(R/\nu)^{1/d}$.
 - (1) 对每个 $r = 1, \dots, R$, 采样: $\psi_r \sim \text{giG}(\alpha - p_0/2, 2b_\tau, 2C_r)$, 其中 $C_r = \sum_{j=1}^d \beta_j^{(r)\top} W_{jr}^{-1} \beta_j^{(r)}$, $\phi_r = \psi_r / \sum_{l=1}^R \psi_l$.
 - (2) $\tau \sim \text{giG}(a_\tau - Rp_0/2, 2b_\tau, 2\sum_{r=1}^R D_r)$, 其中 $D_r = C_r / \phi_r$.
- 3: 使用回溯拟合程序采样 $\{\beta_j^{(r)}, \omega_{jr}, \lambda_{jr}\}$, 以生成跨分量的边际条件分布的序列抽样.
 - (a) 抽取 $[w_{jr}, \lambda_{jr} | \beta_j^{(r)}, \phi_r, \tau] = [w_{jr} | \lambda_{jr}, \beta_j^{(r)}, \phi_r, \tau][\lambda_{jr} | \beta_j^{(r)}, \phi_r, \tau]$.
 - (1) 抽取 $\lambda_{jr} \sim \text{Ga}(a_\lambda + p_j, b_\lambda + \|\beta_j^{(r)}\|_1 / \sqrt{\phi_r \tau})$;
 - (2) 对于 $1 \leq k \leq p_j$, 独立抽取 $w_{jr,k} \sim \text{giG}\left(\frac{1}{2}, \frac{\lambda_{jr}^2}{2}, \beta_{j,k}^2 / (\phi_r \tau)\right)$.
 - (b) 从多元正态分布: $\beta_j^{(r)} \sim N(\mu_{jr}, \Sigma_{jr})$ 抽取, 其中 $\mu_{jr} = \frac{\Sigma_{jr} H_j^{(r)\top} \tilde{y}}{\sigma^2}$, $\Sigma_{jr} = \left(\frac{H_j^{(r)\top} H_j^{(r)}}{\sigma^2} + \frac{W_{jr}^{-1}}{\phi_r \tau}\right)^{-1}$. 以及

$$h_{i,j,k}^{(r)} = \sum_{d_1=1, \dots, d_d=1}^{p_1, \dots, p_d} I(d_j = k) x_{d_1, \dots, d_d} \left(\prod_{l \neq j} \beta_{l,i_l}^{(r)} \right),$$

$$\mathbf{H}_{i,j}^{(r)} = (h_{1,j,1}^{(r)} / \sqrt{\rho_1}, \dots, h_{i,j,p_j}^{(r)} / \sqrt{\rho_i})',$$

$$\tilde{y}_i = \frac{y_i}{\sqrt{\rho_i}} (\rho_i + 1 - y_i(z'_i \gamma + \sum_{l \neq r} < X_i, B_l >))$$

- 4: 从共轭正态条件分布 $\gamma \sim N(\mu_\gamma, \Sigma_\gamma)$ 更新 γ , 其中 $\mu_\gamma = \Sigma_\gamma Z^T (\tilde{y} y \rho)$, $\Sigma_\gamma = (G^T G / \sigma^2 + \Sigma_{0\gamma})^{-1}$, 以及 $\mathbf{G}_{i,p_z} = \mathbf{Z}_{i,p_z} / \sqrt{\rho_i}$ 和 $\tilde{y}_i = \rho_i + 1 - y_i < X_i, B >$.

2.4.4 基于 Logistic 损失的贝叶斯张量模型

算法 3 BT-LR 算法

- 1: 从 Pólya-gamma 分布更新 ω_i : $\omega_i \sim \text{PG}(1, \langle X_i, \mathbf{B} \rangle + z_i^T \gamma)$.
 - 2: 对 $[\alpha, \Phi, \tau | \mathbf{B}, \mathbf{W}]$ 组合采样为 $[\alpha | \mathbf{B}, \mathbf{W}][\Phi, \tau | \alpha, \mathbf{B}, \mathbf{W}]$:
 - (a) 使用 griddy-Gibbs 采样 $[\alpha | \mathbf{B}, \mathbf{W}]$: 对每个 $\alpha \in \mathcal{A}$, 通过从 $[\Phi, \tau | \alpha, \mathbf{B}, \mathbf{W}]$ 采样 M 次, 构建参考集. 令 $w_{j,l} = \pi(\mathbf{B} | \alpha, \Phi_l, \tau_l, \mathbf{W})\pi(\Phi_l, \tau_l | \alpha)$, 其中 $1 \leq l \leq M$, $p(\alpha | \mathbf{B}, \mathbf{W}) = \pi(\alpha) \sum_{l=1}^M w_{j,l} / M$, 以及

$$\Pr(\alpha = \alpha_j | -) = \frac{p(\alpha_j | \mathbf{B}, \mathbf{W})}{\sum_{\alpha \in \mathcal{A}} p(\alpha | \mathbf{B}, \mathbf{W})}$$
 - (b) 对特定组分采样尺度参数 $[\Phi, \tau | \alpha^*, \mathbf{B}, \mathbf{W}] = [\Phi | \mathbf{B}, \mathbf{W}][\tau | \Phi, \mathbf{B}, \mathbf{W}]$; 设 $p_0 = \sum_j^d p_j$, $a_\tau = R\alpha$, $b_\tau = \alpha(R/v)^{1/d}$.
 - (1) 对每个 $r = 1, \dots, R$, 采样: $\psi_r \sim \text{giG}(\alpha - p_0/2, 2b_\tau, 2C_r)$, 其中 $C_r = \sum_{j=1}^d \beta_j^{(r)\top} \mathbf{W}_{jr}^{-1} \beta_j^{(r)}$, $\phi_r = \psi_r / \sum_{l=1}^R \psi_l$.
 - (2) $\tau \sim \text{giG}(a_\tau - Rp_0/2, 2b_\tau, 2\sum_{r=1}^R D_r)$, 其中 $D_r = C_r / \phi_r$.
 - 3: 使用回溯拟合程序采样 $\{\beta_j^{(r)}, \omega_{jr}, \lambda_{jr}\}$, 以生成跨分量的边际条件分布的序列抽样.
 - (a) 抽取 $[w_{jr}, \lambda_{jr} | \beta_j^{(r)}, \phi_r, \tau] = [w_{jr} | \lambda_{jr}, \beta_j^{(r)}, \phi_r, \tau][\lambda_{jr} | \beta_j^{(r)}, \phi_r, \tau]$.
 - (1) 抽取 $\lambda_{jr} \sim \text{Ga}(a_\lambda + p_j, b_\lambda + \|\beta_j^{(r)}\|_1 / \sqrt{\phi_r \tau})$;
 - (2) 对于 $1 \leq k \leq p_j$, 独立抽取 $w_{jr,k} \sim \text{giG}(\frac{1}{2}, \frac{\lambda_{jr}^2}{2}, \beta_{j,k}^2 / (\phi_r \tau))$.
 - (b) 从多元正态分布中抽取 $\beta_j^{(r)}$: $\beta_j^{(r)} \sim N(\mu_{jr}, \Sigma_{jr})$, 其中 $\mu_{jr} = \Sigma_{jr}(\Omega(H_j^{(r)})^T \tilde{y})$ 且 $\Sigma_{jr} = ((H_j^{(r)})^T \Omega H_j^{(r)} + \mathbf{W}_{jr}^{-1} / (\phi_r \tau))^{-1}$, 其中 $\tilde{y} = \kappa / \omega$, $\kappa = (y_1 - N_1/2, \dots, y_n - N_n/2)$, $N_1 = \dots = N_n = 1$, 且 Ω 是一个对角矩阵, 其对角线元素为 ω_i 's.
 - 4: 从共轭正态条件分布 $\gamma \sim N(\mu_\gamma, \Sigma_\gamma)$ 更新 γ , 其中 $\mu_\gamma = \Sigma_\gamma Z^T (\tilde{y} \omega)$, $\Sigma_\gamma = (G^T G + \Sigma_{0\gamma}^{-1})^{-1}$, 以及 $G_{i,p_z} = Z_{i,p_z} * \sqrt{\omega_i}$ 且 $\tilde{y}_i = \kappa_i / \omega_i - \langle X_i, \mathbf{B} \rangle$.
-

2.4.5 参数设置

通过选择先验分布中合适的超参数值, 可以获得良好的整体性能. 基于 Guhaniyogi 等人的研究^e, 我们将全局尺度 τ 的超先验参数设为 $a_\tau = 1$ 和 $b_\tau = \alpha R^{(1/d)}$, 其中 R 是假设的 PARAFAC 分解中的秩, 并设置 $\alpha_1 = \dots = \alpha_R = 1/R$. 对于共同速率参数 λ_{jr} , 我们设置 $a_\lambda = 3$ 和 $b_\lambda = \sqrt[4]{a_\lambda}$.

在 SVM 损失下, 缩放参数 σ^2 是一个固定参数, 可以手动调整以获得最大的模型性能. Lyu 等人^f 测试了从 0.1 到 10 的几个调整参数 σ^2 值, 并选择 $\sigma^2 = 6$. 为了确定拟合模型的秩, 使用秩 2–5 拟合了所提出的模型, 并选择了使偏差信息准则 (DIC) 分数最小的秩. DIC

衡量一组贝叶斯分层模型的拟合优度，通过对模型复杂性进行调整，以惩罚更复杂的模型。

第三章 模拟数据研究

3.1 数据生成

我们通过几种模拟设置来阐述方法的性能，并使用其他已有的方法进行比较，这些方法基于各种类型的生成数据集，包括几种类型的函数信号以及由 SVM 和逻辑损失函数生成的数据。我们考虑了四种不同类型的张量系数 \mathbf{B} 来生成二元结果，设置如下：

场景 1 在此设置中，张量 \mathbf{B} 由秩 $R_0 = 3$ 和维度 $p = c(48, 48)$ 的秩- R PARAFAC 分解构建。每个 $\beta_j^{(r)}$ 都是从独立的二项分布 $Binomial(2, 0.2)$ 生成的。在构建张量之后，我们将张量 \mathbf{B} 单元格的最大值设置为 1。图 3.1 展示了该场景的张量图像。

```

1 p <- c(48, 48)
2 R <- 3
3
4 # 构造 rank-R PARAFAC 张量
5 generate_parafac_tensor <- function(p, R, prob = 0.2, size = 2) {
6   A_list <- list()
7   B_list <- list()
8
9   for (r in 1:R) {
10     a_r <- rbinom(p[1], size = size, prob = prob)
11     b_r <- rbinom(p[2], size = size, prob = prob)
12     A_list[[r]] <- a_r
13     B_list[[r]] <- b_r
14   }
15
16   tensor <- matrix(0, nrow = p[1], ncol = p[2])
17   for (r in 1:R) {
18     tensor <- tensor + outer(A_list[[r]], B_list[[r]
19                               ])
20   }
21
22   # 标准化为最大值为 1
23   tensor <- tensor / max(tensor)
24   return(tensor)
25 }
```

```

25
26 # 生成张量
27 set.seed(123)
28 Beta_tens <- generate_parafac_tensor(p = c(48, 48), R = 3)

