TULIP: A Toolbox for Linear Discriminant Analysis with Penalties

by Yuqing Pan, Qing Mai and Xin Zhang

Abstract Linear discriminant analysis (LDA) is a powerful tool in building classifiers with easy computation and interpretation. Recent advancements in science technology have led to the popularity of datasets with high dimensions, high orders and complicated structure. Such datasetes motivate the generalization of LDA in various research directions. The `R` package `TULIP` integrates several popular high-dimensional LDA-based methods and provides a comprehensive and user-friendly toolbox for linear, semi-parametric and tensor-variate classification. Functions are included for model fitting, cross validation and prediction. In addition, motivated by datasets with diverse sources of predictors, we further include functions for covariate adjustment. Our package is carefully tailored for low storage and high computation efficiency. Moreover, our package is the first `R` package for many of these methods, providing great convenience to researchers in this area.

Introduction

Linear discriminant analysis (LDA) is one of the most popular classification method and a cornerstone for multivariate statistics (**?**, e.g). Classical LDA builds a linear classifier based on p-dimensional multivariate predictor $\mathbf{X} \in \mathbb{R}^p$ to distinguish K classes and to predict the class label $Y \in \{1, \ldots, K\}$. Despite its simplicity, LDA is shown to be very accurate on many benchmark datasets (**???**). Moreover, LDA is easily interpretable and is thus often used as a visualizing tool for exploratory data analysis.

In recent decades, the advancements in science and technology have enabled researchers to collect datasets with increasing sizes and complexity. Such datasets pose challenges to LDA. Four challenges that we tackle with this package are as follows. First, in research areas such as biology, genomics and psychology, we often have more predictors than samples. However, LDA is not applicable on these high-dimensional data, because sample covariance matrix becomes not invertible when the number of predictors exceeds the sample size.

Secondly, when we have a large number of predictors, variable selection is often desired such that we can obtain a sparse classifier involving only a small proportion of the variables. On one hand, **??** showed in theory that variable selection is critical for accurate classification. On the other hand, sparse classifiers much easier to interpret in practice. However, LDA generally does not perform variable selection.

Thirdly, contemporary datasets often have complicated structure that renders the linear classifier in LDA inadequate. For example, in the presence of thousands of predictors, it may be inappropriate them to model all of them with the normal distribution. Moreover, research in neuroimaging, computational biology and personalized recommendation produces data in the form of matrices (2-way tensor) or tensors. The analysis of tensor datasets requires considerable modification to the vector-based LDA model.

Last but not least, integrative analysis with multiple data sources are drawing researchers' attention recently. Co-existence of diverse data types, such as vector, matrix and tensor calls for more sophisticated models to integrate the information from them to improve classification accuracy. It is critical to model the dependence among different types of data to reduce the noise level in the data and improve prediction accuracy (**?**).

Motivated by these challenges, many methods have been proposed to generalize LDA to datasets with high dimensions, non-normality and/or higher order predictors. In this package, we implement six methods that generalize LDA to contemporary complicated datasets. All of them are developed under models closely related to the LDA model, and penalties are imposed to achieve classification accuracy and variable selection in high dimensions. These methods include:

1. Direct sparse discriminant analysis (DSDA): DSDA generalizes the classical LDA model to high dimensions when there are only two classes (**?**). It formulates high-dimensional LDA into a penalized least squares problem.
2. Regularized optimal affine discriminant (ROAD): under the same model as DSDA, ROAD fits a sparse classifier by minimizing the classification error under the ℓ_1 constraint (**?**).
3. Sparse optimal scoring (SOS) for binary problems: SOS is also developed under the LDA model (**?**). It penalizes the optimal scoring problem (**?**). We focus on its application in binary problems.
4. Semiparametric sparse discriminant analysis (SeSDA): SeSDA assumes a semiparametric model where data transformation can be applied to alleviate the non-normality. In practice, SeSDA

	DSDA	SOS	ROAD	SeSDA	MSDA	CATCH
Classes	Binary	Binary	Binary	Binary	Multi-class	Multi-class
Data type	Vector	Vector	Vector	Vector	Vector	Tensor
Model	LDA	LDA	LDA	SeLDA	LDA	TDA/CATCH
Covariate adjustment	Yes	No	No	No	Yes	Yes

Table 1: Comparison of model settings between models. SOS was originally proposed to deal with both binary and multiclass problems, but we focus on binary problems in the package. Model SeLDA stands for Semi-parametric linear discriminant analysis, which is introduced in Section 2.2.4. Model TDA/CATCH represents tensor discriminant analysis and covariate-adjusted tensor in high-dimensions, which are illustrated in Section 2.2.5 and 2.3.6.

finds the data-driven transformation and then performs model-fitting on the transformed data (**?**).

5. Multiclass sparse discriminant analysis (MSDA): Instead of focusing on binary problems, MSDA considers the multiclass LDA model (**?**). It takes note of the fact that the Bayes' rule can be estimated with minimizing a quadratic loss. To account for the multiclass structure, a group lasso penalty (**?**) is applied to achieve variable selection.
6. Covariate-adjusted tensor classification in high-dimensions (CATCH): CATCH (**?**) is developed for tensor predictors. It takes advantage of the tensor structure to significantly reduce the number of parameters and hence alleviate computation complexity.

See Table 1 for a comparison of these methods. Despite their different model assumptions and formulas, all of them have strong theoretical support and excellent empirical performance. We further note that they can be combined with covariate adjustment when multiple data sources are available. Our package `TULIP` integrates diverse discriminant analysis models and supportive functions to make it a convenient and well-equipped toolbox. It has several notable advantages. First, we not only include functions for model fitting, but also cross validation functions for easy control of the sparsity level, and prediction functions for the prediction of future observations. In addition, we provide covariate adjustment functions that efficiently remove heterogeneity in the predictors and combine information from covariates. Second, our package greatly facilitates the application of DSDA, ROAD and SeSDA for R users, as they do not have public R packages on CRAN outside ours. Third, although MSDA and SOS have been implemented in packages `msda` and `sparseLDA`, we carefully modify their algorithms in our implementation to lower storage cost and/or speed up computation.

We acknowledge that many other efforts have been spent on topics closely related to that of our paper. On one hand, by now a large number of high-dimensional discriminant analysis methods have been developed. Some excellent examples include **???????????**. On the other hand, in the literature, many works study matrix/tensor regression and classification methods. Many of them impose low rank assumption (**????????**). All these methods have been reported to have great performance, but a comprehensive study of them is apparently out of the scope of our current paper.

The rest of this paper is organized as follows. We start with a brief overview of discriminant analysis models in Section 2.2. Model estimation and implementation details are discussed in Section 2.3. Section 2.4 contains instructions and examples on the usage of the package. A real data example is given in Section 2.5 to confirm the numerical performance of methods in the package.

Discriminant Analysis Models and Bayes Rules

Bayes rule for classification

Recall that $Y \in \{1, \ldots, K\}$ is the categorical response (class indicator), and we use the generic $\mathcal{X}$ to denote the predictor and (potential) additional covariate. Specifically, $\mathcal{X} = \mathbf{X} \in \mathbb{R}^p$ in classical multivariate discriminant analysis; $\mathcal{X} = \mathbf{X} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ in tensor discriminant analysis; and $\mathcal{X} = (\mathbf{X}, \mathbf{U})$ in covariate-adjusted classification settings, where $\mathbf{U} \in \mathbb{R}^q$ is additional covariates and $\mathbf{X}$ can be either vector or tensor. Our goal is to construct the optimal classifier to distinguish and predict Y based on $\mathcal{X}$ under various settings. Denote $\pi_k = \Pr(Y = k)$ and f_k as the conditional distribution of $\mathcal{X}$ within Class k (e.g. $f_k(\mathcal{X}) = f(\mathbf{X}, \mathbf{U} \mid Y = k)$ is the joint distribution of $\mathbf{X}$ and $\mathbf{U}$ given $Y = k$, in presence of $\mathbf{U}$). The optimal classifier, often referred to as the Bayes rule, is thus

$$\delta(\mathbf{X}) = \arg\max_k \{\log \pi_k + \log f_k(\mathcal{X})\}. \tag{1}$$

The Bayes rule achieves the lowest classification error possible (**?**). Therefore, it is our ultimate goal to estimate the Bayes rule. However, additional model assumptions are often needed for f_k to ensure statistical and computational efficiency. Consider the classical LDA setting of $\mathbf{X} \in \mathbb{R}^p$ and $Y \in \{1, \ldots, K\}$. To gain intuition, we often assume that within each class, the predictor follows a normal distribution with different means and a common covariance matrix. Then the Bayes rule is a linear function of $\mathbf{X}$ and can be straightforwardly estimated.

In the rest of this section, we discuss various statistical models that have been widely studied in the literature, along with the Bayes rules under these assumptions. Specifically, we review the classical LDA model, the semiparametric LDA model, and the tensor discriminant analysis model. We also discuss a general framework for covariate adjustment.

The Linear Discriminant Analysis Model (LDA)

Given a multivariate predictor $\mathbf{X} \in \mathbb{R}^p$ and $Y \in \{1, \ldots, K\}$, the LDA model assumes that $\mathbf{X}$ is normally distributed within each class, i.e,

$$\mathbf{X} \mid (Y = k) \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}), \quad \Pr(Y = k) = \pi_k, \quad k = 1, \ldots, K, \tag{2}$$

where $\boldsymbol{\mu}_k \in \mathbb{R}^p$ is the mean of $\mathbf{X}$ within class k, and $\boldsymbol{\Sigma} \in \mathbb{R}^{p \times p}$ is the common within class covariance matrix.

Define $\boldsymbol{\beta}_k = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_1)$ for $k = 1, \cdots, K$. The Bayes' rule turns out to be a linear function:

$$\widehat{Y} = \arg\max_k \Pr(Y = k \mid \mathbf{X}) = \arg\max_{k=1,\ldots,K} \{\log \pi_k + \boldsymbol{\beta}_k^T (\mathbf{X} - \boldsymbol{\mu}_k/2)\}. \tag{3}$$

The LDA model is simple yet elegant. All the parameters in this model have natural interpretations, while the Bayes rule has a nice linear form. An interesting fact about the Bayes rule in (3) is that it does not explicitly involve the p^2-dimensional parameter $\boldsymbol{\Sigma}^{-1}$. Instead, $\boldsymbol{\Sigma}^{-1}$ is only implicitly included in the discriminant directions $\boldsymbol{\beta}_k$. Moreover, it can be shown that the Bayes rule is equivalent to first reducing data to $\mathbf{X}^T\boldsymbol{\beta}_2, \ldots \mathbf{X}^T\boldsymbol{\beta}_K$ and then fitting the LDA model on the $(K-1)$-dimensional space. Therefore, to estimate the Bayes rule in high dimensions, our interest centers on the estimation of $\boldsymbol{\beta}_k$. We assume that $\boldsymbol{\beta}_k$'s are sparse with many elements being zero. Enforcement of this sparsity assumption will facilitate our estimation and naturally lead to variable selection.

