LDATree: Integrating LDA and Decision Trees for Enhanced Classification with Missing Values

by

Siyu Wang

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Statistics)

at the

UNIVERSITY OF WISCONSIN–MADISON

2024

Date of final oral examination: December 5th, 2024

The dissertation is approved by the following members of the Final Oral Committee:

Wei-Yin Loh, Professor, Statistics
Peter Chien, Professor, Statistics
Lu Mao, Associate Professor, Biostatistics and Medical Informatics
Yinqiu He, Assistant Professor, Statistics
Yongyi Guo, Assistant Professor, Statistics

ACKNOWLEDGMENTS

I want to express my deepest gratitude to my advisor, Prof. Wei-Yin Loh. I never imagined that a single course in my first semester in Madison would fundamentally change my life. He showed me that learning can be incredibly fun and that Statistics can be learned in a completely new and exciting way! Six years have passed, but I still feel the same passion I had in his STAT 602. He supported me every step of the way, especially during COVID-19, allowing me to work remotely and stay with my family, who were overjoyed to have me home. He provided tremendous comfort and encouragement when I got stuck in my research. And that's not all — he's also an incredible tour guide! Thanks to him, Albert and I enjoyed the most authentic street food in Singapore. Words cannot express how much he has shaped my life. I know I will miss our weekly conversations deeply.

I would like to sincerely thank Prof. Yongyi Guo and Prof. Yinqiu He for their valuable time, suggestions, and comments. Special thanks to Prof. Peter Chien, Prof. Lu Mao, and Prof. Sameer Deshpande for raising insightful questions during my preliminary exam. I greatly appreciate the fresh perspectives you brought to this project.

I want to thank my internship manager, Allen Shu, for his support and guidance. The project we completed together was crucial in my full-time job search. I also want to thank Roger Chang, a friendly and athletic guy who always bought me milk tea, and Ray Su, who is incredibly smart, candid, and resourceful.

I owe my friends a huge thanks; without their support, this Ph.D. journey would have been impossible. Thanks to Bingye Bai and Mingyang Ni, who were there for me when my grandpa passed away. Thanks to Dongyu Lyu, Sicheng Bian, Yu Luan, Zhongrui Zhang, and Xiaoxuan Ren, who spent countless nights helping me develop memorization skills (through poker games). Thanks to Jiyun Zhang, Minghao Sun, Hanzhe Teng, Changhao Shi, Rencong Zhang, Peiji Zhang, Baiheng Chen, and Yudi Wang for their hospitality when I needed companionship. Thanks to Xiaoxi Sun and Zongru Li for sharing their genuine insights when I needed them most. Thanks to Keyanna Qi and Tim Mai for their deep care and for making Madison feel like home. I am incredibly grateful to the best roommates in the world, Linquan Ma and Jiatong Li, who treated me with patience, understanding, and love. You always have my deepest respect — and not just because of the countless delicious meals you cooked. To my best friend, Kehui Yao, and his wife, Yingjiang Jiang, I cannot thank you enough; you have become an irreplaceable part of my life.

I want to thank my wife, Shiyu Ding, and my family for their support. Lastly, I want to thank God. You have guided me through each step of my life and have shown me that there is someone in this world who knows me completely and still loves me unconditionally.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

We present a new classification tree method called LDATree and its variant stepLDATree. We first revise some of the LDA variants and develop an improved version of stepwise LDA that can better handle non-invertible within-class scatter matrices. Then, this stepwise LDA serves as the node model and splitting rule in the decision tree framework, allowing for fast and non-axis-orthogonal splits. Within the LDATree framework, we explored several approaches for handling missing values and found that simple methods work best: using the median with missing flags for numerical variables and new levels for categorical variables. Simulation studies and real data analysis show that LDATree and stepLDATree generally outperform single-tree methods and help identify several scenarios where LDATree and stepLDATree outperform the random forest. We also implement our algorithms into R packages `folda` and `LDATree`, which are available on CRAN.

1 INTRODUCTION

Linear discriminant analysis (LDA), which assumes Gaussian densities on the covariates, is a powerful linear method for classification problems. LDA aims to find linear combinations of features that can best separate the groups by maximizing the ratio of between-group variance and within-group variance. Another popular classifier with a linear decision boundary is the decision tree, which recursively partitions the sample space into rectangular regions. We aim to combine the advantages of both LDA and decision trees to develop a better classifier.

One of the biggest problems of the tree-based methods is that it usually cuts in an axis-orthogonal direction. For numerical covariates, the splitting rule for each non-terminal node has the form $x_i \leq a$. For categorical covariates, the splitting rule can be written as $x_i \in \{c_1, c_2, \ldots, c_k\}$ where $\{c_1, c_2, \ldots, c_k\}$ are levels of the variable X_i. Alternatively, this can be written as $\bigcup_{i=1,2,\ldots,k}(D_i = 1)$ by transforming X_i into a 0/1 dummy matrix $D_1, D_2, D_3, \ldots, D_J$. The decision tree loses its effectiveness when the real decision boundary is not orthogonal to the axes. e.g., Figure 1.1. The decision tree has 10 splits and uses a staircase function to approximate the boundary line. However, achieving better performance requires adding many additional splits, consequently demanding more data. Additionally, this approximation's accuracy decreases as the dimensions increase, e.g., Figure 1.2. The gap between the true boundary and the fitted boundary from the decision tree becomes larger, and it might take the decision tree hundreds of splits to approximate this hyperplane well enough. LDA, which can directly fit a hyperplane decision boundary in the high-dimension space, is more suitable for this problem.

On the other hand, LDA uses the Gaussian density assumption, and the decision boundary for each class is usually a polygon. Therefore, LDA's performance may decline when the class centroids are closely situated together or when symmetry is

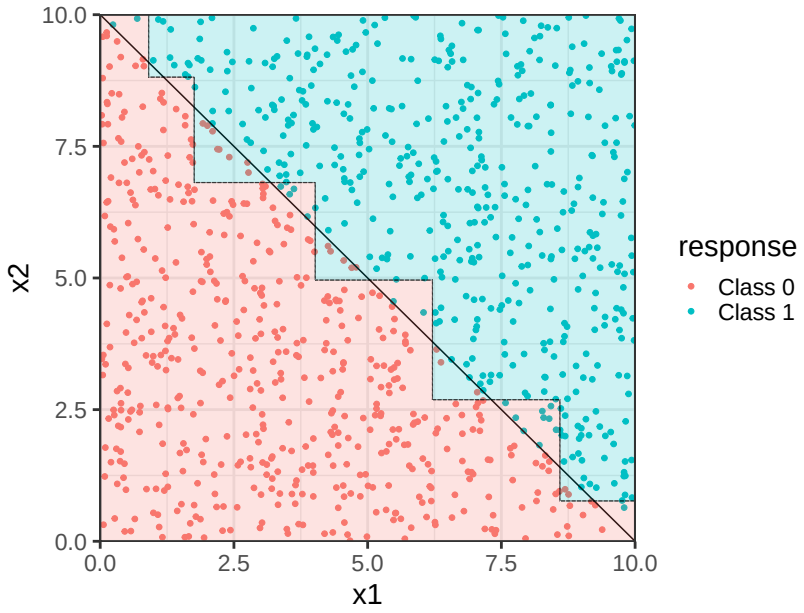

Figure 1.1: 2D Decision boundary in Section 1. Straight line: actual boundary. Staircase: from `rpart` (a decision tree algorithm in R), and the colored background represents the prediction regions from `rpart`.

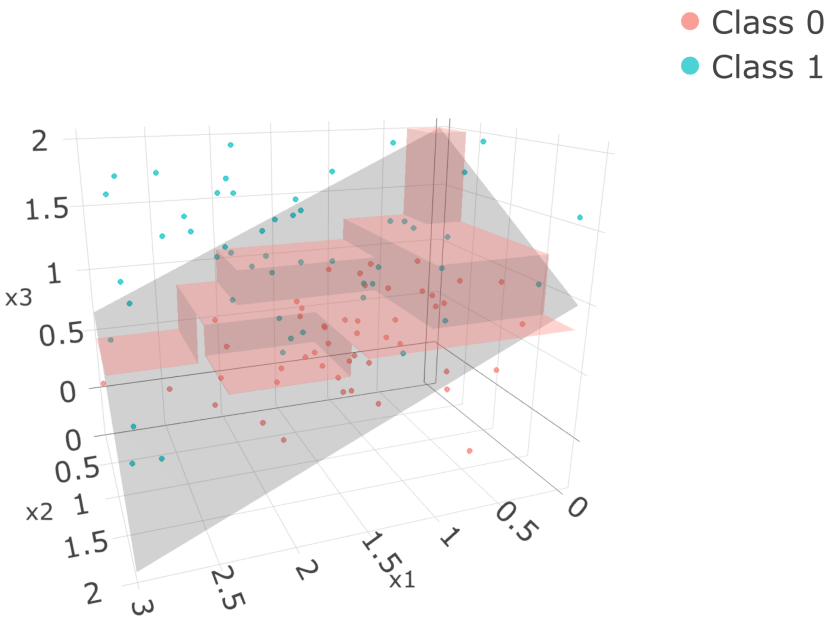

Figure 1.2: 3D Decision boundary in Section 1. Gray surface: actual boundary. Red step-function surface: from `rpart`.

present (Figure 1.3). By first applying tree-based methods to split the data on the x variable, the class centroids can be separated and the symmetry destroyed. This enables LDA to carry out its classification task more effectively (Figure 1.4).