```

场景 2 张量图像由秩 $R_0 = 3$ 的秩- R PARAFAC 分解模拟。这里，我们没有从已知分布生成张量边缘，而是手动设置了 $\beta_j^{(r)}$ 的每个值。图 3.2 展示了该场景的张量图像。

```

1 # 手动设置边缘向量
2 a1 <- rep(0, 48)
3 a1[38:48] <- 1
4 b1 <- rep(0, 48)
5 b1[5:12] <- 1
6
7 a2 <- rep(0, 48)
8 a2[38:48] <- 1
9 b2 <- rep(0, 48)
10 b2[33:43] <- 1
11
12 a3 <- rep(0, 48)
13 a3[5:12] <- 1
14 b3 <- rep(0, 48)
15 b3[c(5:12, 33:43)] <- 1
16
17 # 构造 PARAFAC 张量
18 Beta_tens <- outer(a1, b1) + outer(a2, b2) + outer(a3, b3)

```

场景 3 与从 PARAFAC 分解生成 2D 张量图像不同，张量系数 \mathbf{B} 对于矩形区域设置为 1，否则为 0。非零元素约占总面积的 30%。图 3.3 展示了该场景的张量图像。

```

1 Beta_tens <- matrix(0, 48, 48)
2 for (i in 15:40) {
3   for (j in 10:35) {
4     Beta_tens[i, j] <- 1
5   }
6 }

```

场景 4 张量系数 \mathbf{B} 对于圆形区域设置为 1，否则为 0。非零元素约占总面积的 10%。图 3.4 展示了该场景的张量图像。

```

1 Beta_tens <- matrix(0, 48, 48)
2 # 圆的中心和半径
3 center_x <- 18
4 center_y <- 18
5 radius <- sqrt(0.10 * 48 * 48 / pi)
6
7 # 填充圆形区域为 1
8 for (i in 1:48) {
9     for (j in 1:48) {
10         # 计算 (i, j) 到圆心的距离
11         if ((i - center_x)^2 + (j - center_y)^2 <= radius
12             ^2) {
13             Beta_tens[i, j] <- 1
14         }
15     }
16 }

```

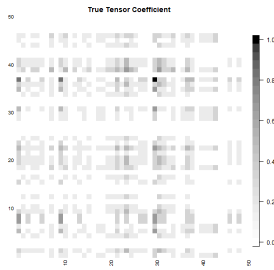


图 3.1 Scene 1

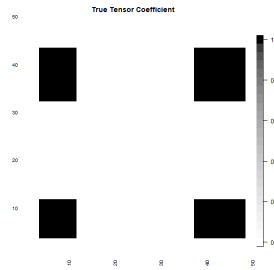


图 3.2 Scene 2

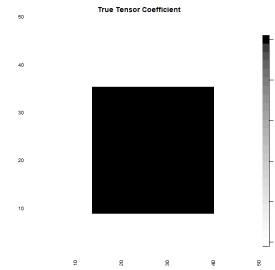


图 3.3 Scene 3

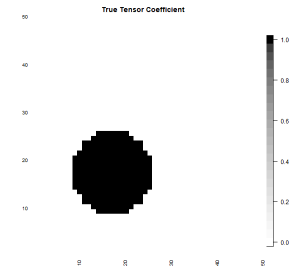


图 3.4 Scene 4

3.2 模型评估

我们使用均方根误差（Root Mean Squared Error, RMSE）与相关系数（correlation coefficient）来评估单元层级张量系数的点估计精度。同时，为了衡量分类准确性，我们计算误分类率（misclassification error）与 F1 分数（F1 score）。具体地，这些指标定义如下：

令 θ_j , $j = 1, \dots, J$ 表示向量化的张量系数，其中 $J = \prod_{k=1}^d p_k$ 为张量系数 \mathbf{B} 中的总单元数。此外，以下术语用于描述 SVM 分类器下的分类性能：

- (1) 真阳性（TP）：分类器正确预测为阳性类别的样本数；
- (2) 假阳性（FP）：分类器将负类样本错误预测为阳性类别的样本数；
- (3) 真阴性（TN）：分类器正确预测为负类类别的样本数；

(4) 假阴性 (FN): 分类器将阳性样本错误预测为负类的样本数。

在逻辑回归模型中, 以上定义相同, 只是将负类替换为零类。此外, TP/FP/TN/FN 的定义亦可用于特征选择性能的评估, 其中正类对应非零系数, 而负类/零类则对应缺失或为零的系数。

系数估计性能指标 包括以下两种:

(1) 均方根误差 (Root-mean-square error), 定义为:

$$\text{RMSE}(\theta) = \sqrt{\frac{1}{J} \sum_{j=1}^J (\hat{\theta}_j - \theta_j)^2} \quad (21)$$

用于衡量估计值与真实值之间的误差。

(2) 估计系数与真实系数之间的相关系数。

```
1 tensor <- getBeta_mcmc(sim$beta.store)
2 tensor_est <- apply(tensor[burnin:nsweep, ], 2, mean) * sim$sy /
  sim$sx
3 rmse_val <- rmse(c(Beta_tens), c(tensor_est))
4 cor_val <- cor(c(Beta_tens), c(tensor_est))
```

分类性能指标 包括:

(i) 误分类率 (misclassification rate), 定义为:

$$\frac{FP + FN}{TP + TN + FP + FN} \quad (22)$$

(ii) F1 分数, 定义为精确率 (Precision) 与召回率 (Recall) 的调和平均, 其中:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \quad (23)$$

F1 分数表达式为:

$$\text{F1} = \frac{2TP}{2TP + FP + FN}. \quad (24)$$

```
1 TP <- sum(clust.test == 1 & y.test == 1)
2 FP <- sum(clust.test == 1 & y.test == -1)
3 FN <- sum(clust.test == -1 & y.test == 1)N
4 f1score <- TP / (TP + (FP + FN) / 2)
```

实验中, 我们将数据按 70:30 的比例划分为训练集与测试集。用于系数估计与特征选择性能评估的指标在训练集上计算, 而分类性能指标在测试集上进行评估。

```

1 train_index <- sort(sample(1:N, 0.7 * N, replace = FALSE))
2 x.train <- X[train_index, , ]
3 y.train <- Ylabel[train_index]
4 x.test <- X[-train_index, , ]
5 y.test <- Ylabel[-train_index]

```

为检验新模型的性能，我们选取两种现有的先进分类方法作为对比模型：

带有 Lasso 惩罚项的逻辑回归模型 (Lasso Logistic Regression) 该模型通过传统逻辑回归的损失函数中引入 L1 正则项以实现特征选择，其优化目标如下：

$$\hat{\beta} = \arg \min_{\beta} \left\{ -\frac{1}{n} \sum_{i=1}^n [y_i \log p_i + (1 - y_i) \log (1 - p_i)] + \lambda \|\beta\|_1 \right\} \quad (25)$$

其中 $p_i = \frac{1}{1 + e^{-x_i^T \beta}}$ ， λ 为正则化参数， $\|\beta\|_1 = \sum_j |\beta_j|$ 表示 L1 范数。该模型由 R 语言中的 glmnet 包实现，适用于变量维度远高于样本数量的高维稀疏问题。

```

1 library(glmnet)
2 cvfit <- cv.glmnet(x.train.mat, y.train, family = "binomial",
  alpha = 1)

```

L1 范数支持向量机模型 (L1-norm Support Vector Machine, SVM) 该模型在传统支持向量机的基础上引入 L1 正则项，以增强特征选择能力，其优化目标函数如下：

$$\hat{\beta} = \arg \min_{\beta} \left\{ \frac{1}{2} \|\beta\|_1 + C \sum_{i=1}^n \max(0, 1 - y_i \cdot x_i^T \beta) \right\} \quad (26)$$

其中 $y_i \in \{-1, 1\}$ ， C 为调节间隔与误分类权衡的正则参数，第一项为 L1 范数惩罚项，第二项为 Hinge 损失函数。该模型由 R 语言中的 Liblinear 包实现，适用于高维低样本的稀疏分类任务。

```

1 library(Liblinear)
2 model <- Liblinear(data = x.train.mat, target = y.train, type =
  5, cost = 1, bias = TRUE, verbose = FALSE)

```

这两种方法均基于张量协变量的向量化表示，将其转换为标量变量后输入统计模型中，因此无法保留张量图像的空间信息。

```

1 # 向量化
2 x.train.mat <- t(apply(x.train, 1, c))
3 x.test.mat <- t(apply(x.test, 1, c))

```

此外，我们还使用网格搜索算法与交叉验证来选择最佳调参值以进行模型拟合。

我们将 MCMC 链的迭代次数设置为 3000，其中包含 1000 次的燃烧期，计算时间因所选的不同因素而有所不同。

表 3.1 标签由 SVM 损失生成时不同模型在每种场景下的表现

Scenarios	Methods	RMSE	Corr.Coef.	Mis. Class.	F1-score
4*Scenario 1	LR w/ lasso	0.186	0.118	0.500	0.576
	L1norm-SVM	0.129	0.233	0.413	0.523
	BT-SVM	0.127	0.591	0.340	0.653
	BT-LR	0.197	0.449	0.360	0.640
4*Scenario 2	LR w/ Lasso	0.395	0.057	0.520	0.487
	L1norm-SVM	0.393	0.168	0.433	0.591
	BT-SVM	0.285	0.820	0.193	0.803
	BT-LR	0.330	0.505	0.347	0.653
4*Scenario 3	LR w/ Lasso	0.541	0.064	0.453	0.585
	L1norm-SVM	0.539	0.166	0.407	0.639
	BT-SVM	0.430	0.768	0.227	0.788
	BT-LR	0.454	0.570	0.320	0.684
4*Scenario 4	LR w/ Lasso	0.315	0.178	0.407	0.639
	L1norm-SVM	0.315	0.213	0.440	0.554
	BT-SVM	0.221	0.769	0.207	0.805
	BT-LR	0.276	0.529	0.347	0.679

表 3.1 和表 3.2 中展示了评估模型性能的相关结果，分别对应场景 1-4。具体而言，表 3.1 展示了在二元结果 Y 由 SVM Loss 生成时的四个场景的结果，而表 3.2 则展示了当 Y 来自逻辑损失时的结果。这些结果表明，提出的两种方法（BT-SVM 和 BT-LR）在系数估计和分类性能上始终优于其他竞争的惩罚方法。

当二元结果数据来自 SVM 损失时：

```

1 # SVM Loss
2 Ylabel <- rep(0, N)
3 Ylabel[Y >= 0] <- 1
4 Ylabel[Y < 0] <- -1

```

BT-SVM 方法具有优越的系数估计（如表 3.1 中较低的 RMSE 和较高的相关系数所示）和改进的分类精度（如表 3.1 中较低的误分类率和较高的 F1 分数所示）。即便在数据由逻辑损失生成时：


```

1 # Logistic loss
2 p <- 1 / (1 + exp(-Y))
3 Ylabel <- rbinom(n = N, size = 1, prob = p)

```

依然有这一特点。

表 3.2 标签由逻辑损失生成时不同模型在每种场景下的表现

Scenarios	Methods	RMSE	Corr.Coeff.	Mis. Class.	F1-score
4*Scenario 1	LR w/ lasso	0.175	0.078	0.493	0.580
	L1norm-SVM	0.192	0.208	0.527	0.448
	BT-SVM	0.120	0.701	0.220	0.793
	BT-LR	0.230	0.456	0.340	0.698
4*Scenario 2	LR w/ Lasso	0.395	0.128	0.547	0.474
	L1norm-SVM	0.393	0.215	0.453	0.534
	BT-SVM	0.233	0.880	0.140	0.844
	BT-LR	0.284	0.670	0.233	0.788
4*Scenario 3	LR w/ Lasso	0.542	0.050	0.427	0.467
	L1norm-SVM	0.539	0.149	0.413	0.544
	BT-SVM	0.426	0.811	0.187	0.823
	BT-LR	0.414	0.555	0.313	0.647
4*Scenario 4	LR w/ Lasso	0.316	0.127	0.447	0.518
	L1norm-SVM	0.315	0.193	0.473	0.530
	BT-SVM	0.221	0.739	0.213	0.790
	BT-LR	0.317	0.479	0.360	0.635