Although the Bayes rule is derived under the somewhat restrictive normality and equal covariance assumptions, the discriminant directions $\boldsymbol{\beta}_k$ are still meaningful when data are non-normal, thanks to their geometric properties. It can be shown that, if we project $\mathbf{X}$ to $\boldsymbol{\beta}_k, k = 1, \ldots, K$, the separation between classes is maximized over all possible sets of $K-1$ linear projections. Consequently, the LDA model is reasonably resistant to model misspecification. However, in some of the cases where the LDA model assumptions are severely violated, one can resort to more flexible models. For example, the quadratic discriminant analysis model (**????**) relaxes the equal covariance assumption, while severely non-normal data can be modeled by the semiparametric model to be discussed in Section 2.2.4.

Covariates Adjustment

In many real-life problems, we have additional covariates along with the predictors. The covariates play two roles in the classification: it has predictive power on it own, and it also accounts part of the variation in the predictors. For example, in genomics studies, we record not only gene expression levels but also age and clinical measurements. In this case, we may view the gene expression levels as the high-dimensional predictor, and the age and clinical measurements as the covariates. We consider an LDA-type model to incorporate the covariates. In addition to the response Y and the predictor $\mathbf{X}$, we denote the covariates as $\mathbf{U} \in \mathbb{R}^q$. We assume that

$$\mathbf{U} \mid (Y = k) \sim N(\boldsymbol{\phi}_k, \boldsymbol{\Psi}), \tag{4}$$

$$\mathbf{X} \mid (\mathbf{U} = \mathbf{u}, Y = k) \sim N(\boldsymbol{\mu}_k + \boldsymbol{\alpha}\mathbf{u}, \boldsymbol{\Sigma}), \tag{5}$$

where $\boldsymbol{\phi}_k \in \mathbb{R}^q$ is the within-class mean, $\boldsymbol{\Psi} \in \mathbb{R}^{q \times q}$, $\boldsymbol{\Psi} > 0$ is the common within class covariance matrix of covariates, and $\boldsymbol{\alpha} \in \mathbb{R}^{p \times q}$ is the dependence of $\mathbf{X}$ on $\mathbf{U}$. We refer to this model as the covariate-adjusted LDA (CA-LDA) model. The CA-LDA model is conceptually similar to the CATCH model (**?**) for tensor, which is to be introduced in Section 2.2.5, but the CA-LDA model focuses on vector predictor $\mathbf{X}$ rather than tensor predictor.

Obviously, the CA-LDA model reduces to the LDA model in the absence of covariates. With the covariates, the CA-LDA model continues to have natural interpretations. Equation (4) indicates that $(\mathbf{U}, Y)$ marginally follow the LDA model. Equation (5) implies that the distribution of $\mathbf{X}$ not only

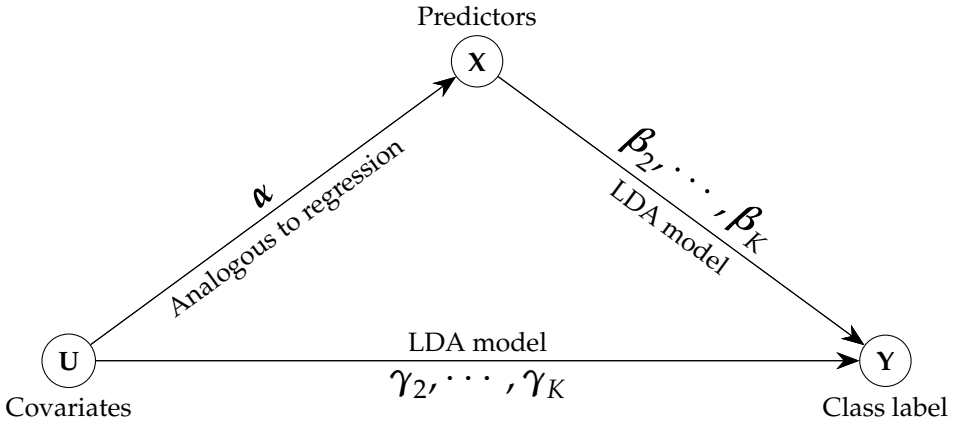

Figure 1: Graphical illustration of the direct and indirect effects. The direct effect of covariate $\mathbf{U}$ on Y follows classical discriminant analysis model measured by $\{\gamma_2, \ldots, \gamma_K\}$. Meanwhile, $\mathbf{U}$ also affects class label through affecting $\mathbf{X}$. Therefore we have $\widehat{Y} = f(\mathbf{X}, \mathbf{U})$.

depends on Y, but also $\mathbf{U}$ through mean dependence. Therefore, within each class, $\mathbf{X}$ is linked to $\mathbf{U}$ through a linear regression model, while, after we adjust for $\mathbf{U}$, $(\mathbf{X}, Y)$ follow the LDA model as well. See Figure 1 for a graphical illustration of the relationship among $\mathbf{X}$, $\mathbf{U}$ and Y.

Under the CA-LDA model, the Bayes' rule is

$$\widehat{Y} = \arg\max_{k=1,\ldots K} \left\{ a_k + \gamma_k^T \mathbf{U} + \boldsymbol{\beta}_k^T (\mathbf{X} - \boldsymbol{\alpha}\mathbf{U}) \right\} \tag{6}$$

where $\gamma_k = \boldsymbol{\Psi}^{-1}(\boldsymbol{\phi}_k - \boldsymbol{\phi}_1)$, $\boldsymbol{\beta}_k = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_1)$ and $a_k = \log(\pi_k/\pi_1) - \frac{1}{2}\gamma_k^T(\boldsymbol{\phi}_k + \boldsymbol{\phi}_1) - \frac{1}{2}\boldsymbol{\beta}_k^T(\boldsymbol{\mu}_k + \boldsymbol{\mu}_1))$ is a scalar that does not involve $\mathbf{X}$ or $\mathbf{U}$. Throughout this paper, we assume that $\mathbf{U}$ is low-dimensional and does not need variable selection, but $\mathbf{X}$ is high-dimensional. In the presence of covariates, $\mathbf{X}$ needs to be first adjusted to $\mathbf{X} - \boldsymbol{\alpha}\mathbf{U}$ before entering the Bayes rule. Similar to the LDA model, we assume that the coefficient of $\mathbf{X} - \boldsymbol{\alpha}\mathbf{U}$, $\boldsymbol{\beta}_k$, is sparse.

The Semiparametric LDA model

Although LDA is reasonably resistant to model misspecification, we may still need more flexible models when data are heavily non-normal. The semiparametric linear discriminant analysis (SeLDA) model (**?**) is proposed for this purpose. SeLDA assumes that there exists a set of strictly monotone univariate transformations $h_1, \ldots, h_p$ such that

$$(h_1(X_1), \cdots, h_p(X_p)) \mid (Y = k) \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}). \tag{7}$$

For identifiability, we further assume that all the diagonal elements in $\boldsymbol{\Sigma}$ are 1, and all elements in $\boldsymbol{\mu}_1$ are 0. We also use the shorthand notation $h(\mathbf{X}) = (h_1(X_1), \cdots, h_p(X_p))$. The transformation h is assumed to be unknown and needs to be estimated from data. The SeLDA model assumes that the LDA model is true up to an unknown transformation. It has the same spirit as the well-known Box-Cox transformation, with which model assumptions are relaxed by proper data mapping.

It is easy to see that the LDA model is a special case of the SeLDA model, if we restrict $h(\mathbf{X}) = \mathbf{X}$. However, in the SeLDA model, we do not impose any parametric assumptions on h, which leads to great flexibility in practice. We further review a formula for h_j that will facilitate its estimation. It can be shown that

$$h_j = \Phi^{-1} \circ F_{1j} = \Phi^{-1} \circ F_{kj} + \mu_{kj}, \tag{8}$$

where Φ is the cumulative distribution function (CDF) of the standard normal random variable, and F_{kj} is the CDF of X_j within Class k. Equation (8) will be used in Section 2.3.4. The SeLDA model also amounts to assuming that the data follow the Gaussian copula model within each class (**???**).

Although the SeLDA model requires much weaker conditions than the LDA model, it preserves many of the desirable properties. One of them is that the Bayes rule continues to be a linear function of the transformed data $h(\mathbf{X})$:

$$\widehat{Y} = \arg\max_{k=1,\ldots,K} \{\log \pi_k + \boldsymbol{\beta}_k^T (h(\mathbf{X}) - \boldsymbol{\mu}_k/2)\}. \tag{9}$$

Consequently, just as in the LDA model, when the dimension is high, we assume that $\boldsymbol{\beta}_k$ is sparse to allow accurate estimation.

Tensor Discriminant Analysis (TDA) and Covariate Adjustment

The tensor discriminant analysis (TDA) model is proposed for classification based on tensor predictors. We first briefly introduce some standard tensor notation (**?**). See Appendenx A for more rigorous definitions. An *M-way tensor* is denoted by a multidimensional array $\mathbf{A} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ where $M \geq 2$, $p_1, \ldots, p_M$ are all positive integers. We often need to multiply an M-way tensor $\mathbf{C}$ by M matrices along each mode $\mathbf{G}_i, i = 1, \ldots, M$, denoted by $[\![\mathbf{C}; \mathbf{G}_1, \ldots, \mathbf{G}_M]\!]$. For example, in Figure 2 we obtain $\mathbf{A} = [\![\mathbf{C}; \mathbf{G}_1, \ldots, \mathbf{G}_3]\!]$ by multiplying a 3-way tensor $\mathbf{C}$ with matrices $\mathbf{G}_i$ along each mode. If $\mathbf{G}_i, i \neq m$ are identity matrices and $\mathbf{G}_m$ is a vector, then we write $\mathbf{C} \bar{\times}_m \mathbf{G}_m = [\![\mathbf{C}; \mathbf{I}, \ldots, \mathbf{G}_m, \ldots, \mathbf{I}]\!]$.

Further, we say a tensor $\mathbf{X} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ follows the tensor normal distribution $TN(\boldsymbol{\mu}, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M)$ if it can be written as

$$\mathbf{X} = \boldsymbol{\mu} + [\![\mathbf{Z}; \boldsymbol{\Sigma}_1^{1/2}, \ldots, \boldsymbol{\Sigma}_M^{1/2}]\!],$$

where $\mathbf{Z} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ has elements all independently standard normal, $\boldsymbol{\mu} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ is the mean tensor, and $\boldsymbol{\Sigma}_m \in \mathbb{R}^{p_m \times p_m}$ are covariance matrices. See Figure 3 for an illustration.