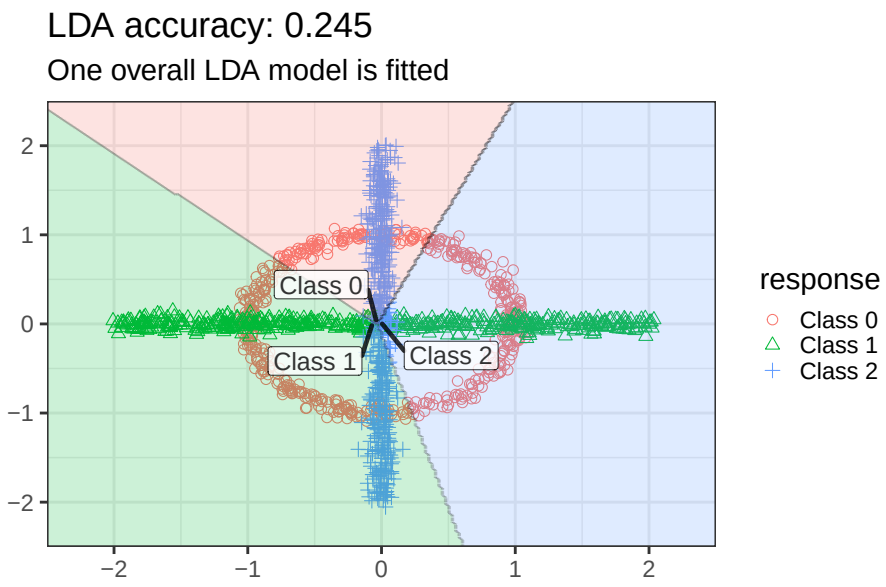

Figure 1.3: Decision boundary from LDA in Section 1. Points are the observed data; the colored background represents LDA prediction regions. Text labels show the class centroids.

LDA can be viewed as a dimensionality reduction technique that projects data onto a lower-dimensional space while preserving the information most critical for distinguishing between groups. These projections in the lower dimension, also known as linear discriminants, can be useful when building a tree. Some previous approaches have also integrated LDA components into decision trees, such as FACT (Loh and Vanichsetakul, 1988) and CRUISE (Kim and Loh, 2003). However, these methods have some limitations and may result in information loss. For instance, FACT ignores the relationship between the response variable and covariates by using principal component analysis (PCA) during variable selection, while CRUISE uses only the top two variables to fit LDA, which is suboptimal. Therefore, none of these methods can be considered truly recursive LDA, which is the goal of this work. The proposed method

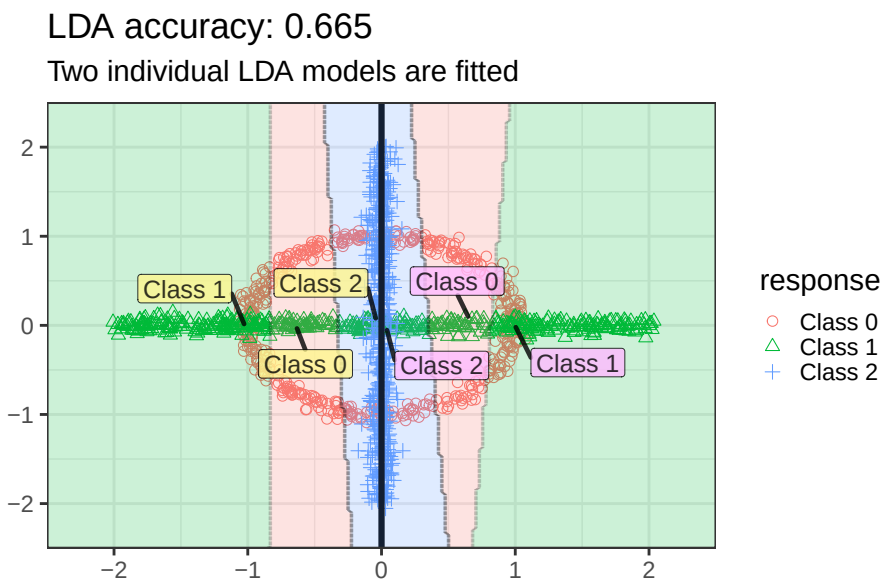

Figure 1.4: Decision boundary from LDA after splitting on $x = 0$ (Section 1).

will be referred to as LDATree for the remainder of the paper.

Another problem is missing values in the datasets, which can compromise the model's effectiveness. LDA's decision boundary is based on linear combinations of covariates, and missing values in any covariate will make the boundary uncomputable. This issue with missing values also appears in other machine-learning techniques, such as the random forest. Thus, we examine some missing value solutions using real data to identify a solid approach for LDA.

This thesis is organized as follows: **Section 2** presents a brief literature review of previous recursive partitioning methods, setting the stage for developing our proposed LDATree method. **Section 3** is the core of this thesis, which documents our effort to refine each component of a decision tree: node models, stopping rules, and splitting rules. **Section 4** discusses different missing value imputation methods and identifies the best method for LDATree using real datasets. **Section 5** contains results from synthetic datasets, while **Section 6** contains results from real data analysis. These two sections also present use cases for stepLDATree and LDATree, respectively. Lastly,

Section 7 concludes the thesis and lists some limitations and directions for future work.

2 BACKGROUND

Recursive partitioning methods are based on the idea of recursively dividing a dataset into subsets based on the values of its features, such that the resulting partitions are as homogeneous as possible with respect to the target response variable. The decision tree, one of the recursive partitioning methods, can be summarized in a top-down tree graph. At each tree node, the sample is split into two or more subsets using a criterion determined through a greedy search to minimize within-node heterogeneity. As the splitting goes on, the decision tree will have more nodes, and each node will have fewer data but be more homogeneous. In the decision tree, child nodes are the nodes that branch out from a parent node. Nodes without child nodes are called terminal nodes (or leaves), while those with child nodes are intermediate nodes. In the simplest classification setting, the plurality class in each terminal node will be the prediction of that node. During prediction, new data will start from the root (top) node, recursively go to one of the child nodes of the current node based on the splitting rules, and eventually end up in one of the terminal nodes.

Compared with other machine learning methods, decision trees are easy to interpret and understand, as they represent the decision-making process in a tree-like structure. Both categorical and numerical data can be well handled by decision trees. Decision trees are non-parametric, meaning they do not require any assumptions about the underlying data distribution. Decision trees are robust to outliers due to the nature of their interval-like cuts on numerical variables, and they can handle missing data in many ways. One method is through surrogate splits, where the tree grows on a different backup variable, and the cut is chosen to mimic the original cut as closely as possible. Another method is through missing flags, where all observations with missing values are assigned to one of the child nodes by minimizing a pre-defined loss function. Last but not least, it has an embedded interaction structure. For most models,

you have to pre-specify the interactions in the model, which will dramatically increase the number of variables and sometimes create a data sparsity problem when the factor variable involved in the interaction has too many levels. Moreover, interaction might only exist in parts of the sample. Using a constant coefficient to model the interaction throughout the sample space might dilute its effect. In the classification tree, the interaction is added to the model automatically, and its effect is local — only in that particular node.

2.1 Existing LDA-Integrated Decision Tree Methods

In this section, we review two existing decision tree approaches that incorporate LDA into their models.

FACT (Loh and Vanichsetakul, 1988) It is a decision tree with multiway splits rather than a binary tree. This approach supports linear combination splits, whose splitting rule has the form $\bigcup_i \{\sum_j a_{ij} x_{ij} \leq b_i\}$ and has been demonstrated to yield higher accuracy than univariate splits. While the CART method (Breiman et al., 1984) exclusively uses numerical variables to create linear combination splits, FACT adopts a more comprehensive approach and includes categorical variables as well. In particular, a categorical variable is transformed into a 0-1 dummy vector and subsequently projected onto the largest discriminant coordinate (CRIMCOORD). In other words, CRIMCOORD records the coefficients from the first discriminant score in an LDA model where the response variable is the original class and the covariates are the 0-1 dummy variables. Unlike earlier techniques that demanded exhaustive searches for linear combination splits, FACT expedites the process by using LDA to find the combinations in one shot. After fitting an LDA model within the node, the space is partitioned into polygons, with each polygon being a node associated with a linear discriminant function. To

prevent the appearance of a singular covariance matrix in LDA, FACT uses PCA to reduce the dimension prior to applying LDA. Additionally, FACT supports unequal misclassification costs, dispersion splits (splits on $\mid x_i - \bar{x} \mid$ instead of x_i), and polar transformations.