图 3.5 展示了使用 BT-SVM 和 BT-LR 估算张量系数的情况。从图中可以看出，我们提出的方法能够广泛地恢复二维张量 \mathbf{B} 的形状，而不受其形状的影响，也不取决于张量信号是否通过 PARAFAC 分解构建。

为了展示预测系数和真实系数之间的相关程度，我们通过图 3.6 展示了不同模型（LR 和 SVM）在不同情景下的表现。

```

1 residual_map <- Beta_tens - matrix(tensor_est, nrow = 48)
2 png("Residual_Tensor_Scenario4_SVM.png", width = 600, height =
  600)
3 image.plot(residual_map,

```

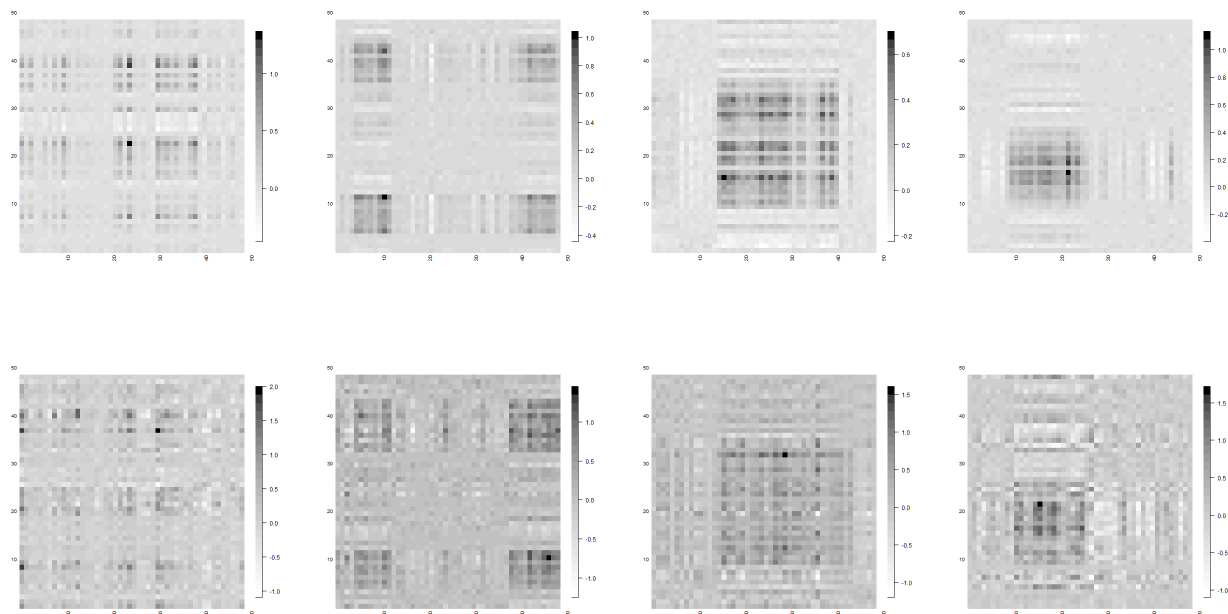


图 3.5 估计的张量系数（第一排由 BT-SVM 模型估计第二排由 BT-LR 模型估计）

```

4 col = topo.colors(25), axes = FALSE,
5 main = "Residual Map: Beta_tens - tensor_est"
6 )
7 dev.off()

```

这些热力图展示了每个情景下模型预测值与真实值之间的残差分布，便于分析每个模型的准确性和误差特征。

相比之下，带惩罚项的竞争方法，例如带 LASSO 惩罚的逻辑回归和 L1norm-SVM 表现不佳。从表 3.1 和表 3.2 中可以发现，真实系数和估计系数之间的相关系数往往接近于零，反映了它们无法检测到真实信号，从而导致系数估计性能不佳，分类效果大打折扣。

3.3 MRI 脑肿瘤分类

数据集包含 155 张脑部肿瘤切片 MRI 图像和 98 张正常 MRI 图像。我们对比了 BT-SVM，BT-LR，带 Lasso 惩罚的逻辑回归以及 L1norm-SVM 对该数据集的分类性能。

我们按照 70:30 划分训练集与测试集。由于该实例样本量较小，在处理 MCMC 链过程时，设置迭代次数为 500 次，其中燃烧期为 200 次。通过将图像数据转换成 48×48 的灰度图构建张量信号，测试结果见表 3.3。可以看出 BT-SVM 在该分类任务中表现最佳，而经典的带惩罚项的模型在这类分类任务中无法有效检测真实信号，导致分类效果较差。

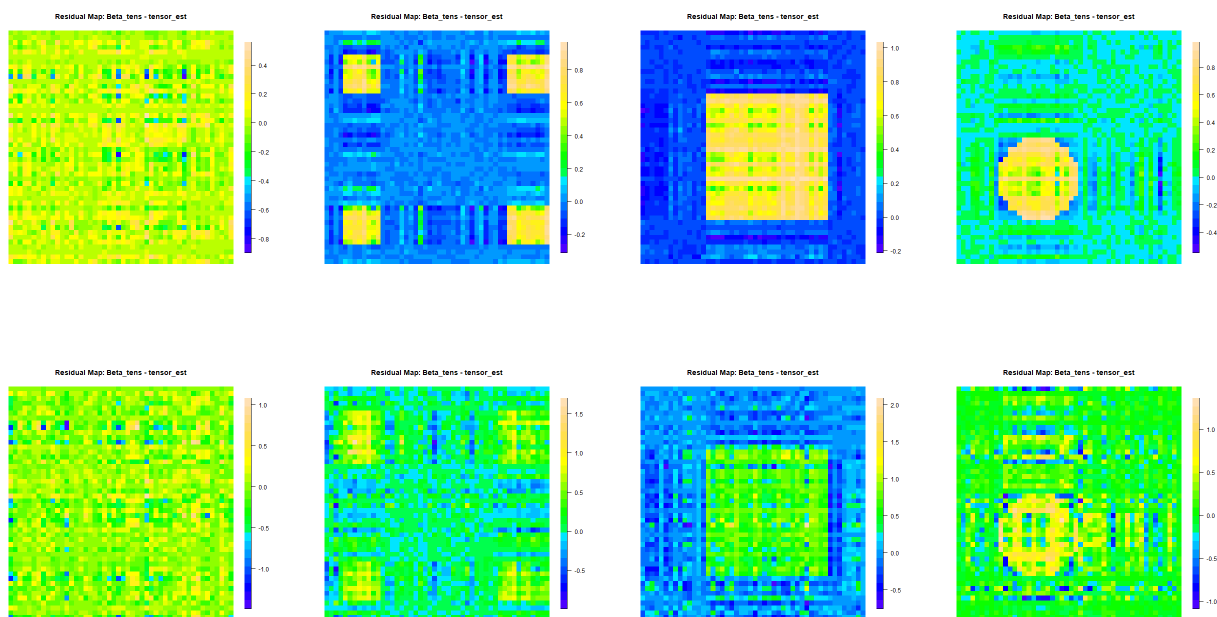


图 3.6 残差图

表 3.3 不同模型对脑肿瘤影像的分类性能

Methods	Mis. Class.	F1-score
LR w/ lasso	0.333	0.702
L1norm-SVM	0.451	0.709
BT-SVM	0.211	0.855
BT-LR	0.303	0.763

附 录

Listing A.1 BT-SVM 算法

```

1 library ( fields )
2 library ( abind )
3 library ( imager )
4 library ( magrittr )
5 SVMMDPL <- function ( Z, X, Ylabel , rank = 3, nsweep = 1e3, nskip = 3, a.lam ,
    b.lam , phi.alpha , scale = TRUE ) {
6     library ( base )
7     library ( statmod )
8     library ( MASS )
9     library ( matlib )
10    library ( stats )
11    library ( plyr )
12    library ( GIGrvg )
13    library ( gtools )
14    library ( coda )
15    library ( fields )
16    library ( dplyr )
17
18    ## data input
19    # N : 样本数量
20    # P: 张量图像大小, 例如 48*48
21    # D: 张量图像的维度数, 对于二维张量, D=2; 对于三维张量, D=3.
22    # Z : 一个包含人口统计协变量的 N 行 pgamma 列 矩阵 (例如, 年龄、性
    别等信息) pgamma是协变量的数量。
23    # X ̃: N * ̃P[1] * P[2] 图像数据张量, N 个 P[1] * P[2] 大小的图
    像。
24    # Ylabel : 包含二分类标签的相应变量。
25
26    # 初始化变量
27    Yi <- Ylabel
28    N <- length ( Yi )
29    P <- dim ( X ) [ -1 ]
30    D <- length ( dim ( X ) ) - 1
31    pgamma <- ncol ( Z )
32

```

```

33     # 标准化
34     if (is.null(Z)) {
35         mz <- NA
36         sz <- NA
37     } else {
38         mz <- colMeans(Z)
39         sz <- rep(1, pgamma)
40     }
41     if (!is.null(Z)) {
42         if (!is.matrix(Z)) {
43             Z <- as.matrix(Z)
44         }
45     }
46     Zt <- Z
47     obs <- as.numeric(Yi)
48     sy <- 1
49     my <- 0
50
51     if (scale) {
52         if (!is.null(Z)) {
53             mz <- colMeans(Z)
54             sz <- apply(Z, 2, function(z) diff(range(z)))
55             sz[sz == 0] <- 1
56             Zt <- Z
57             for (jj in 1:pgamma) {
58                 Zt[, jj] <- (Z[, jj] - mz[jj]) / sz[jj]
59             }
60         }
61         Xt <- 0 * X
62         mx <- apply(X, c(2:(D + 1)), function(z) mean(z, na.rm = T
63             ))
64         sx <- apply(X, c(2:(D + 1)), function(z) diff(range(z, na.
65             rm = T)))
66         sx[sx == 0] <- 1
67         if (D == 2) {
68             for (jj in 1:nrow(X)) Xt[jj, , ] <- (X[jj, , ] -
69                 mx) / sx
69         } else if (D == 3) {
70             for (jj in 1:nrow(X)) Xt[jj, , , ] <- (X[jj, , , ]
71                 - mx) / sx