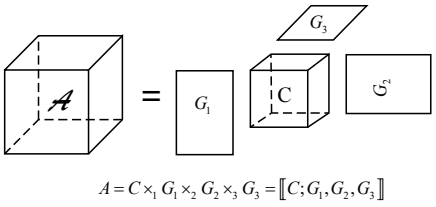

Figure 2: Tucker decomposition of tensor $\mathbf{A}$.

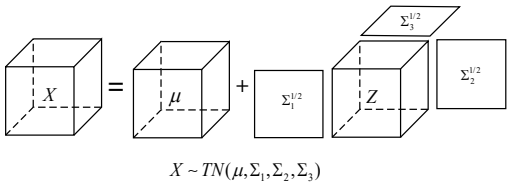

Figure 3: Tensor normal distribution

Now we discuss the tensor discriminant analysis (TDA) model. Consider the M-way tensor predictor $\mathbf{X} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ where $M \geq 2$ and class label $Y \in \{1, \ldots, K\}$. The TDA model assumes that

$$\mathbf{X} \mid (Y = k) \sim TN(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M), \quad \Pr(Y = k) = \pi_k \tag{10}$$

where $\boldsymbol{\mu}_k \in \mathbb{R}^{p_1 \times \cdots p_M}$, $\boldsymbol{\Sigma}_m \in \mathbb{R}^{p_m \times p_m}$ is the within-class mean, $\boldsymbol{\Sigma}_m > 0$ is the common within-class covariance matrix along the m-th mode of the tensor, and $0 < \pi_k < 1$ is the prior probability for Class k. Compared to the LDA model, TDA utilizes the tensor normal distribution to model $\mathbf{X}$ within each class. By taking advantage of the tensor structure, TDA drastically reduces the number of unknown parameters (**?**). It can be seen that the TDA model requires $O(\sum_{m=1}^M p_m^2)$ parameters to model the dependence among $\mathbf{X}$. However, if we ignore the tensor structure and assume the LDA model on the vectorized version of $\mathbf{X}$, the covariance matrix has $O(\prod_{m=1}^M p_m^2)$ parameters.

Under the TDA model, the Bayes' rule is

$$\widehat{Y} = \arg \max_{k=1,\ldots K} \{a_k + \langle \mathbf{B}_k, \mathbf{X} \rangle\} \tag{11}$$

where $\mathbf{B}_k = [\![\boldsymbol{\mu}_k - \boldsymbol{\mu}_1; \boldsymbol{\Sigma}_1^{-1}, \ldots, \boldsymbol{\Sigma}_M^{-1}]\!]$, and $a_k = \log(\pi_k / \pi_1) - \langle \mathbf{B}_k, \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_1) \rangle$ is a scalar that does not involve $\mathbf{X}$. It can be seen that the Bayes rule is again a linear function in $\mathbf{X}$, with the linear coefficients $\mathbf{B}_k$. In high dimensions, we again impose the sparsity assumption by assuming that many elements in $\mathbf{B}_k$ are zeros.

Similar to the vector case, when additional covariates are provided, the TDA model can be combined with covariate adjustment. **?** proposed the CATCH model for this purpose. In addition to $(Y, \mathbf{X})$, we are given the covariates $\mathbf{U} \in \mathbb{R}^q$. The CATCH model assumes that

$$\mathbf{U} \mid (Y = k) \sim N(\boldsymbol{\phi}_k, \boldsymbol{\Psi}), \tag{12}$$

$$\mathbf{X} \mid (\mathbf{U} = \mathbf{u}, Y = k) \sim TN(\boldsymbol{\mu}_k + \boldsymbol{\alpha} \bar{\times}_{(M+1)} \mathbf{u}, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M). \tag{13}$$

where $\boldsymbol{\phi}_k \in \mathbb{R}^q$ is the within-class mean of $\mathbf{U}$, $\boldsymbol{\Psi} \in \mathbb{R}^{q \times q}$ is the within-class covariance of $\mathbf{U}$, and $\boldsymbol{\alpha} \in \mathbb{R}^{p_1 \times \cdots \times p_M \times q}$ characterizes the dependence of $\mathbf{X}$ on $\mathbf{U}$. The parameters in the CATCH model can be interpreted in the same way as the CA-LDA model in Section 2.2.3.

The Bayes' rule under the CATCH model is

$$\widehat{Y} = \arg\max_{k=1,\ldots K} \left\{ a_k + \boldsymbol{\gamma}_k^T \mathbf{U} + \langle \mathbf{B}_k, \mathbf{X} - \boldsymbol{\alpha} \bar{\times}_{(M+1)} \mathbf{U} \rangle \right\}, \tag{14}$$

where $\boldsymbol{\gamma}_k = \boldsymbol{\Psi}^{-1}(\boldsymbol{\phi}_k - \boldsymbol{\phi}_1)$, and $a_k = \log(\pi_k/\pi_1) - \frac{1}{2}\boldsymbol{\gamma}_k^T(\boldsymbol{\phi}_k + \boldsymbol{\phi}_1) - \langle \mathbf{B}_k, \frac{1}{2}(\boldsymbol{\mu}_k + \boldsymbol{\mu}_1)\rangle$ is a scalar that does not involve $\mathbf{X}$ or $\mathbf{U}$. Similar to the TDA model, we assume that $\mathbf{B}_k$ is sparse in high dimensions, but impose no further sparsity assumptions on other parameters.

Methods

In this section, we formally introduce the six methods implemented by the package: DSDA, ROAD, SOS, SeSDA, MSDA and CATCH. Throughout the rest of this paper, we denote $\widehat{\boldsymbol{\Sigma}}$ as the pooled sample covariance, $\widehat{\boldsymbol{\mu}}_k$ as the within-class sample mean, n as the sample size, and n_k as the sample size in class k. All the methods involve a tuning parameter $\lambda > 0$ that controls the amount of sparsity. Hence, when we refer to an estimate $\widehat{\boldsymbol{\beta}}$, it should be understood as $\widehat{\boldsymbol{\beta}}(\lambda)$, although we suppress λ in most estimates for presentation convenience. We will discuss the tuning parameter in detail in Section 2.3.8.

Direct sparse discriminant analysis (DSDA)

The direct sparse discriminant analysis (DSDA) is proposed for binary classification under the LDA model in (2). Recall that our main interest is in estimating the coefficients $\boldsymbol{\beta}_k$ in the Bayes rule (3). Because DSDA assumes that there are only two classes, it suffices to estimate $\boldsymbol{\beta} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$. In high dimensions, we assume that $\boldsymbol{\beta}$ is sparse. Let $y_i = -\frac{n_1}{n}$ if $Y_i = 1$ and $y_i = \frac{n}{n_2}$ if $Y_i = 2$. DSDA first solves the penalized least squares problem

$$(\widehat{\boldsymbol{\beta}}^{\text{DSDA}}, \widehat{\beta}_0^{\text{DSDA}}) = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^p, \beta_0\in\mathbb{R}} \left\{ n^{-1} \sum_{i=1}^{n} (y_i - \beta_0 - \mathbf{X}_i^T\boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}, \tag{15}$$

where $\lambda > 0$ is the tuning parameter, $\sum_{j=1}^{p} |\beta_j|$ is the LASSO penalty (**?**), and $\widehat{\boldsymbol{\beta}}^{\text{DSDA}}$ is our estimate for $\boldsymbol{\beta}$. Because of the LASSO penalty, $\widehat{\boldsymbol{\beta}}^{\text{DSDA}}$ is typically sparse. To estimate the Bayes rule, we further estimate the LDA model on the reduced data $\{Y_i, \mathbf{X}_i^T\widehat{\boldsymbol{\beta}}^{\text{DSDA}}\}_{i=1}^n$.

Numerical and theoretical studies show that DSDA consistently estimate the Bayes rule under mild conditions. Also, DSDA can be computed very efficiently, as (15) is a heavily-studied ℓ_1 penalized least squares problem. Our implementation utilizes `glmnet` to solve (15).

Regularized optimal affine discriminant (ROAD)

Regularized optimal affine discriminnat (ROAD, **?**) is another binary penalized discriminant analysis method for high-dimensional data. ROAD estimates $\boldsymbol{\beta}$ by

$$\widehat{\boldsymbol{\beta}}^{\text{ROAD}} = \arg\min \boldsymbol{\beta}^T\widehat{\boldsymbol{\Sigma}}\boldsymbol{\beta} \tag{16}$$

$$\|\boldsymbol{\beta}\|_1 \le c, \boldsymbol{\beta}^T(\widehat{\boldsymbol{\mu}}_2 - \widehat{\boldsymbol{\mu}}_1)/2 = 1. \tag{17}$$

We remark that **?** independently proposed the ℓ_1-Fisher's discriminant analysis method that closely resembles ROAD, but the developments of ROAD and the ℓ_1-Fisher's discriminant analysis have different emphasis. ROAD clarifies several theoretical aspects of high-dimensional classification, while ℓ_1-Fisher's discriminant analysis is developed for simultaneous testing for gene pathways. For simplicity, we focus on ROAD in what follows.

In its optimization, the constraint of ℓ_1-norm can be recast as a ℓ_1-penalty with parameter λ. ROAD rewrites (16) as

$$\widehat{\boldsymbol{\beta}}^{\text{ROAD}} = \arg\min_{\boldsymbol{\beta}^T(\widehat{\boldsymbol{\mu}}_2 - \widehat{\boldsymbol{\mu}}_1)/2=1} \boldsymbol{\beta}^T\widehat{\boldsymbol{\Sigma}}\boldsymbol{\beta} + \lambda\|\boldsymbol{\beta}\|_1 \tag{18}$$

The authors of ROAD proposed to solve (18) by replacing the nonconvex constraint with a quadratic penalty. However, we adopt a different approach to solve (18). It is showed in **?** that the solution paths of DSDA and ROAD are equivalent. In other words, for any $\lambda > 0$, there exists $\tilde{\lambda} > 0$ such that $\widehat{\boldsymbol{\beta}}^{\text{DSDA}}(\lambda) \propto \widehat{\boldsymbol{\beta}}^{\text{ROAD}}(\tilde{\lambda})$. Because DSDA produces a solution path much faster than the original proposal of ROAD, we solve ROAD by first finding the solution path of DSDA for a range of λ, and then find each corresponding $\tilde{\lambda}$ to recover the solution path of ROAD.

Sparse optimal scoring (SOS) in binary problems

We also implement the successful discriminant analysis method, sparse optimal scoring (SOS, **?**). We focus on binary problems, where we are able to greatly improve the computation speed. For multiclass problems, SOS can be solved by the R package **sparseLDA**.