CRUISE One of the options available in CRUISE (Kim and Loh, 2003) adopts a univariate split and fits a bivariate LDA model within each terminal node. First, regarding variable selection for univariate splits, FACT utilizes the ANOVA F-statistic to rank both numerical and categorical variables (through CRIMCOORD). Loh and Shih discovered that FACT has selection bias towards categorical variables, leading them to develop the QUEST method (Loh and Shih, 1997), which applies ANOVA F-statistic to numerical variables and the contingency table χ^2-statistic to categorical variables. However, the ANOVA F-statistic can only detect differences in group means and is insensitive to the difference in the shape of the density functions. To fix that, CRUISE divides all numerical variables at their sample quartiles and then conducts a χ^2 test. Meanwhile, CRUISE uses only two variables to fit LDA in terminal nodes, allowing the model to share some data complexity while maintaining interpretability. It screens $\binom{p}{2}$ pairs of variables and uses the MANOVA Wilk's Λ-statistic (for testing the equality of the class mean vectors) to select the best pair of variables. In contrast to FACT, which uses a pre-stopping rule, CRUISE determines the optimal tree size by initially constructing an oversized tree and subsequently pruning it back using cost-complexity pruning via cross-validation.

2.2 Case Study: Fishcatch Dataset

Moving forward, we will use the fishcatch dataset as a case study to demonstrate the results of these decision tree approaches. The Fishcatch data set contains seven

measurements of 159 fish caught in Finland's Lake Laengelmavesi, obtained from the *Journal of Statistics Education* data archive (`https://jse.amstat.org/datasets/fishcatch.dat.txt`). The fish are from 7 species: (1) 35 Bream, (2) 11 Parkki, (3) 56 Perch, (4) 17 Pike, (5) 20 Roach, (6) 14 Smelt, (7) 6 Whitefish. The `sex` variable has 87 cases (54.7%) missing, and the `weight` variable has 1 case (0.6%) missing. For now, we focus on the complete dataset without missing values and ignore the `sex` variable. A hidden feature of this dataset is that it is mainly sorted by `Length1` within each fish species, with only three exceptions. For the fish with missing `weight`, we imputed it by 550g, given that the fish above and below the observation had `weight` of 500g and 600g, respectively. Table 2.1 provides detailed information about the data. Our task is to use the six variables to predict the fish species.

Table 2.1: Variables in the fishcatch dataset

`Weight`	Weight of the fish (in grams)
`Length1`	Length from the nose to the beginning of the tail (in cm)
`Length2`	Length from the nose to the notch of the tail (in cm)
`Length3`	Length from the nose to the end of the tail (in cm)
`Heightpc`	Maximal height as percent of Length3
`Widthpc`	Maximal width as percent of Length3

Figure 2.1 presents a decision tree created by the FACT algorithm. An LDA model is fitted at each intermediate node, unless the pre-stopping condition is met. When an LDA model is fitted, the sample is partitioned based on the predicted value from the model. For instance, seven child nodes are formed in the root node as all seven classes are predicted. On the other hand, fewer classes are predicted in lower nodes, resulting in fewer child nodes. The plot shows that FACT performed well in predicting Bream, Parkki, Pike, and Smelt species, but it struggled to differentiate between Perch, Roach, and Whitefish species. Using leave-one-out cross-validation, FACT produces 31 prediction errors among 159 samples.

Figure 2.2 displays a decision tree generated by the CRUISE algorithm. The algorithm splits on `Heightpc`, and the splitting points are calculated in a similar fashion to

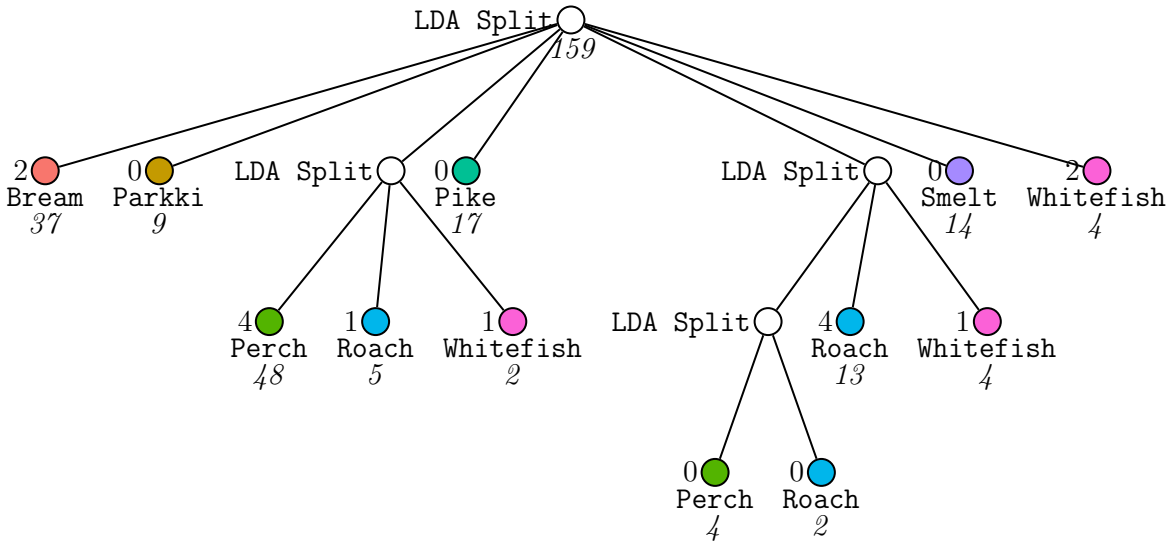

Figure 2.1: FACT Decision Tree on fishcatch Dataset (Section 2.2), showing predicted species and sample size under each node. Misclassified cases (training errors) are indicated by the number next to the node.

FACT, but with an added Box-Cox transformation before fitting LDA to better satisfy the normal assumption. Subsequently, a bivariate LDA model is fitted in each terminal node. CRUISE made 10 errors out of 159 samples in the leave-one-out cross-validation, which is a substantial improvement from FACT. Compared to FACT, CRUISE uses univariate splits which can be interpreted. For instance, the first split is based on `Heightpc`, which denotes the percentage ratio between height and length. As a result, fish species like Bream and Pike (Figure 2.3) can be more easily distinguished, with these two species occupying opposite ends of the decision tree. CRUISE also encounters difficulty differentiating between Perch, Roach, and Whitefish species, as these classes contribute to all of the training errors.

Lastly, an LDA model is fitted to the data, and Figure 2.4 displays a scatter plot of the fish based on the first two (out of six) discriminant coordinates. With just 1 error, LDA achieves a significantly better performance in comparison to the earlier two techniques. This plot also reveals a drawback of the marginal analysis on low dimensions — overlapping. Roach and Whitefish overlap on this 2D plot, but this plot

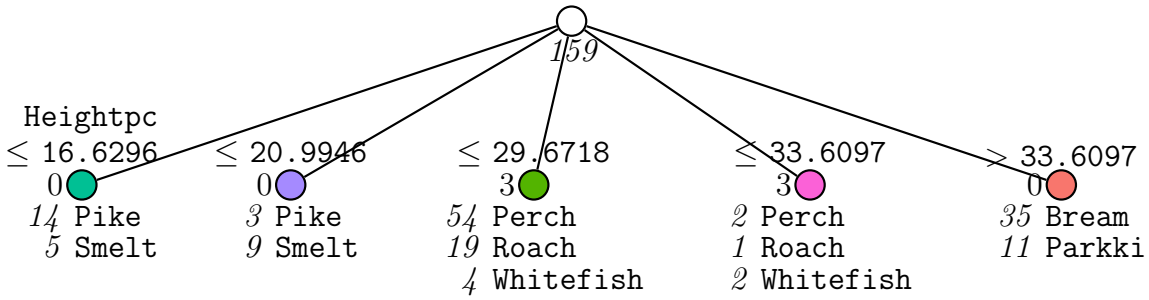

Figure 2.2: CRUISE Decision Tree on fishcatch Dataset (Section 2.2): Species distribution is shown below each node, with the training error from bivariate LDA indicated by the number next to the node. Splitting criteria is displayed above the nodes, and the resulting intervals are sorted in ascending order from left to right.

(a) Sea Bream (from fish-man.co.uk)

(b) Northern Pike (from Wisconsin Department of Natural Resources)

Figure 2.3: Two fish with different height-to-length ratios (Section 2.2).

is a 2D shadow from a 6D space. They are correctly classified with the help of the features not shown in the plot, which may help to explain why the linear combination splits perform better than univariate splits.

Our LDATree builds upon previous methods while introducing significant innovations that effectively address their key weaknesses and enhance overall performance. Starting from the next section, we will discuss all necessary components of a decision tree and our efforts to refine them.