```

```

69         }
70     } else {
71         if (!is.null(Z)) {
72             mz <- rep(0, pgamma)
73             sz <- rep(1, pgamma)
74             Zt <- Z
75         }
76         mx <- array(0, dim = dim(X)[-1])
77         sx <- array(1, dim = dim(X)[-1])
78         Xt <- X
79     }
80
81     # MCMC setup
82
83     x.train.nona <- Xt
84     x.train.nona[is.na(Xt)] <- 0
85
86     if (missing(a.lam)) a.lam <- rep(3, rank)
87     if (missing(b.lam)) b.lam <- (a.lam)**(1 / (2 * D))
88     if (missing(phi.alpha)) phi.alpha <- rep(1 / rank, rank)
89
90     phi.a0 <- sum(phi.alpha)
91     a.vphi <- phi.a0
92     b.vphi <- phi.alpha[1] * rank^(1 / D)
93
94     s0 <- 1
95     a.t <- 0.1
96     b.t <- 1 # b.t = 2.5/2 * s0^2
97     # s0 = 1; a.t = 2.5/2; b.t = 2.5/2 * s0^2
98     tau2 <- 6 # tuning parameter sigma^2
99
100    phi <- rdirichlet(1, phi.alpha)
101    varphi <- rgamma(1, a.vphi, b.vphi)
102    tau.r <- phi * varphi
103
104    # tensor
105    lambda <- matrix(rgamma(rank * D, a.lam[1], b.lam[1]), rank, D)
106    omega <- lapply(1:D, function(x) array(rexp(rank * P[x], .5 * (a.
        lam[1] / b.lam[1])), dim = c(rank, P[x])))

```

```

107     beta <- lapply(1:D, function(x) array(rnorm(rank * P[x]), dim = c(
        rank, P[x])))
108 B <- lapply(1:rank, function(x) array(NA, dim = P))
109 # latent parameter
110 rho <- matrix(0, N, 1)
111
112 tens.mean <- getmean(x.train.nona, beta, rank) # <X,B>
113 if (!is.null(Z)) {
114     pred.mean <- Zt %*% gam
115 } else {
116     pred.mean <- as.matrix(rep(0, length(Yi)))
117 }
118 yest <- pred.mean + tens.mean
119
120 rho_store <- matrix(0, N, nsweep)
121 mu_store <- matrix(0, N, nsweep)
122 alpha_store <- rep(NA, nsweep)
123 gam_store <- array(data = NA, dim = c(nsweep, pgamma))
124 tau2_store <- rep(NA, nsweep)
125 phi_store <- array(data = NA, dim = c(nsweep, rank))
126 varphi_store <- array(data = NA, dim = c(nsweep, 1))
127 beta_store <- lapply(1:nsweep, function(x) lapply(1:D, function(y)
        array(dim = c(rank, P[y]))))
128 omega_store <- lapply(1:nsweep, function(x) lapply(1:D, function(y)
        ) array(dim = c(rank, P[y]))))
129 lambda_store <- array(data = NA, dim = c(nsweep, rank, D))
130 hyppar_store <- array(data = NA, dim = c(nsweep, rank, 2))
131
132 par.grid <- expand.grid(
133   alam = seq(2.1, D + 1, length.out = 5),
134   zeta = seq(0.5, ceiling(10 * rank**(1 / (2 * D))) / 2) / 10, length
        .out = 5)
135 )
136 # alpha.grid <- exp(log(rank) * seq(-d, -1/(2*d), length.out = 10)
        ); M <- 10
137 # alpha.grid <- exp(log(rank) * seq(-d, -0.1, length.out = 10)); M
        <- 20
138 alpha.grid <- seq(rank**(-D), rank**(-0.1), length.out = 10)
139 M <- 20

```

```

140     score.store <- array(data = NA, dim = c(n sweep, length(alpha.grid)
141     ))
142
143     # MCMC
144     tt <- Sys.time()
145     for (sweep in 1:n sweep) {
146         # Step 1 in Algorithm 1
147         for (sub in 1:N) {
148             rhoi_mu <- abs(1 - obs[sub] * yest[sub])^(-1)
149             if (rhoi_mu == Inf) {
150                 rhoi_mu <- 1e2
151             }
152             rhoi <- rinvgauss(1, mean = rhoi_mu, shape = 1 /
153                 tau2)
154             rho[sub] <- rhoi^(-1)
155         }
156         # store rho
157         rho_store[, sweep] <- rho
158
159         ## 对 Gamma 采样
160         if (!is.null(Z)) {
161             ZZ <- crossprod(Zt, Zt)
162             Z_rho <- matrix(NA, N, pgamma)
163             for (i in 1:N) {
164                 Z_rho[i, ] <- Zt[i, ] / sqrt(rho[i]) # G
165             }
166             ZZ_rho <- crossprod(Z_rho, Z_rho) # G^T * G
167
168             Sig.g <- chol2inv(chol(diag(pgamma) + ZZ_rho /
169                 tau2))
170             mu.g <- Sig.g %*% (crossprod(Zt * obs, (rho + 1 -
171                 obs * tens.mean) / (rho * tau2)))
172             gam <- mu.g + chol(Sig.g) %*% rnorm(pgamma)
173         } else {
174             gam <- 0
175         }
176         ## 更新 pred.mean
177         if (!is.null(Z)) {
178             pred.mean <- Zt %*% gam
179         } else {

```



```

176         pred.mean <- as.matrix(rep(0, length(Yi)))
177     }
178
179     ## 更新 (a.lam, b.lam)
180     Cjr <- sapply(1:rank, function(rr) {
181         bb <- sapply(1:D, function(jj) sum(abs(beta[[jj]][
182             rr, ])))
183         bb <- bb / sqrt(tau.r[rr])
184         return(bb)
185     })
186     mfun <- function(z, rank) {
187         o <- sapply(1:D, function(x) {
188             return(lgamma(z[1] + P[x]) - lgamma(z[1])
189                 + z[1] * log(z[2] * z[1]) - (z[1] + P[x]
190                 ) * log(z[2] * z[1] + Cjr[x, rank]))
191         })
192         return(sum(o))
193     }
194     ll <- sapply(1:rank, function(rr) apply(par.grid, 1, mfun,
195         rank = rr))
196     par.wt <- apply(ll, 2, function(z) {
197         return(exp(z - logsum(z)))
198     })
199     ixr <- apply(par.wt, 2, sample, x = c(1:nrow(par.grid)),
200         size = 1, replace = F)
201     for (rr in 1:rank) {
202         a.lam[rr] <- par.grid[ixr[rr], 1]
203         b.lam[rr] <- par.grid[ixr[rr], 2] * a.lam[rr]
204     }
205
206     ## 更新 (alpha, phi, varphi)
207     draw.phi_tau <- function(alpha) {
208         len <- length(alpha)
209
210         m.phialpha <- rep(alpha[1], rank)
211         m.phia0 <- sum(m.phialpha)
212         m.avphi <- m.phia0
213         ## assumes b.vphi const (use: alpha 1 / R)
214
215         Cr <- sapply(1:rank, function(rr) {

```

```

211         bb <- sapply(1:D, function(jj) {
212             crossprod(
213                 beta[[jj]][rr, ],
214                 diag(1 / omega[[jj]][rr, ]) %*%
215                     beta[[jj]][rr, ]
216             )
217         })
218     return(bb)
219 })
220 score.fn <- function(phi.alpha, phi.s, varphi.s,
221     Cstat) {
222     ldirdens <- function(v, a) {
223         c1 <- lgamma(sum(a))
224         c2 <- sum(lgamma(a))
225         return((c1 - c2) + sum((a - 1) *
226             log(v)))
227     }
228     ldir <- apply(phi.s, 1, ldirdens, a = phi.
229         alpha)
230     lvarphi <- dgamma(varphi.s, sum(phi.alpha)
231         , b.vphi, log = T)
232     dnorm.log <- -rowSums(Cstat) / (2 * varphi
233         .s) - (sum(P) / 2) * sapply(1:length(
234         varphi.s), function(ii) {
235         return(sum(log(varphi.s[ii] * phi.
236             s[ii, ])))
237         })
238     return(dnorm.log + ldir + lvarphi)
239 }
240
241 phi <- NULL
242 varphi <- NULL
243 scores <- NULL
244 if (len > 1) {
245     phi <- matrix(0, M * length(alpha.grid),
246         rank)
247     varphi <- matrix(0, M * length(alpha.grid)
248         , 1)

```

```

241         Cstat <- matrix(0, M * length(alpha.grid),
242                           rank)
243
244         scores <- list()
245
246         ## get reference set
247         for (jj in 1:len) {
248             m.phialpha <- rep(alpha[jj], rank)
249             m.phia0 <- sum(m.phialpha)
250             m.avphi <- m.phia0
251
252             ## draw phi
253             Cr1 <- colSums(Cr)
254             phi.a <- sapply(1:rank, function(
255                 rr) {
256                 rgig(M, m.phialpha[rr] -
257                     sum(P) / 2, Cr1[rr], 2
258                     * b.vphi)
259             })
260             phi.a <- t(apply(phi.a, 1,
261                             function(z) {
262                                 return(z / sum(z))
263                             })) ## [M x rank]
264
265             ## draw varphi ##colSums(Cr / t(
266                 replicate(d, z)))
267             Cr2 <- t(apply(phi.a, 1, function(
268                 z) {
269                 return(Cr1 / z)
270             }))
271             varphi.a <- apply(Cr2, 1, function
272                 (z) {
273                 return(rgig(1, m.avphi -
274                     rank * sum(P) / 2, sum(
275                         z), 2 * b.vphi))
276             })
277             phi[seq((jj - 1) * M + 1, jj * M),
278                ] <- phi.a
279             varphi[seq((jj - 1) * M + 1, jj *
280                M)] <- varphi.a

```

```

268         Cstat[seq((jj - 1) * M + 1, jj * M
269             ), ] <- Cr2
270     }
271     scores <- lapply(alpha.grid, function(z) {
272         return(score.fn(rep(z, rank), phi,
273             varphi, Cstat))
274     })
275     lmax <- max(unlist(scores))
276     scores <- sapply(scores, function(z) {
277         return(mean(exp(z - lmax)))
278     })
279 } else {
280     ## draw phi
281     Cr1 <- colSums(Cr)
282     phi <- sapply(1:rank, function(rr) {
283         rgig(1, m.phialpha[rr] - sum(P) /
284             2, Cr1[rr], 2 * b.vphi)
285     })
286     phi <- phi / sum(phi)
287
288     ## draw varphi
289     Cr2 <- Cr1 / phi
290     varphi <- rgig(1, m.avphi - rank * sum(P)
291         / 2, sum(Cr2), 2 * b.vphi)
292 }
293 return(list(phi = phi, varphi = varphi, scores =
294     scores))
295 }
296
297 ## sample astar
298 o <- draw.phi_tau(alpha.grid)
299 astar <- sample(alpha.grid, size = 1, prob = o$scores)
300 score <- o$scores / sum(o$scores)
301 # cat(sprintf('scores: %s\n', paste(round(score, 2),
302     collapse = ', ')))
303 score.store[sweep, ] <- score
304
305 ## sample (phi, varphi)
306 o <- draw.phi_tau(astar)
307 phi <- o$phi

```

```

302     varphi <- o$varphi
303     tau.r <- varphi * phi
304     phi.alpha <- rep(aster, rank)
305     phi.a0 <- sum(phi.alpha)
306     a.vphi <- phi.a0
307
308     # update rank specific params
309
310     # update rank specific params
311     for (r in 1:rank) {
312         for (j in 1:D) {
313             tens.mu.r <- getmean(x.train.nona, beta,
314                                   rank, r)
315
316             betj <- getouter_list(lapply(beta[-j],
317                                           function(x) x[r, ]))
318             H <- matrix(NA, N, P[j])
319             H_rho <- matrix(NA, N, P[j])
320             for (i in 1:N) {
321                 if (D == 2) {
322                     H[i, ] <- apply(x.train.nona[i, , ], j,
323                                     function(x) {
324                     321                     return(sum(x *
325                                     betj))
326                                     })
327                     H_rho[i, ] <- H[i, ] /
328                         sqrt(rho[i]) # H
329                 } else if (D == 3) {
330                     H[i, ] <- apply(x.train.nona[i, , , ], j,
331                                     function(x) {
326                     return(sum(x *
327                                     betj))
328                                     })
329                     H_rho[i, ] <- H[i, ] /
330                         sqrt(rho[i])
331                 }
332             }
333             HH_rho <- crossprod(H_rho, H_rho) # H^T*H