In binary problems, SOS creates a dummy variable $\mathbf{Y}^{dm} \in \mathbb{R}^{n\times 2}$ as a surrogate for the categorical response Y, where $Y_{ik}^{dm} = 1\{Y_i = k\}$. Then SOS estimates coefficient by solving

$$\begin{aligned}\widehat{\boldsymbol{\beta}}^{\text{SOS}} &= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^2,\boldsymbol{\beta}\in\mathbb{R}^p}\{\|\mathbf{Y}^{dm}\boldsymbol{\theta}-\widetilde{\mathbf{X}}\boldsymbol{\beta}\|^2+\lambda\|\boldsymbol{\beta}\|_1\},\\ &\quad \text{s.t } \tfrac{1}{n}\boldsymbol{\theta}^T\mathbf{Y}^{dm^T}\mathbf{Y}^{dm}\boldsymbol{\theta}=1, \boldsymbol{\theta}^T\mathbf{Y}^{dm^T}\mathbf{Y}^{dm}1=0,\end{aligned} \tag{19}$$

where $\widetilde{\mathbf{X}}$ is the centered $\mathbf{X}$, and $\boldsymbol{\theta}\in\mathbb{R}^2$ is the score for the two classes. SOS is a popular penalized discriminant analysis method because of its impressive empirical performance. It can be solved by iteratively minimizing the objective function in (19) over $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$.

However, we take another approach to solve SOS with lower computation cost. **?** showed that $\widehat{\boldsymbol{\beta}}^{\text{SOS}}$ is closely related to the DSDA estimator defined in (15). Let $\hat{\pi}_y = \frac{n_y}{n}$. We have that

$$\widehat{\boldsymbol{\beta}}^{\text{SOS}}(\lambda) = \sqrt{\hat{\pi}_1\hat{\pi}_2}\widehat{\boldsymbol{\beta}}^{\text{DSDA}}(\frac{\lambda}{\sqrt{\hat{\pi}_1\hat{\pi}_2}}). \tag{20}$$

Therefore, to solve for $\widehat{\boldsymbol{\beta}}^{\text{SOS}}(\lambda)$, we first find $\widehat{\boldsymbol{\beta}}^{\text{DSDA}}(\frac{\lambda}{\sqrt{\hat{\pi}_1\hat{\pi}_2}})$ with DSDA, and rescale it to obtain the SOS solution. This approach avoids iteration between $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$, and is often faster than the original algorithm for SOS.

Semiparametric sparse discriminant analysis (SeSDA)

SeSDA (**?**) fits the SeLDA model in (7) for binary problems. It is expected to have better performance than DSDA when data are heavily non-normal. SeSDA has two steps. First, we find an estimate $\widehat{h}$ for the unknown function h. Second, we apply DSDA on the pseudo data $(\widehat{h}(\mathbf{X}), Y)$. In what follows, we focus on the estimation of h.

Two estimators have been proposed for h based on (8), the naive estimator and the pooled estimator. Without loss of generality, we assume that Class 1 has more observations than Class 2. Denote $\tilde{F}_{1j}$ as the empirical CDF of X_j within Class 1. To avoid infinity values at tails, we further Winsorize $\tilde{F}_{1j}$ to $\hat{F}_{1j}$, where

$$\hat{F}_{1j}(x) = \begin{cases} 1-1/n_1^2 & \text{if } \tilde{F}_{1j}(x) > 1-1/n_1^2\\ \tilde{F}_{1j}(x) & \text{if } 1/n_1^2 \le \tilde{F}_{1j}(x) \le 1-1/n_1^2\\ 1/n_1^2 & \text{if } \tilde{F}_{1j}(x) < 1/n_1^2.\end{cases}$$

The naive estimator is shown to consistently estimate h, but in practice it is vulnerable to loss of efficiency, as it only utilizes one class of data. Therefore, the pooled estimator is proposed as a more efficient estimator.

Similar to $\hat{F}_{1j}$, we denote $\hat{F}_{2j}$ as the empirical CDF of X_j within Class 2 Winsorized at $(1/n_2^2, 1-1/n_2^2)$. We first find an estimate for $\boldsymbol{\mu}_{2j}$ as $\widehat{\boldsymbol{\mu}}_{2j}^{(\text{pool})} = \hat{\pi}_1\hat{\mu}_{2j}^{(1)} + \hat{\pi}_2\hat{\mu}_{2j}^{(2)}$, where $\hat{\mu}_{2j}^{(1)} = \frac{1}{n_2}\sum_{Y_i=2}\Phi^{-1}\circ\hat{F}_{1j}(X_{ij})$, $\hat{\mu}_{2j}^{(2)} = -\frac{1}{n_1}\sum_{Y_i=1}\Phi^{-1}\circ\hat{F}_{2j}(X_{ij})$. Then the pooled estimator for h_j is

$$\widehat{h}_j^{(\text{pool})} = \hat{\pi}_1\hat{h}_j^{(1)} + \hat{\pi}_2\hat{h}_j^{(2)}, \tag{21}$$

where $\hat{h}_j^{(1)} = \Phi^{-1}\circ\widehat{F}_{1j}$ and $\hat{h}_j^{(2)} = \Phi^{-1}\circ\widehat{F}_{2j} + \hat{\mu}_{2j}^{(\text{pool})}$. The pooled estimator is usually more accurate than the naive estimator because it utilizes both classes to form an estimate for h_j.

Multiclass sparse discriminant analysis (MSDA)

Up to now, we have focused on binary classifiers. In this section, we discuss a multiclass classifier under the LDA model (2). Assume that $K \ge 2$. By the Bayes rule (3), we need to estimate the coefficients $\boldsymbol{\beta}_k = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_k - \boldsymbol{\mu}_1), k = 2,\ldots,K$. There is no need to estimate $\boldsymbol{\beta}_1$, as it is zero by definition. As in the binary problems, we continue to assume that the classifier is sparse in high dimensions, in the

Algorithm 1 Algorithm for MSDA

1. Compute $\widehat{\mathbf{\Sigma}}$ and $\widehat{\delta}^k = (\widehat{\mu}_k - \widehat{\mu}_1), k = 1, 2, \cdots, K$.
2. Initialize $\widehat{\boldsymbol{\beta}}_k^{(0)}$ and compute $\widetilde{\boldsymbol{\beta}}_k^{(0)}$ by $\widetilde{\boldsymbol{\beta}}_{k,j} = \frac{\hat{\delta}_j^k - \sum_{l \neq j} \hat{\sigma}_{lj} \hat{\beta}_{kl}}{\hat{\sigma}_{jj}}$.
3. For steps $w = 1, 2, \ldots$, do the following until convergence:
   for each element $j = 1, \ldots, p$,
   (a) Compute
   $$\widehat{\boldsymbol{\beta}}_{\cdot j}^{(w)} = \widetilde{\boldsymbol{\beta}}_{\cdot j}^{(w-1)} (1 - \frac{\lambda}{\| \widetilde{\boldsymbol{\beta}}_{\cdot j}^{(w-1)} \|})_+; \tag{24}$$
   (b) Update
   $$\tilde{\beta}_{kj} = \frac{\hat{\delta}_j^k - \sum_{l \neq j} \hat{\sigma}_{lj} \hat{\beta}_{kl}^{(w)}}{\hat{\sigma}_{jj}}. \tag{25}$$
4. At convergence, output $\boldsymbol{\beta}_k$.

sense that only a few predictors are relevant to classification. However, this sparsity assumption has slightly different implication in multiclass problems. Note that, for any X_j, if any one of $\beta_{2j}, \ldots, \beta_{Kj}$ is nonzero, X_j is important for classification, as it helps with distinguishing between at least one pair of classes. Therefore, in order for an X_j to be unimportant, we have to have $\beta_{2j} = \ldots = \beta_{Kj} = 0$. In other words, the coefficients $\boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_K$ has a group sparsity structure.

The multi-class sparse discriminant analysis (MSDA) has been proposed for fitting a sparse classifier under the context of interest. It takes note of the fact that, on the population level, we have

$$(\boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_K) = \arg \min_{\boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_K} \sum_{k=2}^{K} \{ \frac{1}{2} \boldsymbol{\beta}_k^T \mathbf{\Sigma} \boldsymbol{\beta}_k - (\boldsymbol{\mu}_k - \boldsymbol{\mu}_1)^T \boldsymbol{\beta}_k \}. \tag{22}$$

Therefore, in high dimensions, MSDA replaces the parameters with the sample estimates and impose the group sparsity structure through group lasso (**?**). More specifically, MSDA estimates $\boldsymbol{\beta}$ by

$$(\widehat{\boldsymbol{\beta}}_2, \cdots, \widehat{\boldsymbol{\beta}}_K) = \arg \min_{\boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_K} \sum_{k=2}^{K} \{ \frac{1}{2} \boldsymbol{\beta}_k^T \widehat{\mathbf{\Sigma}} \boldsymbol{\beta}_k - (\widehat{\boldsymbol{\mu}}_k - \widehat{\boldsymbol{\mu}}_1)^T \boldsymbol{\beta}_k \} + \lambda \sum_{j=1}^{P} \| \boldsymbol{\beta}_{\cdot j} \| . \tag{23}$$

The problem in (23) can be solved by a blockwise coordinate descent algorithm (**?**) summarized in Algorithm 1. We refer to Algorithm 1 as the original MSDA algorithm. The `R` package `msda` implements such an algorithm. However, the original MSDA algorithm can be demanding on storage for high-dimensional data, because it requires the input of $\widehat{\mathbf{\Sigma}} \in \mathbb{R}^{p \times p}$. When p is very large, the original MSDA algorithm can be practically inapplicable. Moreover, because of the sparse nature of $\boldsymbol{\beta}$, many elements in $\widehat{\mathbf{\Sigma}}$ are never used, and the calculation of them leads to unnecessary computation burden.