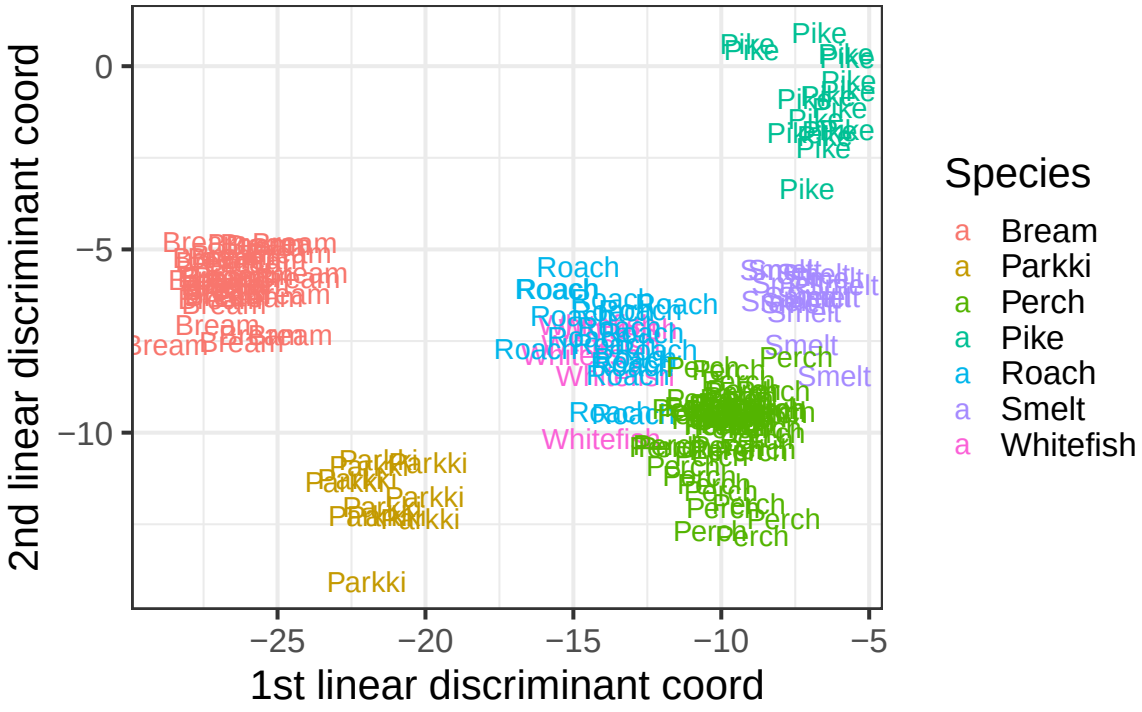

Figure 2.4: LDA Result on fishcatch Dataset (Section 2.2).

3 LDATREE & STEPLDATREE ALGORITHM

3.1 Node Model: LDA Variants

The node model specifies the model used at each terminal node, and its simplest form is the majority vote which classifies each instance into the most common class within the node, as seen in CART and Random Forest. In our algorithm, we use an LDA model instead of this majority vote, introducing a more sophisticated classification rule for prediction and enhancing the algorithm's ability to capture non-axis-orthogonal patterns. However, the original LDA cannot handle certain scenarios effectively, and the most popular R implementation `MASS::lda` (Venables and Ripley, 2002) has its weaknesses, which affect LDA's performance. In this section, we will explore some alternatives to address these issues.

In traditional LDA, the goal is to find linear projections that best separate classes. These decision boundaries are found by maximizing the ratio of between-class scatter matrix to within-class scatter matrix, similar to optimizing a signal-to-noise ratio. Suppose we have a data matrix $\mathbf{X} \in \mathbb{R}^{N\times M}$ with N observations and M features. Our response $\mathbf{y} \in \mathbb{R}^N$ is a factor vector containing J classes. Let $\mathbf{x}_{ji} \in \mathbb{R}^M$ represent the ith observation from class j, $\bar{\mathbf{x}}_j \in \mathbb{R}^M$ be the mean vector for class j derived from its n_j instances, and $\bar{\mathbf{x}} \in \mathbb{R}^M$ denote the overall mean vector across all samples. $\mathbf{H}_B \in \mathbb{R}^{J\times M}$ and $\mathbf{H}_W \in \mathbb{R}^{N\times M}$ are defined as:

$$\begin{aligned}
\mathbf{H}_B &= \left[\sqrt{n_1}\left(\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}\right), \sqrt{n_2}\left(\bar{\mathbf{x}}_2 - \bar{\mathbf{x}}\right), \ldots, \sqrt{n_J}\left(\bar{\mathbf{x}}_J - \bar{\mathbf{x}}\right)\right]^T, \\
\mathbf{H}_W &= \left[\left(\mathbf{x}_{11} - \bar{\mathbf{x}}_1\right), \ldots, \left(\mathbf{x}_{1n_1} - \bar{\mathbf{x}}_1\right), \left(\mathbf{x}_{21} - \bar{\mathbf{x}}_2\right), \ldots, \left(\mathbf{x}_{2n_2} - \bar{\mathbf{x}}_1\right), \ldots, \left(\mathbf{x}_{Jn_J} - \bar{\mathbf{x}}_J\right)\right]^T.
\end{aligned} \tag{3.1}$$

Then, the between-class scatter matrix $\mathbf{S}_B$, within-class scatter matrix $\mathbf{S}_W$, and total

scatter matrix $\mathbf{S}_T$ can be defined as:

$$\begin{aligned}
\mathbf{S}_B &= \sum_{j=1}^{J} n_j \left(\bar{\mathbf{x}}_j - \bar{\mathbf{x}}\right)\left(\bar{\mathbf{x}}_j - \bar{\mathbf{x}}\right)' &&= \mathbf{H}_B^T \mathbf{H}_B \\
\mathbf{S}_W &= \sum_{j=1}^{J} \sum_{i=1}^{n_j} \left(\mathbf{x}_{ji} - \bar{\mathbf{x}}_j\right)\left(\mathbf{x}_{ji} - \bar{\mathbf{x}}_j\right)' &&= \mathbf{H}_W^T \mathbf{H}_W \\
\mathbf{S}_T &= \sum_{j=1}^{J} \sum_{i=1}^{n_j} \left(\mathbf{x}_{ji} - \bar{\mathbf{x}}\right)\left(\mathbf{x}_{ji} - \bar{\mathbf{x}}\right)' &&= \mathbf{S}_B + \mathbf{S}_W .
\end{aligned} \tag{3.2}$$

Fisher's criterion aims to find transformation vectors $\mathbf{w}_{M\times 1}$ that maximizes the ratio:

$$\arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{3.3}$$

The optimal $\mathbf{W}$ is derived by solving an eigenvalue problem involving $\mathbf{S}_W^{-1}\mathbf{S}_B$. The resulting $\mathbf{W}$ projects the original data $\mathbf{X}$ into orthogonal linear discriminant scores $\mathbf{X}\mathbf{w}_i$, ranked in a descending order of their signal-to-noise ratios.

However, challenges arise when the within-class scatter matrix, $\mathbf{S}_W$, is not invertible, such as when there are more variables than observations or when variables are linearly dependent. Given that $\mathrm{rank}(\mathbf{S}_B) = \min(M, J-1)$, the transformation matrix $\mathbf{W}$ we seek has at most $M \times (J-1)$ dimensions. That's why typically, in a J-class classification, LDA gives us $J-1$ discriminant scores. For the within-class scatter matrix, $\mathrm{rank}(\mathbf{S}_W) = \min(N-J, M)$. When $M > N-J$, $\mathbf{S}_{W(M\times M)}$ becomes singular. This situation makes directly computing LDA's linear boundary impossible. One might consider reducing the number of variables by selecting the top-K variables based on certain criteria, such as importance scores, to make the scatter matrix invertible. Yet, this method has drawbacks:

- Importance scores are often based on other machine learning methods with

different criteria, which may not align with LDA's objective function.

- Variables are typically ranked by their marginal (or univariate) effects, not the combined effect from linear combinations that LDA seeks to find.
- Some machine learning methods need to finish building the entire classifier before they can rank the variables, which can be time-consuming.

Therefore, it may be more effective to explore solutions within the LDA family. This issue, known as the small sample size (SSS) problem, has been widely addressed in the literature with several solutions proposed (Mai, 2013; Tharwat et al., 2017). We will discuss some of these approaches and identify the most suitable one for our application.

Existing Methods

PCA-LDA (Belhumeur et al., 1997) This method, also known as Discriminant Analysis of Principal Components (DAPC) (Jombart et al., 2010), uses Principal Component Analysis (PCA) to find a smaller subspace where the within-class scatter matrix isn't singular and then apply Fisher's criterion. However, this PCA-LDA method focuses only on the structure of independent variables and neglects the dependent variable. Additionally, choosing the right number of principal components (PCs) is a challenge in practice.

Pseudo-inverse LDA (Fukunaga, 2013) To find the inverse of a singular $\mathbf{S}_W$ matrix, one alternative is the Moore-Penrose inverse, also used by the R package `MASS::lda` to mitigate the SSS problem. It involves using Singular Value Decomposition (SVD) to identify the rank of $\mathbf{S}_W$ and then projecting the data onto the space defined by the singular vectors with non-zero singular values. Fisher's criterion is then applied in this subspace to identify the directions with the most discriminant power. It is worth mentioning that this approach is equivalent to PCA-LDA when the number of principal components is set to $\mathrm{rank}(\mathbf{S}_W)$.

Unlike the general PCA-LDA, which may use varying numbers of principal components, this method maintains as much information as possible from the original $\mathbf{S}_W$'s column space. It discards the null space of $\mathbf{S}_W$, calculates $\mathbf{S}_B$ in the reduced space, and extracts up to $J-1$ discriminant directions. However, like PCA-LDA, this method might overlook important information in the null space of $\mathbf{S}_W$.