```

```

332      # K: Sigma_jr
333      K <- chol2inv(chol(HH_rho / tau2 + diag(1
334                    / omega[[j]][r, ]) / tau.r[r]))
335      # mm: y tilda
336      mm <- rho + 1 - obs * pred.mean - obs *
337          tens.mu.r
338      # posterior mean
339      bet.mu.jr <- K %*% crossprod(H / tau2, mm
340                                   * obs / rho)
341      # update beta_jr
342      beta[[j]][r, ] <- bet.mu.jr + chol(K) %*%
343          rnorm(P[j])
344
345      ## update lambda.jr
346      lambda[r, j] <- rgamma(1, a.lam[r] + P[j],
347                             b.lam[r] + sum(abs(beta[[j]][r, ])) /
348                             sqrt(tau.r[r]))
349      ## update omega.jr
350      omega[[j]][r, ] <- sapply(1:P[j], function
351                                (kk) rgig(1, 1 / 2, beta[[j]][r, kk]^2
352                                           / tau.r[r], lambda[r, j]^2))
353      # omega[r,j,] <- sapply(1:p, function(kk){
354          a <- lambda[r,j]^2; b <- beta[[r]][kk,j
355          ]^2 / tau.r[r]; map <- besselK(sqrt(a*b)
356          ),0.5 + 1) / besselK(sqrt(a*b), 0.5) *
357          sqrt(b / a); return(map)})
358      }
359      }
360
361      ## Update <X,B> and Z*gamma
362      tens.mean <- getmean(x.train.nona, beta, rank)
363      if (!is.null(Z)) {
364          pred.mean <- Zt %*% gam
365      }
366
367      ## store params
368      beta.store[[sweep]] <- beta
369      tau2.store[sweep] <- tau2
370      if (!is.null(Z)) {
371          gam.store[sweep, ] <- gam

```

```

360         } else {
361             gam.store[sweep] <- gam
362         }
363         alpha.store[sweep] <- astar ## not intercept
364         phi.store[sweep, ] <- phi
365         varphi.store[sweep, ] <- varphi
366         omega.store[[sweep]] <- omega
367         lambda.store[sweep, , ] <- lambda
368         sapply(1:rank, function(rr) hyppar.store[sweep, rr, ] <-
                 c(a.lam[rr], b.lam[rr]))
369
370         cat("iteration: ", sweep, "\n")
371     }
372
373     tt <- abs(tt - Sys.time())
374     cat("Time out: ", tt, "\n")
375
376     ##### finalize #####
377     out <- list(
378         nsweep = nsweep,
379         rank = rank,
380         mu_store = mu_store,
381         P = P,
382         D = D,
383         rho_store = rho_store,
384         par.grid = par.grid, alpha.grid = alpha.grid, my = my, sy = sy, mz
            = mz, sz = sz, mx = mx, sx = sx, Zt = Zt, Xt = Xt, obs = obs,
            a.t = a.t, b.t = b.t, gam.store = gam.store, alpha.store =
            alpha.store, beta.store = beta.store, phi.store = phi.store,
            varphi.store = varphi.store, omega.store = omega.store, lambda.
            store = lambda.store, hyppar.store = hyppar.store, score.store
            = score.store, time = tt
385     )
386
387     class(out) <- "tensor.reg"
388     return(out)
389 }
390
391
392 # aux functions

```

```

393
394 getouter_list <- function(bet) {
395     D <- length(bet)
396     if (D == 1) {
397         return(bet[[1]])
398     }
399     if (D == 2) {
400         return(outer(bet[[1]], bet[[2]]))
401     } else {
402         return(outer(getouter_list(bet[1:(D - 1)]), bet[[D]]))
403     }
404 }
405
406 TP.rankR <- function(X.allr) {
407     R <- ncol(X.allr[[1]])
408     if (is.null(R)) {
409         return(getouter_list(X.allr))
410     } else {
411         Y <- array(0, dim = c(as.numeric(lapply(X.allr, function(x)
412             length(x[, 1])))))
413         for (r in c(1:R)) {
414             Y <- Y + getouter_list(lapply(X.allr, function(x)
415                 x[, r]))
416         }
417         return(Y)
418     }
419 }
420
421 getmean <- function(X, beta, rank, rank.exclude = NULL) {
422     idx <- setdiff(1:rank, rank.exclude)
423     B <- Reduce("+", lapply(idx, function(r) getouter_list(lapply(beta
424         , function(x) x[r, ]))))
425     mu.B <- apply(X, 1, function(xx, bb) sum(xx * bb), bb = B)
426     return(mu.B)
427 }
428
429 tensor.mean <- function(x, n) {
430     Reduce("+", x) / n
431 }

```



```

430 logsum <- function(lx) {
431     return(max(lx) + log(sum(exp(lx - max(lx)))))
432 }
433
434 getBeta_mcmc <- function(beta.store) {
435     nsweep <- length(beta.store)
436     D <- length(beta.store[[1]])
437     rank <- nrow(beta.store[[1]][[1]])
438     P <- sapply(1:D, function(x) ncol(beta.store[[1]][[x]]))
439     Beta_mcmc <- array(dim = c(nsweep, prod(P)))
440     for (i in 1:nsweep) {
441         coef <- rep(0, prod(P))
442         for (r in 1:rank) {
443             coef <- coef + c(getouter_list(lapply(beta.store[[
444                 i]], function(x) x[r, ])))
445             Beta_mcmc[i, ] <- coef
446         }
447         quantile(Beta_mcmc)
448         return(Beta_mcmc)
449     }
450
451 add <- function(x) Reduce("+", x)
452
453 uncollapse <- function(str, collapse = "", mode = "character") {
454     a <- unlist(strsplit(str, collapse))
455     mode(a) <- mode
456     return(a)
457 }
458
459 rmse <- function(y, yhat) {
460     return(sqrt(mean((y - yhat)^2, na.rm = T)))
461 }
462
463 logsum <- function(lx) {
464     return(max(lx) + log(sum(exp(lx - max(lx)))))
465 }
466
467 # 构造 rank-R PARAFAC 张量
468 generate_parafac_tensor <- function(p, R, prob = 0.2, size = 2) {

```

```

469     A_list <- list()
470     B_list <- list()
471
472     for (r in 1:R) {
473         a_r <- rbinom(p[1], size = size, prob = prob)
474         b_r <- rbinom(p[2], size = size, prob = prob)
475         A_list[[r]] <- a_r
476         B_list[[r]] <- b_r
477     }
478
479     tensor <- matrix(0, nrow = p[1], ncol = p[2])
480     for (r in 1:R) {
481         tensor <- tensor + outer(A_list[[r]], B_list[[r]])
482     }
483
484     # 标准化为最大值为1
485     tensor <- tensor / max(tensor)
486     return(tensor)
487 }
488
489 load_images <- function(image_dir, target_size = c(48, 48)) {
490     images <- list()
491     labels <- c()
492
493     for (label in c("yes", "no")) {
494         folder_path <- file.path(image_dir, label)
495         label_value <- ifelse(label == "yes", 1, -1)
496
497         image_files <- list.files(folder_path, full.names = TRUE)
498
499         for (file in image_files) {
500             img <- load.image(file)
501
502             # 如果是RGB彩色图，则转为灰度图
503             if (spectrum(img) == 3) {
504                 img <- grayscale(img)
505             }
506
507             img_resized <- resize(img, target_size[1], target_size[2])

```

```

508         img_matrix <- as.array(img_resized)
509
510         # 转为 (height, width, 1) 格式
511         img_array <- array(img_matrix[, , 1, 1], dim = c(
512             target_size[2], target_size[1], 1))
513
514         images[[length(images) + 1]] <- img_array
515         labels <- c(labels, label_value)
516     }
517 }
518
519 # 去掉通道维度, 转为 (N, 48, 48)
520 images_tensor <- abind(images, along = 0)[, , , 1]
521
522 return(list(images = images_tensor, labels = labels))
523 }
524 # set.seed(123)
525
526 # ==== Scenario 1 ====
527 Beta_tens <- generate_parafac_tensor(p = c(48, 48), R = 3)
528
529 # # ==== Scenario 2 ====
530 # a1 <- rep(0, 48)
531 # a1[38:48] <- 1
532 # b1 <- rep(0, 48)
533 # b1[5:12] <- 1
534
535 # a2 <- rep(0, 48)
536 # a2[38:48] <- 1
537 # b2 <- rep(0, 48)
538 # b2[33:43] <- 1
539
540 # a3 <- rep(0, 48)
541 # a3[5:12] <- 1
542 # b3 <- rep(0, 48)
543 # b3[c(5:12, 33:43)] <- 1
544
545 # Beta_tens <- outer(a1, b1) + outer(a2, b2) + outer(a3, b3)
546

```

```

547 # # ==== Scenario 3 ====
548 # Beta_tens <- matrix(0, 48, 48)
549 # for (i in 15:40) {
550     # for (j in 10:35) {
551         # Beta_tens[i, j] <- 1
552     # }
553 # }
554
555 # # ==== Scenario 4 ====
556 # Beta_tens <- matrix(0, 48, 48)
557
558 # # 圆的中心和半径
559 # center_x <- 18
560 # center_y <- 18
561 # radius <- sqrt(0.10 * 48 * 48 / pi)
562
563 # # 填充圆形区域为 1
564 # for (i in 1:48) {
565     # for (j in 1:48) {
566         # # 计算 (i, j) 到圆心的距离
567         # if ((i - center_x)^2 + (j - center_y)^2 <= radius^2)
568             # Beta_tens[i, j] <- 1
569         # }
570     # }
571 # }
572
573 png("Beta_tens_Scenario4.png", width = 600, height = 600)
574 image.plot(Beta_tens,
575 col = gray.colors(25, start = 1, end = 0), axes = FALSE,
576 main = "True Tensor Coefficient"
577 )
578 mtext(text = seq(10, 50, 10), side = 2, line = 0.3, at = seq(10, 50, 10) /
579       48, las = 1, cex = 0.8)
580 mtext(text = seq(10, 50, 10), side = 1, line = 0.3, at = seq(10, 50, 10) /
581       48, las = 2, cex = 0.8)
582 dev.off()
583
584 # 生成 2D 图像和二元相应变量
585 N <- 500