Therefore, in our implementation we modify the original MSDA algorithm for lower storage and computation cost for high-dimensional data. Note that $\widehat{\mathbf{\Sigma}}$ is only used in updating rule (25). We take advantage of two properties of this updating rule (25). First, given the natural element-wise property of coordinate descent algorithm, only the j-th column of covariance matrix $\widehat{\mathbf{\Sigma}}_{\cdot j}$ is needed in each iteration. The full covariance matrix is never used during the computation process. Therefore, it is not necessary to store the huge covariance matrix. Secondly, a large number elements of $\widehat{\boldsymbol{\beta}}$ are exactly 0. Hence among the column $\widehat{\mathbf{\Sigma}}_{\cdot j}$, we only need to compute the rows corresponding to the nonzero coefficients. These facts motivate us to develop the modified MSDA algorithm. The modified MSDA algorithm is largely identical to the original algorithm, but with two important distinctions. On one hand, in Step 1 we only require the input of $\widehat{\delta}^k$ but not $\widehat{\mathbf{\Sigma}}$. On the other hand, Step 3(b) in (25) is replaced with

$$\tilde{\beta}_{kj} = \frac{(n-K)\hat{\beta}_j^k - \sum_{l \neq j} \hat{\beta}_{kl}^{(m)} (\sum_{k=1}^{K} [\sum_{i \in \mathbf{T}_k} (X_{il} - \mu_{kl})(X_{ij} - \mu_{kj})])}{\sum_{k=1}^{K} [\sum_{i \in \mathbf{T}_k} (X_{ij} - \mu_{kj})^2]}, \tag{26}$$

where $\mathbf{T}_k = \{i : y_i = k\}$. By doing so, we avoid the storage and the computation of the full matrix of $\widehat{\mathbf{\Sigma}}$.

In computing (26), we further use three tricks to speed up the computation. Firstly, we calculate and store all diagonal elements in the covariance matrix as they will be called multiple times. Secondly, we keep the indexes of nonzero elements in $\mathbf{T}_k$ and update it every time we observe a new nonzero element. Hence we do not need to check all elements to locate the nonzero ones in each iteration. Thirdly, we update equation (26) by only computing elements corresponding to the nonzero indexes in $\mathbf{T}_k$. With these three tricks, the modified algorithm reduces the space complexity from $O(p^2)$ to $O(p)$, and is also faster than the original algorithm for large p.

Covariate-adjusted tensor classification in high dimensions (CATCH)

When $\mathbf{X}$ is a tensor instead of a vector, we need to fit the TDA model or the CATCH model (in presence of covariates) for better efficiency and accuracy. **?** proposed the CATCH method to fit both models, but in this section we focus on the CATCH method on the TDA model, where there is no covariate. The inclusion of covariates will be discussed in Section 2.3.7.

Recall that, under the TDA model, we aim to estimate the parameters $\mathbf{B}_k = [\![\boldsymbol{\mu}_k - \boldsymbol{\mu}_1; \boldsymbol{\Sigma}_1^{-1}, \ldots, \boldsymbol{\Sigma}_M^{-1}]\!]$. We first rewrite $\mathbf{B}_k$ as solutions to estimating equations:

$$(\mathbf{B}_2, \ldots, \mathbf{B}_K) = \arg\min_{\mathbf{B}_2,\ldots,\mathbf{B}_K} \sum_{k=2}^{K} \left(\langle \mathbf{B}_k, [\![\mathbf{B}_k; \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M]\!] \rangle - 2\langle \mathbf{B}_k, \boldsymbol{\mu}_k - \boldsymbol{\mu}_1 \rangle \right),$$

where for two M-way tensors $\mathbf{A}, \mathbf{C}$, $\langle \mathbf{A}, \mathbf{C} \rangle = \sum_{j_1 \cdots j_M} a_{j_1 \cdots j_M} c_{j_1 \cdots j_M}$ is the inner product of two tensors. To estimate $\mathbf{B}_k$, we find the within-class sample mean $\widehat{\boldsymbol{\mu}}_k$ as the estimate for $\boldsymbol{\mu}_k$, and moment-based unbiased estimators $\widehat{\boldsymbol{\Sigma}}_m$ for $\boldsymbol{\Sigma}_m$; see the formulas in Appendix C. We further add the group LASSO penalty for variable selection. Therefore, CATCH solves the following problem:

$$\min_{\mathbf{B}_2,\ldots,\mathbf{B}_K} \left[\sum_{k=2}^{K} \left(\langle \mathbf{B}_k, [\![\mathbf{B}_k; \widehat{\boldsymbol{\Sigma}}_1, \ldots, \widehat{\boldsymbol{\Sigma}}_M]\!] \rangle - 2\langle \mathbf{B}_k, \widehat{\boldsymbol{\mu}}_k - \widehat{\boldsymbol{\mu}}_1 \rangle \right) + \lambda \sum_{j_1 \ldots j_M} \sqrt{\sum_{k=2}^{K} b_{k,j_1 \cdots j_M}^2} \right]. \tag{27}$$

CATCH can be solved by a coordinate descent algorithm with an explicit updating formula in each iteration.

Covariates adjustment

When we have additional covariates $\mathbf{U}$, the CA-LDA model or the CATCH model should be fitted. Whether $\mathbf{X}$ is a vector or a tensor, a key step for the covariate adjustment is the estimation of $\boldsymbol{\alpha}$, the dependence of $\mathbf{X}$ on $\mathbf{U}$. We use the maximum likelihood estimator (MLE). Denote $\overline{\mathbf{U}}_k$ as the sample mean of $\mathbf{U}$ within class k and $\overline{\mathbf{X}}_k$ as the sample mean of $\mathbf{X}$ within class k. Define group-wise centered data $\widetilde{\mathbf{X}}_i = \mathbf{X}_i - \overline{\mathbf{X}}_{Y_i}$, $\widetilde{\mathbf{U}}_i = \mathbf{U}_i - \overline{\mathbf{U}}_{Y_i}$.

For vector-variate $\mathbf{X}_i \in \mathbb{R}^p$, we adjust for covariate $\mathbf{U}$ by $\mathbf{X}_i - \widehat{\boldsymbol{\alpha}}\mathbf{U}_i$, where $\widehat{\boldsymbol{\alpha}} \in \mathbb{R}^{q \times p}$ is the MLE,

$$\widehat{\boldsymbol{\alpha}} = (\widetilde{\mathbf{U}}^T \widetilde{\mathbf{U}})^{-1} \widetilde{\mathbf{U}}^T \widetilde{\mathbf{X}}. \tag{28}$$

For tensor-variate $\mathbf{X}_i \in \mathbb{R}^{p_1 \times \cdots \times p_M}$, we let $\boldsymbol{\alpha}_{j_1 \cdots j_M} \in \mathbb{R}^q$ be the regression coefficient of univariate $X_{i,j_1 \cdots j_M}$ on multivariate $\mathbf{U}_i \in \mathbb{R}^q$. Then the MLE for $\boldsymbol{\alpha}_{j_1 \cdots j_M}$ is $\widehat{\boldsymbol{\alpha}}_{j_1 \cdots j_M} = (\widetilde{\mathbf{U}}^T \widetilde{\mathbf{U}})^{-1} \widetilde{\mathbf{U}}^T \widetilde{X}_{j_1 \cdots j_M}$, which can be expressed more explicitly as,

$$\widehat{\boldsymbol{\alpha}}_{j_1 \cdots j_M} = \left\{ \sum_{k=1}^{K} \sum_{Y_i = k} (\mathbf{U}_i - \overline{\mathbf{U}}_k)(\mathbf{U}_i - \overline{\mathbf{U}}_k)^T \right\}^{-1} \left\{ \sum_{k=1}^{K} \sum_{Y_i = k} (\mathbf{U}_i - \overline{\mathbf{U}}_k)(X_{i,j_1 \cdots j_M} - \overline{X}_{k,j_1 \cdots j_M}) \right\}. \tag{29}$$

Afterwards, the ensemble of all $\widehat{\boldsymbol{\alpha}}_{j_1 \cdots j_M}$, $\widehat{\boldsymbol{\alpha}}$, is our estimator for $\boldsymbol{\alpha}$. The covariate-adjusted predictor is then obtained as $\mathbf{X}_i - \widehat{\boldsymbol{\alpha}} \bar{\times}_{M+1} \mathbf{U}_i$.

Selection of the tuning parameter

We recommend selecting the tuning parameter in all methods by cross validation, which is implemented in our package as supportive functions for most of the methods. In cross validation, a sequence of potential tuning parameters is supplied. For each candidate tuning parameter λ, the dataset is random split into L folds. Then we fit L classifiers, each of which is fitted on $L-1$ folds of the data and validated on the remaining one fold. The average validation error rate of the L classifiers is used as a

	Parameters	λ	`dfmax`	`model` option	Covariate	Cross validation
DSDA	Binary vector	✓			✓	✓
ROAD	Binary vector	✓				
SOS	Binary vector	✓				
SeSDA	Binary vector	✓				✓
MSDA	Multi-class vector	✓	✓	✓	✓	✓
CATCH	Multi-class tensor	✓	✓		✓	✓

Table 2: Method description and major parameters. Penalty parameter λ controls the size of ℓ_1-penalty. Parameter `dfmax` limits the maximum number of non-zero variables. Parameter `model` specifies the version of implementation for MSDA.

measurement of the performance of the corresponding λ. The λ with the smallest average validation error is used in our final model fitting.

If desired, our package can automatically generate a sequence of tuning parameters for all the methods. They will first compute the smallest λ that shrinks all coefficients to zero; this value is taken as the upper bound of tuning range. Then the upper bound is multiplied by a small number to generate the lower bound. Finally, a sequence of tuning parameters is uniformly generated between the lower and the upper bound.

Using the R Package

The R package **TULIP** provides user-friendly functions to fit discriminant analysis model and perform predictions on vector and tensor data. The package can be downloaded through cran link or install in R through `install.packages('TULIP')`. In installing package, the pre-required packages **MASS** for LDA model fitting, packages **Matrix** (**?**) and **tensr** (**?**) for matrix and tensor operations, and the package **glmnet** for LASSO are also automatically installed. Users do not need to install them separately. To guarantee higher computation efficiency of the package, core algorithms of MSDA and CATCH are implemented in Fortran, which have already been compiled and can also be used directly.

Among all the six methods, there is always a tuning parameter λ to control the size of sparsity. On the implementation aspect, MSDA and CATCH also have parameter `dfmax` to limit the number of selected variables and will only return the solutions with number of non-zero elements less than `dfmax`. Furthermore, MSDA has a `model` option to specify version of implementation between multi.original and multi.modified. The methods are summarized in Table 2.

The functions in the package consists of two parts. One part contains core functions which generate solution paths of all the methods, including functions `dsda`, `road`, `sos`, `SeSDA`, `msda` and `catch`. Since binary classification can be regarded as a special case of multi-class problems, we also embedded DSDA into `msda` function. See Section 2.4.1 for details. The other part includes supportive functions to perform covariate adjustment, prediction, cross validation and handle some special cases.