Null Space LDA (Chen et al., 2000) This method reduces to the original LDA when $\mathbf{S}_W$ is of full rank. If not, it searches for the most discriminant directions within the null space of $\mathbf{S}_W$. The rationale is that if there exists a transformation $\mathbf{w}$ such that $\mathbf{w}^T\mathbf{S}_W\mathbf{w} = 0$ and $\mathbf{w}^T\mathbf{S}_B\mathbf{w} \neq 0$, Fisher's criterion is maximized to ∞. Therefore, the null space of $\mathbf{S}_W$ is considered to contain the most valuable discriminant information and should be used rather than discarded. However, this approach relies solely on the null space of $\mathbf{S}_W$, totally neglecting valuable information in the column space of $\mathbf{S}_W$ and being ineffective when the null space of $\mathbf{S}_W$ lies in the null space of $\mathbf{S}_B$, meaning there is no discriminant information in the null space of $\mathbf{S}_W$.

Direct LDA (Yu and Yang, 2001) Many LDA variants use simultaneous diagonalization to implement Fisher's criterion. Typically, they first transform the within-class scatter matrix ($\mathbf{S}_W$) into an identity matrix and then find an orthogonal transformation to maximize the between-class scatter matrix ($\mathbf{S}_B$). The final selected directions correspond to the eigenvectors associated with the largest eigenvalues. Conversely, Direct LDA adopts a different approach by transforming $\mathbf{S}_B$ first and then $\mathbf{S}_W$. This is because the ratio in Fisher's criterion becomes zero when $\mathbf{w}^T\mathbf{S}_B\mathbf{w} = 0$, indicating that the null space of $\mathbf{S}_B$ is not informative. Direct LDA, therefore, first finds a subspace with a maximum of $J-1$ dimensions that maximizes $\mathbf{S}_B$, and then searches for the smallest $\mathbf{S}_W$ within this subspace.

An advantage of Direct LDA is its speed, especially in handling high-dimensional data. This is because the SVD involved in simultaneous diagonalization is of a smaller dimension, $N \times J$, instead of $N \times M$. However, as noted in (Gao and Davis, 2006), this method has significant drawbacks. For example, beginning simultaneous diagonalization with $\mathbf{S}_W$ preserves more information, allowing for more dimensions to find the largest $\mathbf{S}_B$. Conversely, starting with $\mathbf{S}_B$ reduces the dimensionality to $J - 1$, often leaving insufficient scope to minimize $\mathbf{S}_W$ effectively. Another perspective is that this method prefers dimensions where $\mathbf{S}_B$ is large; however, typically these dimensions also have a large $\mathbf{S}_W$, which results in a less substantial increase in the overall ratio from Fisher's criterion. Additionally, unlike some other LDA variants, Direct LDA does not reduce to the original LDA when $\mathbf{S}_W$ is invertible.

LDA/GSVD (Howland et al., 2003; Ye et al., 2004) The incorporation of Generalized Singular Value Decomposition (GSVD) into LDA enhances the efficiency of simultaneous diagonalization. Denoting the diagonal elements (singular values) of between-class and within-class scatter matrices after GSVD as α_i^2 and β_i^2, respectively, the following properties hold:

$$1 \geq \alpha_1^2 \geq \alpha_2^2 \geq \cdots \geq \alpha_M^2 \geq 0, \tag{3.4a}$$

$$0 \leq \beta_1^2 \leq \beta_2^2 \leq \cdots \leq \beta_M^2 \leq 1, \tag{3.4b}$$

$$\alpha_i^2 + \beta_i^2 = 1. \tag{3.4c}$$

A modified Fisher's criteria is used here, which maximizes $\sum_{i=1}^{M} \alpha_i^2$. To preserve all discriminant information, it suffices to keep only the first $\mathrm{rank}(\mathbf{S}_B)$ columns from $\mathbf{W}$ where $\alpha_i^2 > 0$. Additionally, this method reduces to the original LDA when $\mathbf{S}_W$ is of full rank.

It is noteworthy that this method is equivalent to Uncorrelated LDA (ULDA) (Ye and Yu, 2005; Ji and Ye, 2008), which provides an alternative perspective. ULDA uses a different way of simultaneous diagonalization, first transforming $\mathbf{S}_T$ and then $\mathbf{S}_B$, where $\mathbf{S}_T = \mathbf{S}_B + \mathbf{S}_W$ represents the total scatter matrix. The advantage of involving $\mathbf{S}_T$ is that its null space is the intersection of the null spaces for $\mathbf{S}_W$ and $\mathbf{S}_B$ ($\mathbf{w}^T\mathbf{S}_B\mathbf{w} = \mathbf{w}^T\mathbf{S}_W\mathbf{w} = 0$). Consequently, the null space of $\mathbf{S}_T$ can be safely ignored without affecting the maximization of Fisher's ratio. ULDA then proceeds with the original LDA in the remaining subspace.

In the same paper, the authors introduce an extension known as Orthogonal LDA (OLDA), which includes an extra QR decomposition step to make the transformation matrix orthogonal. (Ye and Xiong, 2006) demonstrated that OLDA is equivalent to Null Space LDA given that

$$\operatorname{rank}(\mathbf{S}_T) = \operatorname{rank}(\mathbf{S}_B) + \operatorname{rank}(\mathbf{S}_W). \tag{3.5}$$

In cases where this condition is not met, Orthogonal LDA tends to perform better by preserving more discriminant directions.

Regularized LDA (Friedman, 1989) To avoid the non-invertibility of $\mathbf{S}_W$, Regularized LDA adds a constant to the diagonal, creating $\mathbf{S}'_W = \mathbf{S}_W + \eta\mathbf{I}_M$. This adjustment ensures invertibility. However, choosing the right value for η requires tuning, and a poor choice could lead to unsatisfactory results. Additionally, this method might be less effective in very high-dimensional settings since it doesn't involve any dimension reduction and may not effectively distinguish signal from noise.

Penalized LDA (Witten and Tibshirani, 2011) Penalized LDA introduces a penalty term to shrink the discriminant vectors, distinguishing it from Regularized LDA.

To deal with non-invertible $\mathbf{S}_W$, it ignores the correlation structure by approximating $\mathbf{S}_W$ with a diagonal matrix. Using the LASSO penalty, this approach can be formulated as:

$$\arg\max_{\mathbf{w}} \left\{ \mathbf{w}^T \hat{\mathbf{S}_B} \mathbf{w} - \lambda \|\mathbf{w}\|_1 \right\} \text{ subject to } \mathbf{w}^T \tilde{\mathbf{S}_W} \mathbf{w} \leq 1 \tag{3.6}$$

where $\hat{\mathbf{S}_B}$ is a MLE or LSE of $\mathbf{S}_B$, and $\tilde{\mathbf{S}_W}$ is a full rank estimate of $\mathbf{S}_W$. The main challenges with this method include its tendency to converge to local optima due to the non-convexity of the problem and the heavy reliance on the choice and tuning of the penalty term. Additionally, finding the appropriate hyperparameter can be time-consuming, particularly when cross-validation is carried out on some large datasets.

Sparse Discriminant Analysis (Clemmensen et al., 2011) It is similar to the Penalized LDA but under the optimal scoring framework. Optimal scoring, as introduced in (Hastie et al., 1994), is a variant of LDA that operates within a regression context. It does not have a direct analytical solution but instead relies on an iterative optimization process. This process alternately optimizes the scores and the regression coefficients until convergence is achieved. Considering our response variable $\mathbf{y}_{N\times 1}$ is converted into dummy variables $\mathbf{Y}_{N\times J}$, the objective function with a LASSO penalty is:

$$\arg\min_{\mathbf{w},\boldsymbol{\theta}} \left\{ \|\mathbf{Y}\boldsymbol{\theta} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|_1 \right\} \text{ subject to } \boldsymbol{\theta}^T \mathbf{Y}^T \mathbf{Y} \boldsymbol{\theta} = 1. \tag{3.7}$$

Simulation

In this section, we will use a synthetic dataset to demonstrate the strengths and weaknesses of those LDA variants. Here are some details on how the simulations are conducted:

- The simulations were implemented in R and executed on the department's Linux server. `klaR::rda` is used for Regularized LDA, `sparseLDA::sda` for Sparse Discriminant Analysis, and `penalizedLDA::PenalizedLDA` for Penalized LDA. Hyperparameters are determined via Cross-Validation, and all other settings are left as default.

- We implemented the remaining five LDA variants as there are no available packages. Notably, most of these variants are primarily used as dimension-reduction tools in their original papers, typically followed by a K-Nearest Neighbors (KNN) classifier. However, due to KNN's slower prediction speed and lack of a simple closed-form solution, we use traditional likelihood-based LDA as the classifier throughout this thesis. Let $\boldsymbol{\mu}_j$ denote the centroid of class j, and π_j the prior probability of class j. The discriminant function of class j for any observation $\mathbf{x}$ is defined as

$$\delta_j(\mathbf{x}) = \mathbf{x}^T(\frac{\mathbf{S}_W}{N-J})^{-1}\boldsymbol{\mu}_j - \frac{N-J}{2}\boldsymbol{\mu}_j^T(\frac{\mathbf{S}_W}{N-J})^{-1}\boldsymbol{\mu}_j + \log\pi_j. \tag{3.8}$$

  Observations are classified to the class with the highest value of the discriminant function.