```

```

584 p <- c(48, 48)
585 rank <- 3
586 X <- array(rnorm(N * prod(p)), dim = c(N, p))
587 Y <- sapply(1:N, function(x) sum(X[x, , ] * Beta_tens, na.rm = T))
588 hist(Y)
589 # # SVM Loss
590 # Ylabel <- rep(0, N)
591 # Ylabel[Y >= 0] <- 1
592 # Ylabel[Y < 0] <- -1
593
594 # Lasso loss
595 p <- 1 / (1 + exp(-Y))
596 Ylabel <- ifelse(p > 0.5, 1, -1)
597
598 # 按照 7: 3 的比例划分训练集和测试集
599 train_index <- sort(sample(1:N, 0.7 * N, replace = FALSE))
600 x.train <- X[train_index, , ]
601 y.train <- Ylabel[train_index]
602 x.test <- X[-train_index, , ]
603 y.test <- Ylabel[-train_index]
604
605 # ==== 脑肿瘤数据 ====
606 # # 设置路径
607 # train_dir <- "E:/BRAIN_TUMOR/BRAIN_TUMOR/train"
608 # val_dir <- "E:/BRAIN_TUMOR/BRAIN_TUMOR/val"
609
610 # # 加载数据
611 # train_data <- load_images(train_dir)
612 # val_data <- load_images(val_dir)
613
614 # # 合并数据
615 # all_images <- abind(train_data$images, val_data$images, along = 1)
616 # all_labels <- c(train_data$labels, val_data$labels)
617
618 # N <- dim(all_images)[1] # 总样本数
619 # train_index <- sort(sample(1:N, 0.7 * N, replace = FALSE))
620
621 # x.train <- all_images[train_index, , ]
622 # y.train <- all_labels[train_index]
623 # x.test <- all_images[-train_index, , ]

```

```

624 # y.test <- all_labels[-train_index]
625 # # 归一化
626 # x.train <- x.train / 255
627 # x.test <- x.test / 255
628
629 burnin <- 1000
630 nsweep <- 3000
631
632 sim <- SVMdpl(Z = NULL, x.train, y.train, nsweep = nsweep, rank = 3, nskip
    = nskip, scale = T)
633 # tensor_est: estimated tensor coefficient
634 tensor <- getBeta_mcmc(sim$beta.store)
635 tensor_est <- apply(tensor[burnin:nsweep, ], 2, mean) * sim$sy / sim$sx
636 rmse_val <- rmse(c(Beta_tens), c(tensor_est))
637 cor_val <- cor(c(Beta_tens), c(tensor_est))
638 cat(sprintf("张量系数估计的 RMSE(均方根误差为: %.6f\n", rmse_val))
639 cat(sprintf("张量系数估计与真实值的相关系数为: %.6f\n", cor_val))
640 # plot estimated tensor coefficient
641 png("Tensor_estimated_Scenario4_SVM.png", width = 600, height = 600)
642 image.plot(tensor_est, col = gray.colors(25, start = 1, end = 0), axes = F
    )
643 mtext(text = seq(10, 50, 10), side = 2, line = 0.3, at = seq(10, 50, 10) /
    48, las = 1, cex = 0.8)
644 mtext(text = seq(10, 50, 10), side = 1, line = 0.3, at = seq(10, 50, 10) /
    48, las = 2, cex = 0.8)
645 dev.off()
646
647 residual_map <- Beta_tens - matrix(tensor_est, nrow = 48)
648 png("Residual_Tensor_Scenario4_SVM.png", width = 600, height = 600)
649 image.plot(residual_map,
650 col = topo.colors(25), axes = FALSE,
651 main = "Residual Map: Beta_tens - tensor_est"
652 )
653 dev.off()
654
655 # get misclassification rate
656 post.tens.mean.test <- array(0, N - length(train_index))
657 for (j in 1:(N - length(train_index))) {
658     post.tens.mean.test[j] <- c(tensor_est) %*% c(x.test[j, , ])
659 }

```

```

660 post.mui.test <- matrix(post.tens.mean.test, ncol = 1)
661 clust.test <- rep(0, N - length(train_index))
662 clust.test[post.mui.test > 0] <- 1
663 clust.test[post.mui.test <= 0] <- -1
664 missclassrate <- 1 - sum(clust.test == y.test) / length(clust.test)
665 cat(sprintf("测试集误分类率为: %.4f\n", missclassrate))
666 # get fl score
667 TP <- 0
668 FP <- 0
669 FN <- 0
670 for (j in 1:length(clust.test)) {
671     if (clust.test[j] == 1 && y.test[j] == 1) {
672         TP <- TP + 1
673     } else if (clust.test[j] == 1 && y.test[j] == -1) {
674         FP <- FP + 1
675     } else if (clust.test[j] == -1 && y.test[j] == 1) {
676         FN <- FN + 1
677     }
678 }
679 flscore <- TP / (TP + (FP + FN) / 2)
680 cat(sprintf("F1-score 为: %.4f\n", flscore))

```

Listing A.2 BT-LR 算法

```

1 library(fields)
2 library(abind)
3 library(imager)
4 library(magrittr)
5 PGtensor <- function(Z, X, Ylabel, n = rep(1, length(Ylabel)), rank = 3,
6     nsweep = 1e3, nskip = 3, a.lam, b.lam, phi.alpha, scale = TRUE) {
7     library(base)
8     library(statmod)
9     library(MASS)
10    library(matlib)
11    library(stats)
12    library(plyr)
13    library(GIGrv)
14    library(gtools)
15    library(coda)
16    library(fields)
17    library(dplyr)

```

```

17     library(BayesLogit)
18
19     ## data input
20     # N : total number of subjects
21     # P: Size of tensor images
22     # D: Dimension of tensor images (2D/3D)
23     # Z : N * pgamma matrix of demographic covariates
24     # X  : N * P[1] * P[2] tensor images
25     # Ylabel : N binary response
26     # n_i = 1
27
28     Yi <- Ylabel
29     N <- length(Yi)
30     P <- dim(X)[-1]
31     D <- length(dim(X)) - 1
32     pgamma <- ncol(Z)
33
34     # 标准化
35     if (is.null(Z)) {
36         mz <- NA
37         sz <- NA
38     } else {
39         mz <- colMeans(Z)
40         sz <- rep(1, pgamma)
41     }
42     if (!is.null(Z)) {
43         if (!is.matrix(Z)) {
44             Z <- as.matrix(Z)
45         }
46     }
47     Zt <- Z
48     obs <- as.numeric(Yi)
49     sy <- 1
50     my <- 0
51
52     if (scale) { ## centering & scale to have unity range
53         if (!is.null(Z)) {
54             mz <- colMeans(Z)
55             sz <- apply(Z, 2, function(z) diff(range(z)))
56             sz[sz == 0] <- 1

```



```

57         Zt <- Z
58         for (jj in 1:pgamma) {
59             Zt[, jj] <- (Z[, jj] - mz[jj]) / sz[jj]
60         }
61     }
62
63     Xt <- 0 * X
64     mx <- apply(X, c(2:(D + 1)), function(z) mean(z, na.rm = T
65         ))
66     sx <- apply(X, c(2:(D + 1)), function(z) diff(range(z, na.
67         rm = T)))
68     sx[sx == 0] <- 1
69     if (D == 2) {
70         for (jj in 1:nrow(X)) Xt[jj, , ] <- (X[jj, , ] -
71             mx) / sx
72     } else if (D == 3) {
73         for (jj in 1:nrow(X)) Xt[jj, , , ] <- (X[jj, , , ]
74             - mx) / sx
75     }
76     } else {
77         ## do nothing;
78         if (!is.null(Z)) {
79             mz <- rep(0, pgamma)
80             sz <- rep(1, pgamma)
81             Zt <- Z
82         }
83         mx <- array(0, dim = dim(X)[-1])
84         sx <- array(1, dim = dim(X)[-1])
85         Xt <- X
86     }
87
88     # MCMC setup
89
90     # require(glmnet)
91     x.train.nona <- Xt
92     x.train.nona[is.na(Xt)] <- 0
93
94     ## hyper-par initialize
95     if (missing(a.lam)) a.lam <- rep(3, rank)
96     if (missing(b.lam)) b.lam <- (a.lam)**(1 / (2 * D))

```

```

93     if (missing(phi.alpha)) phi.alpha <- rep(1 / rank, rank)
94
95     phi.a0 <- sum(phi.alpha)
96     a.vphi <- phi.a0
97     b.vphi <- phi.alpha[1] * rank^(1 / D)
98
99     s0 <- 1
100    a.t <- 0.1
101    b.t <- 1 # b.t = 2.5/2 * s0^2
102    # s0 = 1; a.t = 2.5/2; b.t = 2.5/2 * s0^2
103
104    phi <- rdirichlet(1, phi.alpha)
105    varphi <- rgamma(1, a.vphi, b.vphi)
106    tau.r <- phi * varphi
107
108    ## Storage/Posterior Quantities
109    if (is.null(Z)) {
110        gam <- 0
111    } else {
112        gam <- rep(0, pgamma)
113    }
114
115    lambda <- matrix(rgamma(rank * D, a.lam[1], b.lam[1]), rank, D)
116    omega <- lapply(1:D, function(x) array(rexp(rank * P[x], .5 * (a.
117        lam[1] / b.lam[1])), dim = c(rank, P[x])))
118
119    beta <- lapply(1:D, function(x) array(rnorm(rank * P[x]), dim = c(
120        rank, P[x]))) # D*rank*P_d
121
122    ### Poly-gamma Parameters
123    # Yi is 1/0 response, n is param from binom dist
124    kappa <- (Yi - 1 / 2) * n
125    w <- rep(0, N)
126
127    tens.mean <- getmean(x.train.nona, beta, rank) # <X,B>
128    if (!is.null(Z)) {
129        pred.mean <- Zt %*% gam
130    } else {
131        pred.mean <- as.matrix(rep(0, length(Yi)))
132    }
133    yest <- pred.mean + tens.mean

```

```

131
132   # Storage/Posterior Quantities
133   w.store <- matrix(0, N, nsweep)
134   mu_store <- matrix(0, N, nsweep)
135   alpha.store <- rep(NA, nsweep)
136   gam.store <- array(data = NA, dim = c(nsweep, pgamma))
137   phi.store <- array(data = NA, dim = c(nsweep, rank))
138   varphi.store <- array(data = NA, dim = c(nsweep, 1))
139   beta.store <- lapply(1:nsweep, function(x) lapply(1:D, function(y)
        array(dim = c(rank, P[y]))))
140   omega.store <- lapply(1:nsweep, function(x) lapply(1:D, function(y)
        ) array(dim = c(rank, P[y]))))
141   lambda.store <- array(data = NA, dim = c(nsweep, rank, D))
142   hyppar.store <- array(data = NA, dim = c(nsweep, rank, 2))
143
144   # create a data frame
145   par.grid <- expand.grid(
146     alam = seq(2.1, D + 1, length.out = 5),
147     zeta = seq(0.5, ceiling(10 * rank**(1 / (2 * D))) / 2) / 10, length
        .out = 5)
148   )
149   # alpha.grid <- exp(log(rank) * seq(-d, -1/(2*d), length.out = 10)
        ); M <- 10
150   # alpha.grid <- exp(log(rank) * seq(-d, -0.1, length.out = 10)); M
        <- 20
151   alpha.grid <- seq(rank**(-D), rank**(-0.1), length.out = 10)
152   M <- 20
153   score.store <- array(data = NA, dim = c(nsweep, length(alpha.grid)
        ))
154
155   # MCMC
156   tt <- Sys.time()
157   for (sweep in 1:nsweep) {
158
159       # Sample w, the Polya-gamma parameter
160       psi <- drop(yest)
161       for (sub in 1:N) {
162           w <- rpg.devroye(N, n, psi)
163       }
164       w.store[, sweep] <- w