To illustrate how to use the functions, we first simulate a binary vector data set named `dat.vec` with dimension $p = 500$ and sample size $n_k = 75$. In the data set, we have $\mathbf{X}_i \mid (Y_i = k) \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}_1 = 0$, $\boldsymbol{\mu}_2 = \boldsymbol{\Sigma}\boldsymbol{\beta}$, $\sigma_{ij} = 0.3$ if $i \neq j$ and $\sigma_{ii} = 1$, $\boldsymbol{\beta}_j = 0.5$ for $1 \leq j \leq 10$ and $\boldsymbol{\beta}_j = 0$ otherwise. We further generate a testing data set with sample size 1000 from the same distribution. Variables in `dat.vec` is summarized in Table 3. Data set `dat.vec` can be simulated by code

```
dat.vec<-sim.bi.vector(1000)
```

Variable	Type	Dimension
x	matrix	150×500
y	vector	150
testx	matrix	1000×500
testy	vector	1000

Table 3: Data set dat.vec.

Moreover, we include two real data sets, GDS1615 and colorimetric sensor array data set, in the package to demonstrate usage of the functions. Data set GDS1615 (**?**) is a vector data set where observations belong to three classes. The original data set contains 127 observations and 22283 variables. Package **msda** preprocessed the data by computing F-test statistics of each variable (**?**), whose definition is in appendix. Hence only 127 variables are kept in the data set. Colorimetric sesor

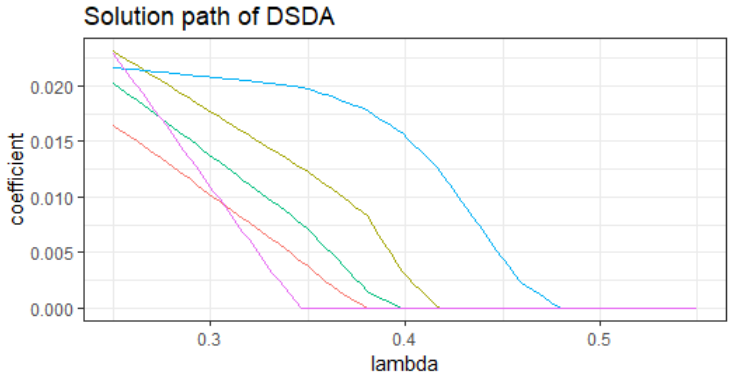

Figure 4: Solution path of five selected variables in a DSDA model.

array data (CSA) was used to show the performance of discriminant analysis method (**?**). It records information of chemical dyes after exposed to volatile chemical toxicants to identify their classes. It contains 147 observations in 21 classes. For each observation, the predictor is a 36×3 matrix. We include two conditions in our dataset, but focus on the Immediately Dangerous to Life or Health (IDLH) condition.

Core functions

Function `dsda`

The following code shows an example of utilizing DSDA. Given the data set `dat.vec`, we fit DSDA on $\{\mathbf{X}, Y\}$ by specifying the tuning range of parameter λ to be a sequence between $[0.005, 0.3]$. Hence the function will generate a solution path. Next, we apply `predict` function on the model and obtain the prediction for each λ and error rate. In the example, we report the minimum error rate and corresponding parameter value.

```
obj <- dsda(dat.vec$x, y=dat.vec$y, lambda=seq(0.005, 0.3, length.out=20))
pred <- predict(obj, dat.vec$testx)
err<- apply(pred, 2, function(x){mean(x!=dat.vec$testy)})
print(min(err))
[1] 0.111
print(obj$lambda[which.min(err)])
[1] 0.02052632
```

If one wishes, `dsda` can also be used in a more automatic way. On one hand, it can be called without supplying a sequence of value for tuning parameter. The function will automatically generate a sequence based on data. On the other hand, the prediction can be performed along with model fitting if testing data is supplied. The function `dsda` will produce the prediction error on the testing data corresponding to each tuning parameter. See the following example.

```
obj <- dsda(dat.vec$x, y=dat.vec$y,testx=dat.vec$testx)
err <- apply(obj$pred, 2, function(x){mean(x!=dat.vec$testy)})
print(min(err))
[1] 0.107
print(obj$lambda[which.min(err)])
[1] 0.03180946
```

Figure 4 shows a solution path of DSDA model. As parameter λ increases, more coefficients will be shrunken towards 0. In addition, DSDA can also integrate the covariate adjustment, model fitting and prediction. The usage is similar to the function `catch`, and we do not give a separate example here to avoid redundancy.

Function `SeSDA`

Function `SeSDA` fits a semiparametric sparse discriminant analysis model on the input vector data. The simulated data `dat.vec` follows normal distribution within each class. We take an exponential transformation on it to violate the normality assumption. The following example shows that `SeSDA` achieves error rates 11%. However, if we directly apply DSDA on the data set, the minimum error rate is as high as 15.8%. Therefore, the preprocessing of SeSDA can indeed help to improve performance under this scenario.

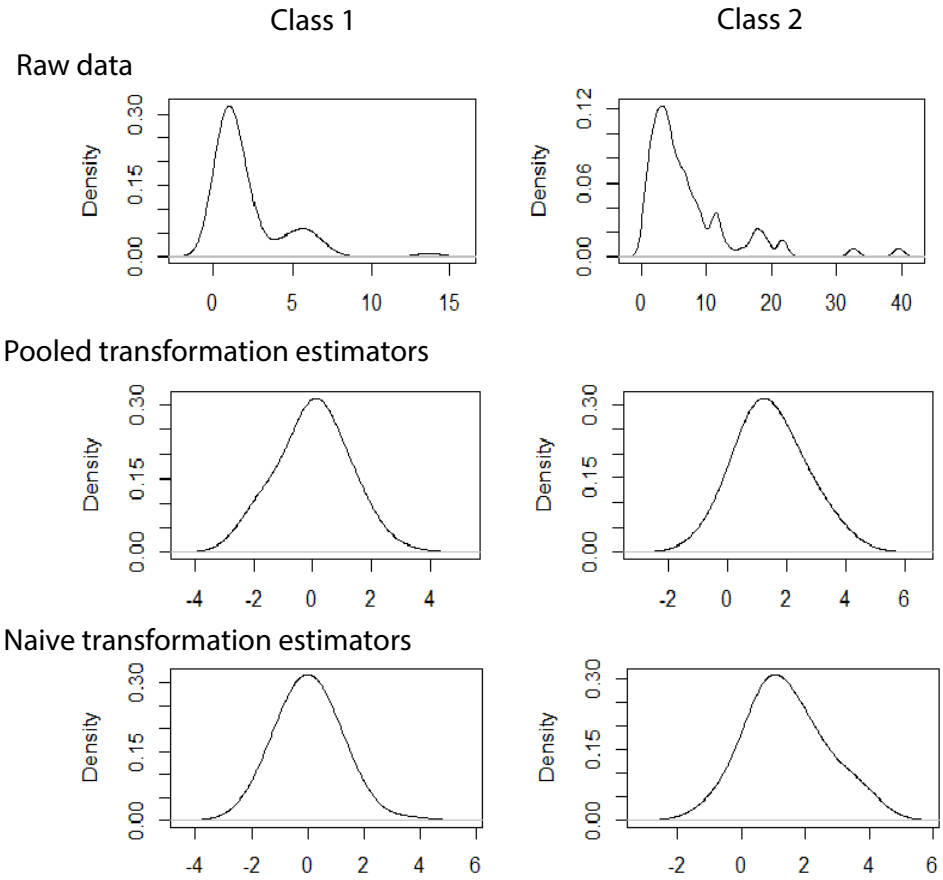

Figure 5: The distribution of the 1st variable in simulated data set among two classes before transformation and after transformation. The top row is before transformation. The second row is after pooled transformation. The bottom row is after naïve transformation.

```
x <- exp(dat.vec$x)
testx <- exp(dat.vec$testx)
obj.SeSDA <- SeSDA(x, y=dat.vec$y)
pred.SeSDA <- predict(obj.SeSDA, testx)
err <- apply(pred.SeSDA, 2, function(x){mean(x!=dat.vec$testy)})
min(err)
[1] 0.11
```

Further, Figure 5 shows how the distribution of the first variable changes after transformation. It is clear that both pooled and naïve transformatins result in approximately normal distribution.

Functions `ROAD` **and** `SOS`

Functions `ROAD` and `SOS` can generate equivalent solution paths as ROAD (**?**) and SOS (**?**) methods on binary vector data, respectively. Both of the two models are fit by calling `dsda` function. Compared to the original package for SOS, **sparseLDA**, our implementation is usually faster, especially when a solution path or parameter tuning is needed. For example, to fit a solution path with 10 possible values of λs on a toy example with $p = 40$, our implementation reduces the computation time by half compared to **sparseLDA**. An example of fitting ROAD and SOS model is as follows. The `lambdas` passed into `ROAD` and `SOS` functions will be directly used by `dsda` function. The `lambdas` returned by the two functions are their corresponding parameters in ROAD and SOS model, respectively. Figure 6 shows the relationship between the λ's that generate the same solution.

```
obj.dsda <- dsda(dat.vec$x, y=dat.vec$y, lambda=seq(0.1, 0.5, length.out=20))
```

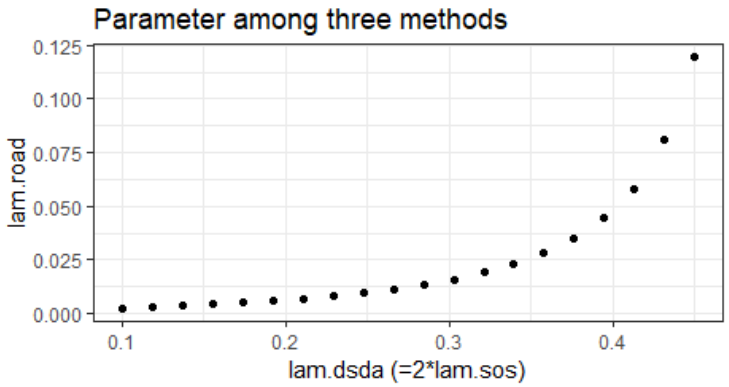

Figure 6: Parameters in ROAD vs. Parameters in DSDA. Notice that the parameters in DSDA are double those of SOS.

```
obj.road <- ROAD(dat.vec$x, y=dat.vec$y, lambda=seq(0.1, 0.5, length.out=20))
obj.sos <- SOS(dat.vec$x, y=dat.vec$y, lambda=seq(0.1, 0.5, length.out=20))
```

Function `msda`

The function `msda` provides an interface to fit MSDA. Similarly to `dsda`, without specification of possible values of λ, the function will automatically generate a sequence of λs. Function `msda` can also perform predictions when testing data is supplied and make adjustments on covariates when covariates exist. We apply `msda` on GDS1615 data set to as a demonstration. We report the minimum training error, its corresponding parameter value and the number of non-zero variables selected by the model.

```
data(GDS1615)
x <- GDS1615$x
y <- GDS1615$y
set.seed(123456)
teindex <- c(sample(which(y==1), sum(y==1)/3), sample (which(y==2),
   sum(y==2)/3), sample(which(y==3), sum(y==3)/3))
obj <- msda(x[-teindex, ], y=y[-teindex], testx=x[teindex, ])
err <- apply(obj$pred, 2, function(x){mean(x!=y[teindex])})
paste(min(err), obj$lambda[which.min(err)], obj$df[which.min(err)] )
[1] "0.04878049 1.446872 19"
```

If one wishes to visualize the discriminant effect, plots of projections on the discriminant coefficients is helpful. A principle component analysis is also optional to show the classification even more clearly. For illustration, we perform principle component analysis on $\mathbf{X}\boldsymbol{\beta}$ where $\boldsymbol{\beta} = \{\boldsymbol{\beta}_2, \boldsymbol{\beta}_3\}$ is the discriminant coefficient. The scatter plot on the two principle components is shown in Figure 7. It is clear to see that the three classes are separated well.