- For PCA-LDA, additional PCs are excluded once at least 95% of the total variance is explained or $\mathbf{S}_W$ becomes singular, whichever occurs first.

- All testing accuracies calculated in both simulations and real data analysis throughout this thesis are derived from a random 70/30 split of the entire data sample.

- R sometimes has stability issues with calculations involving small numbers. Additionally, when a matrix is singular, different methods of calculating its rank can yield inconsistent results, impacting the reliability of the algorithm. To address this, we implement some tricks, such as scaling the variables. While theoretically, scaling the variables doesn't affect LDA solutions, we find it significantly improves stability in practice.

- The classical LDA package `MASS::lda` in R triggers errors when $\mathbf{S}_W$ is singular but $\mathbf{S}_B$ is not, a limitation that also affects dependent packages like `sparseLDA`. To compute the discriminant function in Equation 3.8, we add a small positive constant (10^{-15}) to the diagonal elements of $\mathbf{S}_W$ if it is singular.

One significant distinction among LDA variants is their approach to handling situations where $\mathbf{w}^T\mathbf{S}_W\mathbf{w} = 0$ and $\mathbf{w}^T\mathbf{S}_B\mathbf{w} \neq 0$. This scenario implies that after the projection $\mathbf{w}$, $\mathbf{X}\mathbf{w}$ are constant within each group but differ across groups. While one might argue this scenario is rare in real datasets and more common in simulations, making it seemingly less relevant, it is critical to explore due to the nature of decision trees. As the tree splits, the sample size in each terminal node decreases, yet the number of variables remains constant. Therefore, it is common to encounter this pattern in terminal nodes where there are only tens of data points but hundreds of variables, particularly when the variables are categorical.

The simulation is conducted as follows: We first generate a response variable consisting of 10 classes, each with 200 observations. Next, we create a dummy matrix (one-hot-encoding) for the response variable, resulting in 10 variables. Additionally, we add 10 mutually independent standard normal noise variables and use these 20 variables for prediction.

The results are summarized in Table 3.1. NLDA, DLDA, LDA/GSVD, and Regularized LDA achieve perfect prediction accuracy. NLDA and LDA/GSVD have good

method	trainAcc	testAcc
DLDA	1 (0)	**1 (0)**
LDA/GSVD	1 (0)	**1 (0)**
NLDA	1 (0)	**1 (0)**
PCA-LDA	0.468 (0.028)	0.407 (0.029)
PLDA	0.149 (0.003)	0.102 (0.007)
SDA	0.112 (0)	0.112 (0)
penalizedLDA	0.112 (0)	0.112 (0)
regularizedLDA	1 (0)	**1 (0)**

Table 3.1: Simulation results in Section 3.1. Two standard deviations are recorded in parentheses. Data corresponds to the DummyMatrix-Class detailed in Table A.1.

performance since they are designed to handle this specific case. DLDA optimizes $\mathbf{S}_B$ first, avoiding complications from $\mathbf{w}^T\mathbf{S}_W\mathbf{w} = 0$. Regularized LDA adds a small disturbance to the diagonal of $\mathbf{S}_W$, making $\mathbf{w}^T\mathbf{S}_W\mathbf{w} \neq 0$, and thus be able to find those discriminative directions. PCA-LDA, being blind to the response variable, mixes signal variables with noise, leading to lower testing accuracy. The remaining three methods perform similarly to random guessing (0.1 testing accuracy) as they cannot effectively extract information from the null space of $\mathbf{S}_W$.

Real Data Analysis

We first examine performance on a dataset characterized by a nearly equal number of variables and observations, a scenario that can potentially lead to overfitting for some methods. The Parkinson dataset from the UCI Machine Learning Repository is used, consisting of 756 observations and 753 variables. Table 3.2 summarizes the results. DLDA and PCA-LDA appear to be the most effective classifiers in this problem. PCA-LDA incorporates a variable selection step through PCA, while DLDA, by initially focusing on $\mathbf{S}_B$, unintentionally performs variable selection by choosing at most $J - 1$ variables. The three LDA variants that require cross-validation marginally outperform random guessing (0.741 testing accuracy). Notably, LDA/GSVD, NLDA, and PLDA all exhibit overfitting, as indicated by their 100% training accuracy, resulting

method	trainAcc	testAcc
DLDA	0.819 (0.005)	**0.834 (0.012)**
LDA/GSVD	1 (0)	0.66 (0.021)
NLDA	1 (0)	0.541 (0.017)
PCA-LDA	0.926 (0.004)	**0.828 (0.013)**
PLDA	1 (0)	0.689 (0.011)
SDA	0.741 (0)	0.741 (0)
penalizedLDA	0.743 (0.004)	**0.751 (0.011)**
regularizedLDA	0.741 (0.004)	**0.759 (0.009)**

Table 3.2: Results on the Parkinson dataset in Section 3.1. Two standard deviations are recorded in parentheses. Data corresponds to the Parkinson-class detailed in Table A.1.

in performance that is worse than random guessing. Section 3.2 discusses our efforts to mitigate this using stepwise LDA.

To further assess the performance of LDA variants, we conducted tests on 49 datasets obtained from public sources. This collection is carefully selected to cover a wide range of datasets: from binary classification to multi-class classification with over 100 classes; from complete datasets to datasets where every observation has at least one missing entry; from datasets containing hundreds of observations to 70,000; and from 4 predictive variables to 3,000. We aim to make this collection a good representative of real-world datasets. Detailed descriptions of these datasets are provided in Table A.1. Missing values are handled using the techniques introduced in Section 4. To normalize results across different datasets, testing accuracies have been scaled using the arithmetic mean, and runtime has been scaled using the geometric mean. For methods that trigger errors or are not completed within the pre-set time limit (5 minutes), the results are treated as censored. In such cases, the runtime is set to 5 minutes, and the testing accuracies are imputed using the proportion of the plurality class in the training set.

The results are summarized in Figure 3.1. The best method would ideally be situated in the bottom right corner, indicating high accuracy and low runtime. It is clear that LDA/GSVD has the highest testing accuracy among all LDA variants. Its testing

accuracy falls within the confidence interval of PCA-LDA, yet it is significantly faster. DLDA turns out to be the fastest LDA variant, albeit with comparatively lower testing accuracy. The three methods in the top left corner — Penalized LDA, Regularized LDA, and Sparse Discriminant Analysis — are notably affected by the runtime limit. While they might be as effective as other candidates if given sufficient time, these slower methods are not considered practical and are disregarded. This is because the LDA model needs to be fitted for each node during tree construction. To maintain both speed and accuracy, such extensive computational costs are unaffordable. Consequently, LDA/GSVD is recognized as the most effective method thus far, and it will be selected as our node model for LDATree.

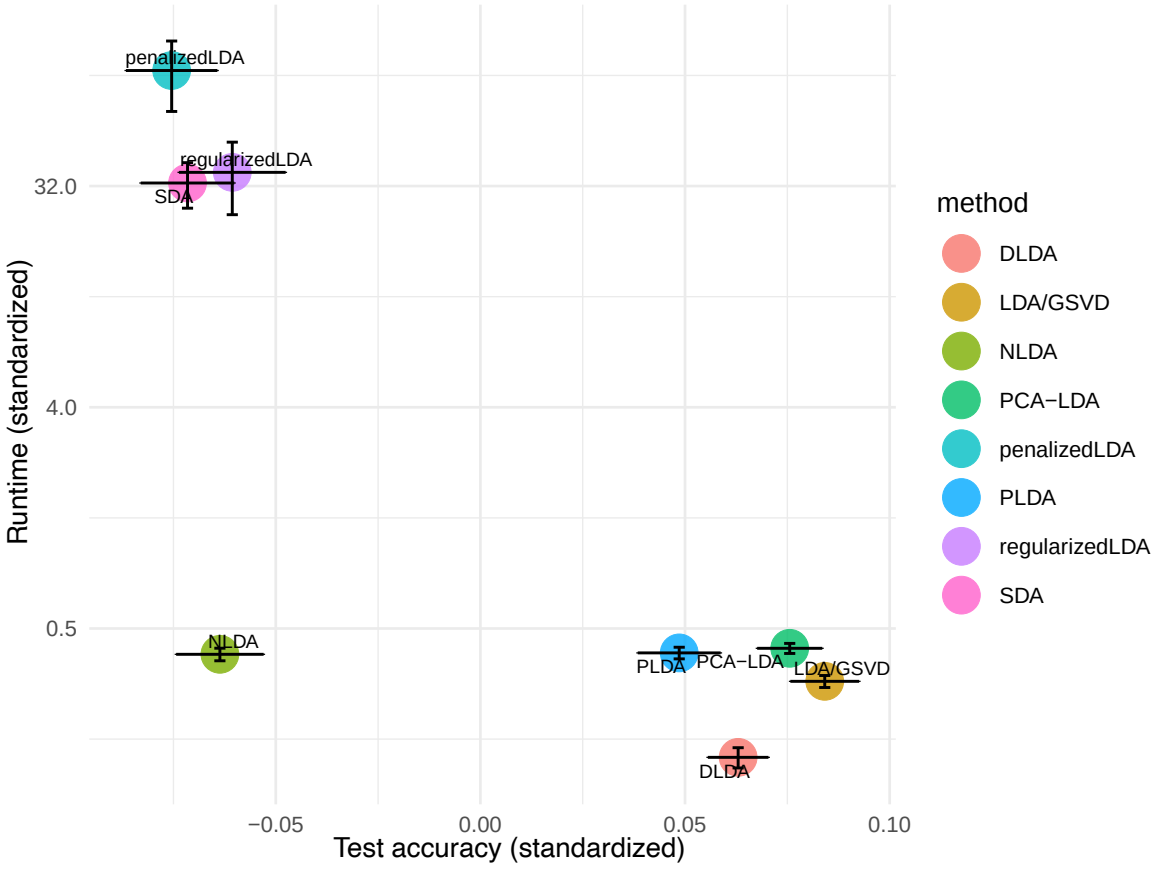

Figure 3.1: Real data analysis results in Section 3.1: Average runtime vs. average testing accuracy. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