```

```

165
166     ## Sample Gamma
167     if (!is.null(Z)) {
168         Z_w <- matrix(NA, N, pgamma)
169         for (i in 1:N) {
170             Z_w[i, ] <- Zt[i, ] * sqrt(w[i]) # G
171         }
172         ZZ_w <- crossprod(Z_w, Z_w) # G^T * G
173
174         Sig.g <- chol2inv(chol(diag(pgamma) + ZZ_w))
175         mu.g <- Sig.g %*% (crossprod(Zt * w, (kappa / w -
176             tens.mean))) # incorporate rho in mean
177         gam <- mu.g + chol(Sig.g) %*% rnorm(pgamma)
178     } else {
179         gam <- 0
180     }
181     ## Update pred.mean
182     if (!is.null(Z)) {
183         pred.mean <- Zt %*% gam
184     } else {
185         pred.mean <- as.matrix(rep(0, length(Yi)))
186     }
187     ## update (a.lam, b.lam)
188     Cjr <- sapply(1:rank, function(rr) {
189         bb <- sapply(1:D, function(jj) sum(abs(beta[[jj]][[
190             rr, ]]))
191         bb <- bb / sqrt(tau.r[rr])
192         return(bb)
193     })
194     mfun <- function(z, rank) {
195         o <- sapply(1:D, function(x) {
196             return(lgamma(z[1] + P[x]) - lgamma(z[1])
197                 + z[1] * log(z[2] * z[1]) - (z[1] + P[x]
198                 ) * log(z[2] * z[1] + Cjr[x, rank]))
199         })
200         return(sum(o))
201     }
202     ll <- sapply(1:rank, function(rr) apply(par.grid, 1, mfun,
203         rank = rr))

```

```

200     par.wt <- apply(11, 2, function(z) {
201         return(exp(z - logsum(z)))
202     })
203     ixx <- apply(par.wt, 2, sample, x = c(1:nrow(par.grid)),
204         size = 1, replace = F)
205     for (rr in 1:rank) {
206         a.lam[rr] <- par.grid[ixx[rr], 1]
207         b.lam[rr] <- par.grid[ixx[rr], 2] * a.lam[rr]
208     }
209     ## update (alpha, phi, varphi)
210     draw.phi_tau <- function(alpha) {
211         len <- length(alpha)
212
213         m.phialpha <- rep(alpha[1], rank)
214         m.phia0 <- sum(m.phialpha)
215         m.avphi <- m.phia0
216         ## assumes b.vphi const (use: alpha 1 / R)
217
218         Cr <- sapply(1:rank, function(rr) {
219             bb <- sapply(1:D, function(jj) {
220                 crossprod(
221                     beta[[jj]][rr, ],
222                     diag(1 / omega[[jj]][rr, ]) %*%
223                     beta[[jj]][rr, ]
224                 )
225             })
226             return(bb)
227         })
228         score.fn <- function(phi.alpha, phi.s, varphi.s,
229             Cstat) {
230             ldirdens <- function(v, a) {
231                 c1 <- lgamma(sum(a))
232                 c2 <- sum(lgamma(a))
233                 return((c1 - c2) + sum((a - 1) *
234                     log(v)))
235             }
236             ldir <- apply(phi.s, 1, ldirdens, a = phi.
237                 alpha)

```

```

235         lvarphi <- dgamma(varphi.s, sum(phi.alpha)
236           , b.vphi, log = T)
237
238         dnorm.log <- -rowSums(Cstat) / (2 * varphi
239           .s) - (sum(P) / 2) * sapply(1:length(
240           varphi.s), function(ii) {
241             return(sum(log(varphi.s[ii] * phi.
242               s[ii, ])))
243           })
244         return(dnorm.log + ldir + lvarphi)
245       }
246
247   phi <- NULL
248   varphi <- NULL
249   scores <- NULL
250   if (len > 1) {
251     phi <- matrix(0, M * length(alpha.grid),
252       rank)
253     varphi <- matrix(0, M * length(alpha.grid)
254       , 1)
255     Cstat <- matrix(0, M * length(alpha.grid),
256       rank)
257     scores <- list()
258
259     ## get reference set
260     for (jj in 1:len) {
261       m.phialpha <- rep(alpha[jj], rank)
262       m.phia0 <- sum(m.phialpha)
263       m.avphi <- m.phia0
264
265       ## draw phi
266       Cr1 <- colSums(Cr)
267       phi.a <- sapply(1:rank, function(
268         rr) {
269         rgig(M, m.phialpha[rr] -
270           sum(P) / 2, Cr1[rr], 2
271           * b.vphi)
272       })
273       phi.a <- t(apply(phi.a, 1,
274         function(z) {

```

```

264         return(z / sum(z))
265     ))) ## [M x rank]
266
267     ## draw varphi ##colSums(Cr / t(
268         replicate(d, z)))
269     Cr2 <- t(apply(phi.a, 1, function(
270         z) {
271         return(Cr1 / z)
272     })))
273     varphi.a <- apply(Cr2, 1, function
274         (z) {
275         return(rgig(1, m.avphi -
276             rank * sum(P) / 2, sum(
277                 z), 2 * b.vphi))
278     })
279     phi[seq((jj - 1) * M + 1, jj * M),
280         ] <- phi.a
281     varphi[seq((jj - 1) * M + 1, jj *
282         M)] <- varphi.a
283     Cstat[seq((jj - 1) * M + 1, jj * M
284         ), ] <- Cr2
285     }
286     scores <- lapply(alpha.grid, function(z) {
287         return(score.fn(rep(z, rank), phi,
288             varphi, Cstat))
289     })
290     lmax <- max(unlist(scores))
291     scores <- sapply(scores, function(z) {
292         return(mean(exp(z - lmax)))
293     })
294     } else {
295         ## draw phi
296         Cr1 <- colSums(Cr)
297         phi <- sapply(1:rank, function(rr) {
298             rgig(1, m.phialpha[rr] - sum(P) /
299                 2, Cr1[rr], 2 * b.vphi)
300         })
301         phi <- phi / sum(phi)
302
303         ## draw varphi

```

```

294         Cr2 <- Cr1 / phi
295         varphi <- rgig(1, m.avphi - rank * sum(P)
                        / 2, sum(Cr2), 2 * b.vphi)
296     }
297     return(list(phi = phi, varphi = varphi, scores =
                scores))
298 }
299
300 ## sample astar
301 o <- draw.ph_i_tau(alpha.grid)
302 astar <- sample(alpha.grid, size = 1, prob = o$scores)
303 score <- o$scores / sum(o$scores)
304 # cat(sprintf('scores: %s\n', paste(round(score <- o$
        scores/sum(o$scores), 2), collapse = ', ')))
305 score.store[sweep, ] <- score
306
307 ## sample (phi, varphi)
308 o <- draw.ph_i_tau(astar)
309 phi <- o$phi
310 varphi <- o$varphi
311 tau.r <- varphi * phi
312 phi.alpha <- rep(astar, rank)
313 phi.a0 <- sum(phi.alpha)
314 a.vphi <- phi.a0
315
316 # update rank specific params
317 for (r in 1:rank) {
318     for (j in 1:D) {
319         tens.mu.r <- getmean(x.train.nona, beta,
                             rank, r)
320         betj <- getouter_list(lapply(beta[-j],
                             function(x) x[r, ]))
321         H <- matrix(NA, N, P[j]) # H
322         H_w <- matrix(NA, N, P[j])
323
324         for (i in 1:N) {
325             if (D == 2) {
326                 H[i, ] <- apply(x.train.
                             nona[i, , ], j,
                             function(x) {

```



```

327         return (sum(x *
328                     betj))
329     H_w[i, ] <- H[i, ] * sqrt(
330         w[i])
331     } else if (D == 3) {
332         H[i, ] <- apply(x.train[,
333             nona[i, , , ], j,
334             function(x) {
335                 return (sum(x *
336                     betj))
337             })
338         H_w[i, ] <- H[i, ] * sqrt(
339             w[i])
340     }
341 }
342 HH_w <- crossprod(H_w, H_w) # HH_w: H*
343                               Omega*H
344
345 # posterior covariance
346 K <- chol2inv(chol(HH_w + diag(1 / omega[[
347     j]][r, ]) / tau.r[r]))
348
349 # mm: y tilda
350 mm <- kappa / w - pred.mean - tens.mu.r
351
352 # posterior mean
353 bet.mu.jr <- K %%% crossprod(H, mm * w)
354
355 # update beta_jr
356 beta[[j]][r, ] <- bet.mu.jr + chol(K) %%%
357     rnorm(P[j])
358
359 ## update lambda.jr
360 lambda[r, j] <- rgamma(1, a.lam[r] + P[j],
361     b.lam[r] + sum(abs(beta[[j]][r, ])) /
362     sqrt(tau.r[r]))
363
364 ## update omega.jr
365 omega[[j]][r, ] <- sapply(1:P[j], function

```

```

      (kk) rgig(1, 1 / 2, beta[[j]][r, kk]^2
        / tau.r[r], lambda[r, j]^2))
356      # omega[r,j,] <- sapply(1:p, function(kk){
        a <- lambda[r, j]^2; b <- beta[[r]][kk, j]
        j^2 / tau.r[r]; map <- besselK(sqrt(a*b)
        ), 0.5 + 1) / besselK(sqrt(a*b), 0.5) *
        sqrt(b / a); return(map)})
357    }
358  }
359
360  tens.mean <- getmean(x.train.nona, beta, rank)
361  if (!is.null(Z)) {
362    pred.mean <- Zt %*% gam
363  }
364
365  cat(sprintf("sweep: %s\n", sweep))
366
367  ## store params
368  beta.store[[sweep]] <- beta
369  if (!is.null(Z)) {
370    gam.store[sweep, ] <- gam
371  } else {
372    gam.store[sweep] <- gam
373  }
374  alpha.store[sweep] <- astar ## not intercept
375  phi.store[sweep, ] <- phi
376  varphi.store[sweep, ] <- varphi
377  omega.store[[sweep]] <- omega
378  lambda.store[sweep, , ] <- lambda
379  sapply(1:rank, function(rr) hyppar.store[sweep, rr, ] <-
    c(a.lam[rr], b.lam[rr]))
380 }
381
382
383 tt <- abs(tt - Sys.time())
384 cat("Time out: ", tt, "\n")
385
386 ##### plotting omitted#####
387
388 ##### finalize #####

```

```

389     out <- list (
390       nsweep = nsweep ,
391       rank = rank ,
392       mu_store = mu_store ,
393       P = P ,
394       D = D ,
395       w.store = w.store ,
396       my = my, sy = sy, mz = mz, sz = sz, mx = mx, sx = sx, gam.store =
           gam.store , beta.store = beta.store , time = tt
397     )
398
399     class(out) <- "tensor.reg"
400     return(out)
401 }
402
403
404 ##### aux functions #####
405
406 getouter_list <- function(bet) {
407   D <- length(bet)
408   if (D == 1) {
409     return(bet[[1]])
410   }
411   if (D == 2) {
412     return(outer(bet[[1]], bet[[2]]))
413   } else {
414     return(outer(getouter_list(bet[1:(D - 1)]), bet[[D]]))
415   }
416 }
417
418 TP.rankR <- function(X.allr) {
419   R <- ncol(X.allr[[1]])
420   if (is.null(R)) {
421     return(getouter_list(X.allr))
422   } else {
423     Y <- array(0, dim = c(as.numeric(lapply(X.allr, function(x)
           length(x[, 1])))))
424     for (r in c(1:R)) {
425       Y <- Y + getouter_list(lapply(X.allr, function(x)
           x[, r]))