We also note that `msda` has an argument `model` that can be specified by users to use different algorithms in MSDA. The options for `model` include `binary`, `multi.original` and `multi.modified`. The option `binary` can only be used in binary problems. If selected, MSDA is solved by DSDA, which gives the same solution with usually less time. However, using this option in multi-class problems will result in an error. The option `multi.original` indicates that MSDA is solved by the original algorithm, which requires the calculation of the full covariance matrix. When the dimension is low, `multi.original` is often efficient. The option `multi.modified`, on the other hand, solves MSDA with the modified algorithm, where only part of the covariance matrix is calculated in each iteration. This option allows MSDA to be applicable in much higher dimensions. Also, when `multi.modified` is selected, we suggest using relatively larger tuning parameters to account for the high dimensionality. If unspecified, `model` is set to be `binary` in binary problems. If the response variable is multi-class, the function will call multi.original implementation for $p \leq 2000$ and multi.modified implementation for $p > 2000$.

Function `catch`

To illustrate usage of function `catch`, we first simulate a data set named `dat.ten` with tensor predictors $\mathbf{X}_i \in \mathbb{R}^{10\times10\times10}$ and covariates $\mathbf{U}_i \in \mathbb{R}^2$. The data is simulated from model $\mathbf{X}_i \mid (Y_i = k) \sim TN(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}_3)$ where $\boldsymbol{\mu}_1 = 0$, $\boldsymbol{\mu}_2 = [\![\boldsymbol{\beta}; \boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Sigma}_3]\!]$, $\boldsymbol{\Sigma}_j = \mathbf{I}$ for $j = 1, 2, 3$, $\boldsymbol{\beta}_{[1:2,1:2,1:2]} = 0.8$ and 0 otherwise. Let $\mathbf{U}_i \mid (Y_i = k) \sim N(\boldsymbol{\phi}_k, \boldsymbol{\psi})$ where $\boldsymbol{\phi}_1 = 0$, $\boldsymbol{\phi}_2 = (0.3, 0.3)$ and $\boldsymbol{\psi} = \mathbf{I}$. The connection

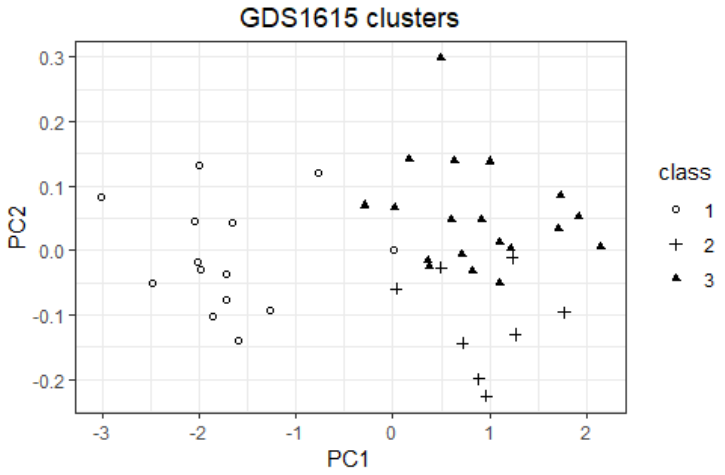

Figure 7: The GDS data projected onto the two principle components of $\mathbf{X}\boldsymbol{\beta}$.

Variable	Type	Dimension
x	list	150. Each element is a $10 \times 10 \times 10$ array.
y	vector	150
z	matrix	150×2
vec_x	matrix	1000×150
testx	list	1000. Each element is a $10 \times 10 \times 10$ array.
testy	vector	1000
testz	matrix	150×2
vec_testx	matrix	1000×1000

Table 4: Data set dat.ten.

between $\mathbf{X}$ and $\mathbf{U}$ is measured by $\boldsymbol{\alpha} \in \mathbb{R}^{10\times10\times10\times2}$ and $\boldsymbol{\alpha}_{[1:5,1:5,1:5,1]} = 1$ and 0 otherwise. Variables in `dat.ten` are summarized in Table 4.

Data set `dat.vec` can be simulated by code

```
dat.ten<-sim.tensor.cov(1000)
```

Function `catch` fits a CATCH model on the input tensor data. Covariates are optional for the function and the function will fit a TDA model when there is no covariate. Function `catch` has already integrated the adjustment step and model fitting step, hence it will automatically adjust for covariates when covariates exist. If one prefers to seperate the adjustment step, he/she can call `adjten` function to make adjustments and then supply the adjusted predictors into `catch`, which we will discuss in supportive functions.

Similar to the two functions above, function `catch` can generate a solution path on default or user specified potential values of the parameter. It will also perform prediction when testing data is specified. To make predictions on CATCH model, user can directly apply `catch` function or separating adjustment and model fitting step and then call the `predict` function. The following example shows how to fit the model and make prediction when covariates exist. As mentioned above, functions `dsda` and `msda` shares the same arguments name for covariates.

```
obj <- catch(dat.ten$x, dat.ten$z, dat.ten$y, dat.ten$testx, dat.ten$testz)
pred <- obj$pred
err <- apply(pred, 2, function(x){mean(x!=dat.ten$testy)})
min(err)
[1] 0.167
obj$lambda[which.min(err)]
[1] 0.4270712
```

An example of applying CATCH to fit model and perform prediction on CSA data is as follows. `catch` function takes list of multi-dimensional array as input. In the dataset, `x` is a list of length 148, where each element is a matrix of dimension 36×3; `y` is a vector whose value ranges between 1 and 21. We use default parameter sequence of length 100 and the prediction for each value of parameter is generated.

```
data(csa)
x <- csa$IDLH
y <- csa$y
teindex <- seq(1,147,7)
obj <- catch(x[-teindex, ], y=y[-teindex], testx=x[teindex, ], nlambda=10)
err <- apply(obj$pred, 2, function(x){mean(x!=y[teindex])})
print(err)
[1] 0.9523819 0.1904762 0.0952381 0.0000000 0.0952381 0.0000000 0.0000000
 0.0000000 0.0000000 0.0000000
```

Other functions

Two special cases

We provide two more functions for two common problems in practice, binary classification and matrix classification, respectively. First, the function `catch_matrix` fits CATCH model on matrix data (2-way tensor), which is a special case of `catch`. The usage of `catch_matrix` is exactly the same as that of `catch`, with the only exception that the predictor has to be a matrix instead of higher-order tensor.

Second, our package includes the function `dsda.all` that integrates cross validation, model fitting and prediction. It requires the input of the training set and testing set. Then the optimal tuning parameter is chosen by cross validation on the training set, and the corresponding testing error is reported. See the following example.

```
obj <- dsda.all(dat.vec$x, dat.vec$y, dat.vec$testx, dat.vec$testy, nfolds = 10)
print(obj$err)
[1] 0.116
```

Supportive functions

The package provides functions `cv.dsda`, `cv.msda`, `cv.SeSDA` and `cv.catch` to perform cross validation. For all of these functions, user can give a sequence of potential values to tune parameter. Otherwise, the function will first fit a model on the entire data set and then perform cross validation on the automatically generated λs from the entire data set. Similar as `msda`, user can specify which model to use in `cv.msda` or let the function determine by input data.

Users can also specify the number of folds by the argument `nfolds`. Another argument `lambda.opt` has two options `"min"` and `"max"`. When multiple λs lead to same error rate, `"min"` will return the smallest tuning parameter with the lowest error rate while `"max"` will return the largest one. We take `cv.dsda` and `cv.catch` as two examples.

```
obj.dsda <- cv.dsda(dat.vec$x, dat.vec$y, nfolds = 10)
obj.catch <- cv.catch(dat.ten$x, dat.ten$z, dat.ten$y, lambda.opt="min")
```

Function `adjten` and `adjvec` implement the adjustment step for tensor and vector data, respectively. It takes training tensor/vector, covariate and response as input, and outputs the adjusted tensor/vector and adjustment coefficients $\boldsymbol{\alpha}$. The adjustement step has already been incorporated into the modeling fitting functions `dsda`, `msda` and `catch`. When user input covariates along with tensor/vector, the model fitting functions will automatically make the adjustment. But if user do not want to use the automatic prediction in model fitting function and prefer to predict via `predict`, user need to first make the adjustment to obtain adjustment coefficient `gamma`, and pass it into function `predict`. Notice that making adjustment and fitting a model on the adjusted tensor and response without covariate is equivalent as fitting a model by inputting the original tensor, covariate and response labels. Examples of two approaches are given as follows.

```
obj <- catch(dat.ten$x, dat.ten$z, dat.ten$y, dat.ten$testx, dat.ten$testz)
obj.adj <- adjten(dat.ten$x, dat.ten$z, dat.ten$y, dat.ten$testx,
   dat.ten$testz)
obj.fit <- catch(dat.ten$x, dat.ten$z, dat.ten$y)
pred <- predict(obj.fit, obj.adj$testxres, dat.ten$z, dat.ten$testz,
   obj.adj$gamma)
```

There are three prediction functions corresponding to `dsda`, `msda` and `catch`, respectively. All of them can be directly called by `predict` and the function will recognize which function to use based on the input fitted model object. When covariate exists, user needs to pass the adjustment coefficient obtained from function `adjten`, the fitted model and testing data altogether to make predictions. Therefore, we encourage user to direct use model fitting functions `msda` and `catch` to fit model and predict categorical responses.

Method	Binary			Multi-class		
Error rate (%) / Time (seconds)	Mean	SE	Time	Mean	SE	Time
DSDA/multi.modified	23.58	0.23	36.30	35.85	0.24	88.8
SeSDA	23.68	0.24	731.28	NA	NA	NA
CATCH	22.79	0.24	78.6	35.22	0.25	101.4
ℓ_1-GLM	23.99	0.16	36.23	35.66	0.21	134.4
SOS	23.87	0.26	100.8	37.07	0.29	1114.8

Table 5: ADHD classification. Average error rates based on 100 replicates and running time of 20 replicates are reported.