Speed Enhancement

Next, we introduce our speed enhancement for the LDA/GSVD algorithm when $N > M$. (Ye and Yu, 2005) presents a ULDA algorithm that diagonalizes $\mathbf{S}_T$ and $\mathbf{S}_B$ separately. Based on our experience, it is slower by a constant factor compared to the GSVD-based version (Howland et al., 2003), which is described in Algorithm 1 (rewritten to suit our needs). However, when the sample size N is large, the SVD decomposition on $\mathbf{K} \in \mathbb{R}^{(J+N)\times M}$ in the line 2 of Algorithm 1 creates a runtime bottleneck. This can be resolved by reducing the dimension of $\mathbf{K}$ before performing SVD (or complete orthogonal decomposition). Since $\mathbf{H}_W$ contributes most of the dimensionality, and the SVD depends on $\mathbf{H}_W^T\mathbf{H}_W$, one possible solution is to replace $\mathbf{H}_W$ with $\mathbf{G}_W \in \mathbb{R}^{M\times M}$, where $\mathbf{H}_W^T\mathbf{H}_W = \mathbf{S}_W = \mathbf{G}_W^T\mathbf{G}_W$. We suggest performing a reduced QR decomposition $\mathbf{H}_W = \mathbf{Q}_W\mathbf{R}_W$ and replacing $\mathbf{H}_W$ with $\mathbf{R}_W$, so that we have

$$\mathbf{H}_W^T\mathbf{H}_W = \mathbf{R}_W^T\mathbf{Q}_W^T\mathbf{Q}_W\mathbf{R}_W = \mathbf{R}_W^T\mathbf{R}_W. \tag{3.9}$$

Park et al. (2007) follows a similar approach, using a Cholesky decomposition $\mathbf{S}_W = \mathbf{C}_W^T\mathbf{C}_W$ and replacing $\mathbf{H}_W$ with $\mathbf{C}_W$. We now use a simulation to evaluate the performance of these two variants and the original LDA/GSVD.

Algorithm 1 ULDA via GSVD (Howland et al., 2003)

Require: Data matrix $\mathbf{X} \in \mathbb{R}^{N\times M}$ with N observations and M features. Class label $\mathbf{y} \in \mathbb{R}^N$ containing J classes

Ensure: Transformation matrix $\mathbf{W} \in \mathbb{R}^{M\times t_2}$ that preserves the class structure

1: Compute $\mathbf{H}_B \in \mathbb{R}^{J\times M}$ and $\mathbf{H}_W \in \mathbb{R}^{N\times M}$ from $\mathbf{X}$ according to equations (3.1).

2: Compute the complete orthogonal decomposition of $\mathbf{K} = \left(\mathbf{H}_B^T, \mathbf{H}_W^T\right)^T \in \mathbb{R}^{(J+N)\times M}$:

$$\mathbf{P}^T\mathbf{K}\mathbf{Q} = \begin{pmatrix}\mathbf{R} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{pmatrix}. \tag{3.10}$$

3: Let $t_1 = \operatorname{rank}(\mathbf{K})$, $t_2 = \operatorname{rank}(\mathbf{P}(1:J, 1:t_1))$. Compute $\mathbf{V}$ from the SVD of $\mathbf{P}(1:J, 1:t_1)$, which satisfies: $\mathbf{U}^T\mathbf{P}(1:J, 1:t_1)\mathbf{V} = \operatorname{diag}(\alpha_1, \alpha_2, \ldots, \alpha_{t_2}, 0, \ldots, 0)$.

4: Compute the first t_2 columns of $\mathbf{Q}\mathbf{R}^{-1}\mathbf{V}$ and assign them to $\mathbf{W}$.

The simulation setup is as follows: in each round, we simulate $N = 10000$ observations from $J = 10$ classes, with each class having the same sampling probability of $1/10$. The features are M mutually independent standard normal noise variables. We then use the three algorithms to calculate the transformation matrix $\mathbf{W}$. We let $M = 2, 4, 8, 16, 32, 64, 128, 256, 512$, and for each M we repeat the simulation 30 times to obtain a confidence band. The results are summarized in Figure 3.2. Their differences in runtime become larger as the number of features increases. For a data matrix of dimension 10000×512, LDA/GSVD with QR decomposition is 32% faster than the original LDA/GSVD implementation (2.6 seconds vs. 3.9 seconds). Consequently, we added this additional QR decomposition step into the LDA/GSVD pipeline.

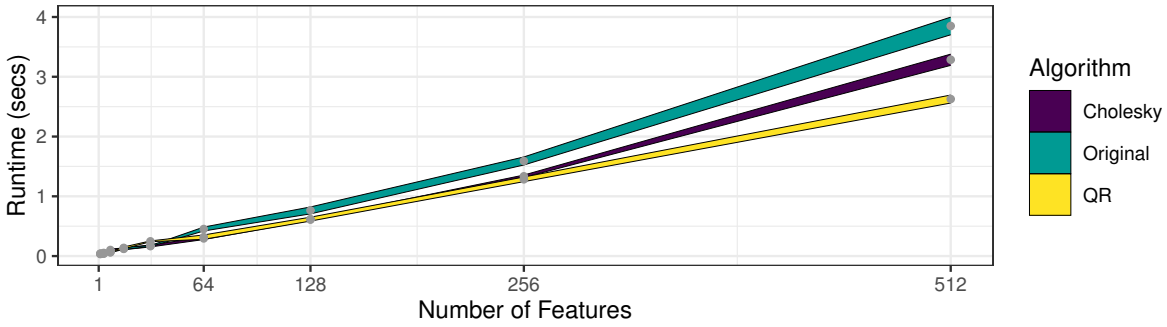

Figure 3.2: The runtime comparisons of three LDA/GSVD implementations (Section 3.1). The data consists of 10,000 observations from 10 classes. The ribbon width represents the 95% confidence interval.

3.2 Node Model: Stepwise LDA

Motivations

Our initial motivation was inspired by the results in Table 3.2 from Section 3.1. We selected LDA/GSVD as our final model due to its superior performance across different types of datasets compared to DLDA. However, in datasets like the Parkinson dataset,

where the number of variables nearly equals the number of observations, LDA/GSVD tends to overfit, whereas DLDA and PCA-LDA perform well. Figure 3.3 illustrates the overfitting patterns on two datasets with a high variable-to-observation ratio, which shows that continuously adding variables does not necessarily improve testing accuracy. For CE-[B]REF_RACE (left), the best testing accuracy is reached with about 12.5% of the variables included, while an even smaller proportion is best for the Parkinson-class dataset. To outperform DLDA and PCA-LDA, we have to find a smart way to rank the variables and select the appropriate number of variables for inclusion in LDA/GSVD, making stepwise LDA a promising solution.

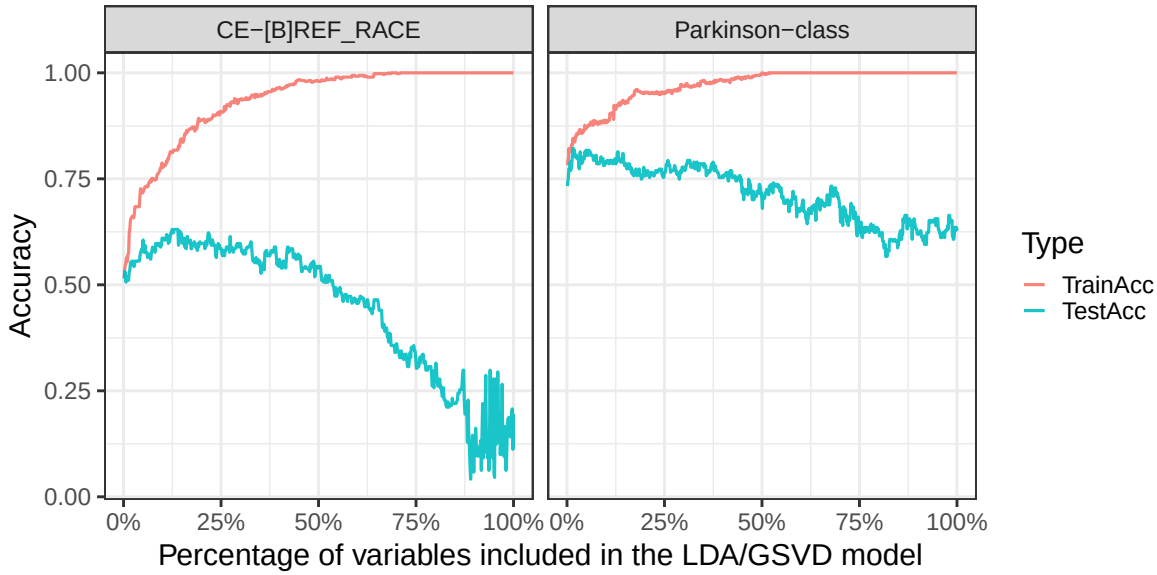

Figure 3.3: Training and testing accuracies when adding more variables. Variables are first ranked based on the GUIDE importance score. Data corresponds to CE-[B]REF_RACE and Parkinson-class detailed in Table A.1.