```

```

426         }
427         return(Y)
428     }
429 }
430
431 getmean <- function(X, beta, rank, rank.exclude = NULL) {
432     idx <- setdiff(1:rank, rank.exclude)
433     B <- Reduce("+", lapply(idx, function(r) getouter_list(lapply(beta
434         , function(x) x[r, ]))))
435     mu.B <- apply(X, 1, function(xx, bb) sum(xx * bb), bb = B)
436     return(mu.B)
437 }
438
439 tensor.mean <- function(x, n) {
440     Reduce("+", x) / n
441 }
442
443 logsum <- function(lx) {
444     return(max(lx) + log(sum(exp(lx - max(lx)))))
445 }
446
447 getBeta_mcmc <- function(beta.store) {
448     nsweep <- length(beta.store)
449     D <- length(beta.store[[1]])
450     rank <- nrow(beta.store[[1]][[1]])
451     P <- sapply(1:D, function(x) ncol(beta.store[[1]][[x]]))
452     Beta_mcmc <- array(dim = c(nsweep, prod(P)))
453     for (i in 1:nsweep) {
454         coef <- rep(0, prod(P))
455         for (r in 1:rank) {
456             coef <- coef + c(getouter_list(lapply(beta.store[[
457                 i]], function(x) x[r, ])))
458         }
459         Beta_mcmc[i, ] <- coef
460     }
461     quantile(Beta_mcmc)
462     return(Beta_mcmc)
463 }
464
465 add <- function(x) Reduce("+", x)

```

```

464
465 uncollapse <- function(str, collapse = "", mode = "character") {
466     a <- unlist(strsplit(str, collapse))
467     mode(a) <- mode
468     return(a)
469 }
470
471 rmse <- function(y, yhat) {
472     return(sqrt(mean((y - yhat)^2, na.rm = T)))
473 }
474
475 logsum <- function(lx) {
476     return(max(lx) + log(sum(exp(lx - max(lx)))))
477 }
478 # ==== 构造 rank-R PARAFAC 张量 ====
479 generate_parafac_tensor <- function(p, R, prob = 0.2, size = 2) {
480     A_list <- list()
481     B_list <- list()
482
483     for (r in 1:R) {
484         a_r <- rbinom(p[1], size = size, prob = prob)
485         b_r <- rbinom(p[2], size = size, prob = prob)
486         A_list[[r]] <- a_r
487         B_list[[r]] <- b_r
488     }
489
490     tensor <- matrix(0, nrow = p[1], ncol = p[2])
491     for (r in 1:R) {
492         tensor <- tensor + outer(A_list[[r]], B_list[[r]])
493     }
494
495     # 标准化为最大值为1
496     tensor <- tensor / max(tensor)
497     return(tensor)
498 }
499
500 load_images <- function(image_dir, target_size = c(48, 48)) {
501     images <- list()
502     labels <- c()
503

```

```

504     for (label in c("yes", "no")) {
505         folder_path <- file.path(image_dir, label)
506         label_value <- ifelse(label == "yes", 1, 0)
507
508         image_files <- list.files(folder_path, full.names = TRUE)
509
510         for (file in image_files) {
511             img <- load.image(file)
512
513             # 如果是RGB彩色图, 则转为灰度图
514             if (spectrum(img) == 3) {
515                 img <- grayscale(img)
516             }
517
518             img_resized <- resize(img, target_size[1], target_
                    size[2])
519             img_matrix <- as.array(img_resized)
520
521             # 转为 (height, width, 1) 格式
522             img_array <- array(img_matrix[, , 1], dim = c(
                    target_size[2], target_size[1], 1))
523
524             images[[length(images) + 1]] <- img_array
525             labels <- c(labels, label_value)
526         }
527     }
528
529     # 去掉通道维度, 转为 (N, 48, 48)
530     images_tensor <- abind(images, along = 0)[, , , 1]
531
532     return(list(images = images_tensor, labels = labels))
533 }
534
535 ##### sample on simulated data #####
536 set.seed(123)
537 # # ==== Scenario 1 ====
538 # Beta_tens <- generate_parafac_tensor(p = c(48, 48), R = 3)
539
540 # ==== Scenario 2 ====
541 a1 <- rep(0, 48)

```

```

542 a1[38:48] <- 1
543 b1 <- rep(0, 48)
544 b1[5:12] <- 1
545
546 a2 <- rep(0, 48)
547 a2[38:48] <- 1
548 b2 <- rep(0, 48)
549 b2[33:43] <- 1
550
551 a3 <- rep(0, 48)
552 a3[5:12] <- 1
553 b3 <- rep(0, 48)
554 b3[c(5:12, 33:43)] <- 1
555
556 Beta_tens <- outer(a1, b1) + outer(a2, b2) + outer(a3, b3)
557
558 # # ==== Scenario 3 ====
559 # Beta_tens <- matrix(0, 48, 48)
560 # for (i in 15:40) {
561 #     for (j in 10:35) {
562 #         Beta_tens[i, j] <- 1
563 #     }
564 # }
565
566 # # ==== Scenario 4 ====
567 # Beta_tens <- matrix(0, 48, 48)
568
569 # # 圆的中心和半径
570 # center_x <- 18
571 # center_y <- 18
572 # radius <- sqrt(0.10 * 48 * 48 / pi)
573
574 # # 填充圆形区域为 1
575 # for (i in 1:48) {
576 #     for (j in 1:48) {
577 #         # 计算 (i, j) 到圆心的距离
578 #         if ((i - center_x)^2 + (j - center_y)^2 <=
579 #             radius^2) {
580 #             Beta_tens[i, j] <- 1
581 #         }
582 #     }
583 # }

```

```

581         #     }
582     # }
583
584 # 保存 Beta_tens 图像
585 png("Beta_tens_.png", width = 600, height = 600)
586 image.plot(Beta_tens,
587 col = gray.colors(25, start = 1, end = 0), axes = FALSE,
588 main = "True Tensor Coefficient"
589 )
590 mtext(text = seq(10, 50, 10), side = 2, line = 0.3, at = seq(10, 50, 10) /
591       48, las = 1, cex = 0.8)
592 mtext(text = seq(10, 50, 10), side = 1, line = 0.3, at = seq(10, 50, 10) /
593       48, las = 2, cex = 0.8)
594 dev.off()
595
596 # Generate 2D images and binary response
597 N <- 500
598 p <- c(48, 48)
599 rank <- 3
600 X <- array(rnorm(N * prod(p)), dim = c(N, p)) # simulated image
601 Y <- sapply(1:N, function(x) sum(X[x, , ] * Beta_tens, na.rm = T))
602 hist(Y)
603
604 # # Lasso Loss
605 # p <- 1 / (1 + exp(-Y))
606 # Ylabel <- rbinom(n = N, size = 1, prob = p)
607
608 # SVM Loss
609 Ylabel <- rep(0, N)
610 Ylabel[Y >= 0] <- 1
611 Ylabel[Y < 0] <- 0
612
613 # 按照 7:3 的比例划分训练集和测试集
614 train_index <- sort(sample(1:N, 0.7 * N, replace = FALSE))
615 x.train <- X[train_index, , ]
616 y.train <- Ylabel[train_index]
617 x.test <- X[-train_index, , ]
618 y.test <- Ylabel[-train_index]
619
620 # ==== 脑肿瘤数据 ====

```



```

619 # # 设置路径
620 # train_dir <- "E:/BRAIN_TUMOR/BRAIN_TUMOR/train"
621 # val_dir <- "E:/BRAIN_TUMOR/BRAIN_TUMOR/val"
622
623 # # 加载数据
624 # train_data <- load_images(train_dir)
625 # val_data <- load_images(val_dir)
626
627 # # 合并数据
628 # all_images <- abind(train_data$images, val_data$images, along = 1)
629 # all_labels <- c(train_data$labels, val_data$labels) ''
630
631 # N <- dim(all_images)[1] # 总样本数
632 # train_index <- sort(sample(1:N, 0.7 * N, replace = FALSE))
633
634 # x.train <- all_images[train_index, , ]
635 # y.train <- all_labels[train_index]
636 # x.test <- all_images[-train_index, , ]
637 # y.test <- all_labels[-train_index]
638 # # 归一化
639 # x.train <- x.train / 255
640 # x.test <- x.test / 255
641
642 # train_index <- sort(sample(1:N, 0.7 * N, replace = FALSE))
643 # x.train <- X[train_index, , ]
644 # y.train <- Ylabel[train_index]
645
646 burnin <- 1000
647 nsweep <- 3000
648 sim <- PGtensor(Z = NULL, x.train, y.train, n = rep(1, length(y.train)),
649               nsweep = nsweep, rank = 3, scale = T)
650
651 # tensor_est: estimated tensor coefficient
652 tensor <- getBeta_mcmc(sim$beta.store)
653 tensor_est <- apply(tensor[burnin:nsweep, ], 2, mean) * sim$sy / sim$sx
654 rmse_val <- rmse(c(Beta_tens), c(tensor_est))
655 cor_val <- cor(c(Beta_tens), c(tensor_est))
656 cat(sprintf("估计的张量系数 RMSE(均方根误差为: %.6f\n", rmse_val))
657 cat(sprintf("张量系数估计与真实值的相关系数为: %.6f\n", cor_val))
658 # plot estimated tensor coefficient

```

```

658 png("Tensor_estimated_Scenario4_LR.png", width = 600, height = 600)
659 # plot estimated tensor coefficient
660 image.plot(tensor_est, col = gray.colors(25, start = 1, end = 0), axes = F
        )
661 mtext(text = seq(10, 50, 10), side = 2, line = 0.3, at = seq(10, 50, 10) /
        48, las = 1, cex = 0.8)
662 mtext(text = seq(10, 50, 10), side = 1, line = 0.3, at = seq(10, 50, 10) /
        48, las = 2, cex = 0.8)
663 dev.off()
664
665 residual_map <- Beta_tens - matrix(tensor_est, nrow = 48)
666 png("Residual_Tensor_Scenario4_LR.png", width = 600, height = 600)
667 image.plot(residual_map,
668 col = topo.colors(25), axes = FALSE,
669 main = "Residual Map: Beta_tens - tensor_est"
670 )
671 dev.off()
672 # get misclassification rate
673 x.test <- X[-train_index, , ]
674 y.test <- Ylabel[-train_index]
675 post.tens.mean.test <- array(0, N - length(train_index))
676 for (j in 1:(N - length(train_index))) {
677     post.tens.mean.test[j] <- c(tensor_est) %*% c(x.test[j, , ])
678 }
679 post.mui.test <- matrix(post.tens.mean.test, ncol = 1)
680 clust.test <- rep(0, N - length(train_index))
681 clust.test[post.mui.test > 0] <- 1
682 clust.test[post.mui.test <= 0] <- 0
683 missclassrate <- 1 - sum(clust.test == y.test) / length(clust.test)
684 cat(sprintf("测试集误分类率为: %.4f\n", missclassrate))
685 # get fl score
686 TP <- 0
687 FP <- 0
688 FN <- 0
689 for (j in 1:length(clust.test)) {
690     if (clust.test[j] == 1 && y.test[j] == 1) {
691         TP <- TP + 1
692     } else if (clust.test[j] == 1 && y.test[j] == 0) {
693         FP <- FP + 1
694     } else if (clust.test[j] == 0 && y.test[j] == 1) {

```

```
695         FN <- FN + 1
696     }
697 }
698 f1score <- TP / (TP + (FP + FN) / 2)
699 cat(sprintf("F1-score 为 : %.4f\n", f1score))
```