Real Data Example

In this section, we will show the performance of the models by a real data set. We considered the attention deficit hyperactivity disorder (ADHD) data set. The dataset is available on NITRC (http://fcon_1000.projects.nitrc.org/indi/adhd200) (**?**). It contains three parts of information: s-MRI data which is a 3-D tensor, covariate information including age, gender and handedness which is a vector, and response label. Among all 930 individuals, there are four types of categorical labels: Typically Developing Childrem (TDC), ADHD Combined, ADHD Hyperactive and ADHD Inattentive.

We downsize the tensor to dimension $24 \times 27 \times 24$ and consider two classification scenarios. One is to combine ADHD Hyperactive with the ADHD Combined since there are only 13 subjects in class ADHD Hyperactive. This results in a multi-class problem with three classes. The second one is to further combine TDC and ADHD Inattentive since none of these two categories have hyperactivity symptoms. This give us a binary problem. We split the dataset into a training set and testing set by ratio 8 : 2.

Given the tensor structure and existence of covariates, the most suitable approach is to apply CATCH on that. We also vectorize the tensor into vectors and stack covariates along with the long vector to apply vector methods. For binary case, DSDA is applied. For multi-class case, MSDA model with `multi.modified` is applied since the dimension is too large to employ `multi.original`. We also compared with SOS (**?**) by its own package **sparseLDA** and ℓ_1-GLM (**?**) by package **glmnet**.

For each replicate, we perform cross validation on training data and record the classification error on testing data. The entire process was repeated for 100 times and we report the mean and standard error of the error rates. The performance is shown in Table 5.

Discussion

Package **TULIP** provides a toolbox to fit various sparse discriminant analysis models, including parametric models DSDA, ROAD, SOS for binary vector data, semiparametric model SeSDA for binary vector data, MSDA for multiclass vector data, and CATCH for multiclass tensor data. As a comprehensive toolbox, the package provides prediction and cross validation functions as well.

Meanwhile, the package propose an approach to handle cases when both predictor and covariates are supplied. The predictor can be vector and tensor, while the covariates are usually low-dimensional vectors. Covariates may have an effect on both response and the predictor. Therefore making adjustment and excluding the influence of covariates from predictor is important. The package includes functions to make the adjustments and can be called easily by supplying covariates in the model fitting function.

Appendices

Tensor Notation

On each dimension, which is named mode, a tensor is composed by vectors of length $(p_k \times 1)$ called mode-k fiber, defined as $A_{i_1 \cdots i_{k-1} I_k i_{k+1} \cdots i_M}$, $I_k = 1, \ldots p_k$. Stacking the mode-1 fiber by row gives the vectorization of a tensor $\text{vec}(\mathbf{A})$, which is a $(\prod_m p_m \times 1)$ column vector. If we unfold the tensor along the k-th mode, we obtain a matrix $\mathbf{A}_{(k)} \in \mathbb{R}^{p_k \times \prod_{l \neq k} p_l}$.

Denote the *mode-k product* of a tensor $\mathbf{A}$ and a matrix $\boldsymbol{\alpha} \in \mathbb{R}^{d \times p_k}$ by $\mathbf{A} \times_k \boldsymbol{\alpha} \in \mathbb{R}$, which results in a tensor of dimension $p_1 \times \cdots \times p_{k-1} \times d \times p_{k+1} \times \cdots \times p_M$. Each element of the product is the product of a mode-k fiber of $\mathbf{A}$ and a row vector of $\boldsymbol{\alpha}$. In particular, the *mode-k vector product* of a tensor $\mathbf{A}$ and a

vector $\mathbf{c} \in \mathbb{R}^{p_k}$ is a $(M-1)$-way tensor as a special case when $d = 1$. The *Tucker decomposition* of a tensor is defined as $\mathbf{A} = \mathbf{C} \times_1 \mathbf{G}_1 \times_2 \cdots \times_M \mathbf{G}_M$, in short of $[\![\mathbf{C}; \mathbf{G}_1, \ldots, \mathbf{G}_m]\!]$. In particular, the vectorization of tucker decomposition has the fact that $\text{vec}([\![\mathbf{C}; \mathbf{G}_1, \ldots, \mathbf{G}_M]\!]) = (\mathbf{G}_M \otimes \cdots \otimes \mathbf{G}_1)\,\text{vec}(\mathbf{C})$, where $\otimes$ denotes Kronecker product. If $\mathbf{X} = \boldsymbol{\mu} + [\![\mathbf{Z}; \boldsymbol{\Sigma}_1^{1/2}, \ldots, \boldsymbol{\Sigma}_M^{1/2}]\!]$, where $\mathbf{Z} \in \mathbb{R}^{p_1 \times \cdots \times p_M}$ and all elements of $\mathbf{Z}$ independently follow the univariate standard normal distribution, we say $\mathbf{X}$ follows a tensor normal distribution $\mathbf{X} \sim TN(\boldsymbol{\mu}, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_M)$. The dependence structure on the j-th mode is measured by $\boldsymbol{\Sigma}_j > 0$. Hence, $\text{vec}(\mathbf{X}) = \text{vec}(\boldsymbol{\mu}) + \boldsymbol{\Sigma}^{1/2}\text{vec}(\mathbf{Z})$, where $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_M \otimes \cdots \otimes \boldsymbol{\Sigma}_1$.

Simulation code

Data sets dat.vec and dat.ten are used to illustrate usage of the functions. Detailed model settings are described in Section 2.4. Here are the code to simulate the two data sets.

Code to simulate data set dat.vec:

```
set.seed(123456)
sigma <- matrix(0.3, 500, 500)
diag(sigma) <- 1
dsigma <- t(chol(sigma))
#define beta and mean
beta <- matrix(0, nrow = 500, ncol = 1)
beta[1:10,1] <- 0.5
M <- matrix(0, nrow = 2, ncol = 500)
M[2,] <- sigma%*%beta
y <- c(rep(1, 75), rep(2, 75))
#generate test data
telabel <- ceiling(runif(1000)*2)
x <- matrix(rnorm(150*500),ncol = 500)%*%t(dsigma)
x[y==2, ] <- x[y==2, ] + M[2,]
testx <- matrix(rnorm(1000*500), ncol = 500) %*% t(dsigma)
testx[telabel==2, ] <- testx[telabel==2, ] + M[2, ]
dat.vec <- list(x = x, y = y, testx = testx, testy = telabel)
```

Code to simulate data set dat.ten:

```
set.seed(123456)
sigma <- array(list(), 3) #define covariance matrices
dsigma <- array(list(), 3)
for (i in 1:3){
   sigma[[i]] <- diag(10)
   dsigma[[i]] <- t(chol(sigma[[i]]))
}
B2 <- array(0, dim=c(10,10,10)) #define B and mean
B2[1:2, 1:2, 1:2] <- 0.8
M <- array(list(), 2)
M[[1]] <- array(0, dim=c(10,10,10))
M[[2]] <- atrans(B2, sigma)
y <- c(rep(1,75), rep(2,75))
coef <- array(0, dim=c(10,10,10,2)) #define alpha
coef[1:5, 1:5, 1:5, 1] <- 1
telabel <- ceiling(runif(1000)*2)
z <- matrix(rnorm(2*150), nrow=150, ncol=2) #generate covariates
z[y==2,] <- z[y==2,] + 0.3
testz <- matrix(rnorm(2*1000), nrow=1000, ncol=2)
testz[telabel==2, ] <- testz[telabel==2, ] + 0.3
vec_x <- matrix(rnorm(1000*150), ncol=150) #generate tensor
x <- array(list(),150)
for (i in 1:150){
   x[[i]] <- array(vec_x[,i], c(10,10,10)) + amprod(coef, t(z[i,]), 4)[,,,1]
   x[[i]] <- M[[y[i]]] + atrans(x[[i]], dsigma)
}
vec_testx <- matrix(rnorm(1000*1000), ncol=1000)
testx <- array(list(), 1000)
for (i in 1:1000){
   testx[[i]] <- array(vec_testx[,i], c(10,10,10)) + amprod(coef, t(testz[i,]),
```

```
 4)[,,,1]
   testx[[i]] <- M[[telabel[i]]] + atrans(testx[[i]], dsigma)
}
dat.ten <- list(x=x, z=z, testx=testx, testz=testz, vec_x=t(vec_x),
   vec_testx=t(vec_testx), y=y, testy=telabel)
```

Estimation of covariance matrices in the TDA/CATCH model

Denote the sample mean of Class k by $\overline{\mathbf{X}}_k$. We first center $\mathbf{X}_i$ within class to obtain the residuals:

$$\widehat{\mathbf{E}}_i = \mathbf{X}_i - \widehat{\boldsymbol{\mu}}_k = \mathbf{X}_i - \overline{\mathbf{X}}_k.$$

Further unfold $\widehat{\mathbf{E}}_i$ along the j-th mode to obtain $\mathbf{W}_{i(j)}$ and find $\widetilde{\mathbf{S}}_j = (n\prod_{l\neq j}^{M} p_l)^{-1}\sum_{i=1}^{n}\mathbf{W}_{i(j)}(\mathbf{W}_{i(j)})^T$. Then our estimator for $\boldsymbol{\Sigma}_j$ is defined as

$$\widehat{\boldsymbol{\Sigma}}_j = \widetilde{s}_{j,11}^{-1}\widetilde{\mathbf{S}}_j \text{ for } j = 1,\ldots,M-1; \widehat{\boldsymbol{\Sigma}}_M = \frac{\widehat{\text{var}}(X_{1\ldots 1})}{\prod_{j=1}^{M}\widetilde{s}_{j,11}}\widetilde{\mathbf{S}}_M. \tag{30}$$

Definition of F-test statistic

The F-test statistic used to preprocess GDS1615 data is defined as

$$f_j = \frac{\sum_{k=1}^{K} n_k(\widehat{\boldsymbol{\mu}}_{kj} - \widehat{\boldsymbol{\mu}}_j)^2/(K-1)}{\sum_{i=1}^{n}(\mathbf{X}_j^i - \widehat{\boldsymbol{\mu}}_{Y^i,j})^2/(n-K)}, \tag{31}$$

where $\mathbf{X}_j^i$ is the j-th variable of i-th observation and $\widehat{\boldsymbol{\mu}}$ is the grand mean.

Yuqing Pan
Florida State University
117 N. Woodward Ave., Tallahassee, FL 32306
USA
yuqing.pan@stat.fsu.edu

Qing Mai
Florida State University
117 N. Woodward Ave., Tallahassee, FL 32306
USA
mai@stat.fsu.edu

Xin Zhang
Florida State University
117 N. Woodward Ave., Tallahassee, FL 32306
USA
henry@stat.fsu.edu