Beyond addressing overfitting, stepwise LDA offers additional advantages. It can serve as a method for variable selection, eliminating noise variables to enhance the robustness of the LDA fit. Reducing the number of variables also simplifies subsequent interpretations. Another benefit is increased speed. The bottleneck of the LDA/GSVD algorithm is the SVD computation on a matrix with dimensions $(N + J) \times M$, where the time complexity of this SVD is $O\Big((N + J) \times M \times \min (N + J, M) \Big)$. Reducing the

total number of variables M can thus considerably decrease runtime, particularly in large datasets.

Existing Stepwise LDA Framework

Stepwise Discriminant Analysis is a variable selection technique for discriminant analysis that iteratively adds or removes variables based on their statistical significance to enhance model performance. Many versions of Stepwise LDA have been implemented, ranging from stand-alone programs like DIRCRIM (McCabe, 1975) and ALLOC-1 (Hermans and Hobbema, 1976), to options within statistical packages like BMDP (Dixon, 1990), SPSS® program (IBM Corp., 2021), and SAS® program (SAS Institute Inc., 2014). While specific implementations may vary in variable selection criteria or prediction methods, most follow a common framework extensively discussed in (Jennrich, 1977). We will briefly introduce this framework below.

Let $\mathbf{S}_T(1, 2, \ldots, p)$ and $\mathbf{S}_W(1, 2, \ldots, p)$ be the total and within-class scatter matrix with p variables $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p)}\}$ added. Then the Wilks' Λ is defined as:

$$\Lambda(1, 2, \ldots, p) = \frac{\det(\mathbf{S}_W(1, 2, \ldots, p))}{\det(\mathbf{S}_T(1, 2, \ldots, p))}. \tag{3.11}$$

After adding $\mathbf{x}^{(p+1)}$, we use partial Wilks' Λ to evaluate its marginal effect:

$$\Lambda(p + 1) = \frac{\Lambda(1, 2, \ldots, p, p + 1)}{\Lambda(1, 2, \ldots, p)}. \tag{3.12}$$

The null hypothesis H_0 states that the variables $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p+1)}\}$ are from a multivariate normal distribution and are independent of the response $\mathbf{y}$. Unless otherwise specified, this H_0 will be assumed as the null hypothesis throughout the remainder of this paper. Under H_0, the partial F-statistic follows an F-distribution:

$$F = \frac{N - J - p}{J - 1} \frac{1 - \Lambda(p + 1)}{\Lambda(p + 1)} \sim F_{J-1,N-J-p}. \tag{3.13}$$

In the $(p+1)$-th step, partial F-statistics are calculated for the remaining $M-p$ variables, and the variable with the largest F-statistic is selected. It will be included in the model if it meets specific inclusion criteria, such as $F \geq 4$, or if the corresponding p-value is below α.

Following the addition of a variable, the deletion phase begins. With $p+1$ variables now in the model, $p+1$ new pairs of scatter matrices $(\mathbf{S}_{W_i}, \mathbf{S}_{T_i})$ are computed, each excluding one variable $\mathbf{x}^{(i)}$. The partial F-statistics are then calculated for each pair, and the variable associated with the smallest F-statistic is considered for removal if the exclusion criterion is satisfied (e.g., $F < 3.996$ in BMDP). This stepwise process continues until all variables have been added, or no further variables can be added or removed.

Alongside Wilks' Λ-statistic, there are three additional statistics commonly used to evaluate the performance of discriminant analysis or, more precisely, Multivariate Analysis of Variance (MANOVA). While these statistics may not be directly used in the stepwise LDA process, many programs include them in their output, providing valuable insights for statisticians to perform further analysis. To understand the distinctions among these four statistics, recall from Section 3.1 where we discussed LDA/GSVD and noted that GSVD simultaneously decomposes $\mathbf{S}_B$ and $\mathbf{S}_W$, with the singular values $\{\alpha_i^2\}, \{\beta_i^2\}$ outlined in Equation 3.4. Fisher's criterion aims to maximize the ratio between between-class scatter and within-class scatter, essentially maximizing α_i^2/β_i^2. Since $\alpha_i^2 + \beta_i^2 = 1$, this is similar to maximizing α_i^2. The four commonly used statistics, each reflecting different aspects of this discriminative power, are:

- Wilk's Λ: $\Lambda = \prod_{i=1}^{M} \beta_i^2$. A smaller Λ indicates larger discriminative power as it corresponds to smaller β_i^2 and thus larger α_i^2. Wilk's Λ is unique in that it is based on the product of the individual β_i^2 values, in contrast to the summation used by the others.

- Pillai's trace: $V = \sum_{i=1}^{M} \alpha_i^2$, aiming to maximize α_i^2.
- Lawley-Hotelling trace (or Hotelling's generalized T^2-statistic): $U = \sum_{i=1}^{M} \frac{\alpha_i^2}{\beta_i^2}$, emphasizing the ratio $\frac{\alpha_i^2}{\beta_i^2}$ and thus giving more weight to dimensions where α_i^2 is close to 1.
- Roy's largest root: $\theta = \max\left(\frac{\alpha_i^2}{\beta_i^2}\right)$, focusing on the maximum ratio, thus capturing the most discriminative direction.

The stepwise approach enhances forward selection by adding a deletion step, potentially achieving superior results. However, in our simulations, we observed that it seldom deletes variables from the model. When deletions do occur, they tend to result from minor fluctuations rather than the discovery of significant patterns that substantially enhance the model's performance. Additionally, the deletion process is time-consuming. The time complexity is $O\Big((p+1) \times p^3\Big)$ to go from p to $(p+1)$ variables, which becomes increasingly slow as more variables are added to the model. Furthermore, given the usage of stepwise LDA in a decision tree structure, any minor decrease in testing accuracy at a single node can often be compensated for by subsequent splits, reducing concerns about the stand-alone stepwise LDA performance of individual nodes. Consequently, we favor forward selection over the forward-and-backward selection. From now on, whenever stepwise LDA is mentioned in this thesis, it refers specifically to forward LDA.

We are interested in exploring stepwise LDA for its potential to select a subset of variables that might improve model performance. However, one can argue that maximizing testing accuracy and maximizing test statistics might not always coincide. (Rencher and Christensen, 2002) highlights that no actual discriminant scores are computed during the stepwise process, suggesting the term *stepwise MANOVA* might be more appropriate than *stepwise LDA*. Furthermore, (Habbema and Hermans, 1977) notes that variable selection based on Wilks' Λ doesn't necessarily yield a higher

classification accuracy. However, it is unavoidable to use certain types of statistics to rank the variables, as screening all possible subsets is impractical. The four types of statistics in the stepwise LDA process are helpful, and we find them align well with testing accuracies in our simulations. We also tried using a validation set to select the best subset of variables. This approach is time-consuming and involves data splitting, which might reduce effectiveness in building the LDA model compared to using all data in traditional approaches.

Problems with Existing Methods

The first problem is the premature stopping. When perfect linear dependency exists in the data matrix, we would expect $\frac{0}{0}$ on the right-hand side of equation (3.11), causing errors in some stepwise LDA programs, such as `klaR::greedy.wilks` in R. Wilks' Λ is not well-defined under perfect linear dependency, and to allow the stepwise selection to continue, a quick fix is to manually set it to 1, indicating no discrimination power.

We know from equation (3.12) that the partial Λ is the ratio of two Wilks' Λ. Most programs will stop the stepwise LDA process when $\Lambda = 0$, as all subsequent partial Λ calculations become $\frac{0}{0}$ and are thus ill-defined. However, stopping at $\Lambda = 0$ isn't always appropriate, as it indicates that one group of classes is perfectly separable from another, but it doesn't necessarily imply perfect separation of all classes in non-binary classifications, as shown in Figure 3.4. After selecting X_2, Wilks' $\Lambda = 0$ since the within-class variance is zero on X_2, causing the stepwise selection to stop. It successfully separates class A from classes B and C but cannot distinguish class B from class C. Additionally, when multiple variables result in $\Lambda = 0$, only one is selected, leading to the potential waste of useful information contained in the remaining variables.

Secondly, we use an example to demonstrate that the distribution of the partial F-statistic does not follow an F-distribution under the stepwise selection framework, casting doubt on the validity of the associated hypothesis testing. Under H_0, the

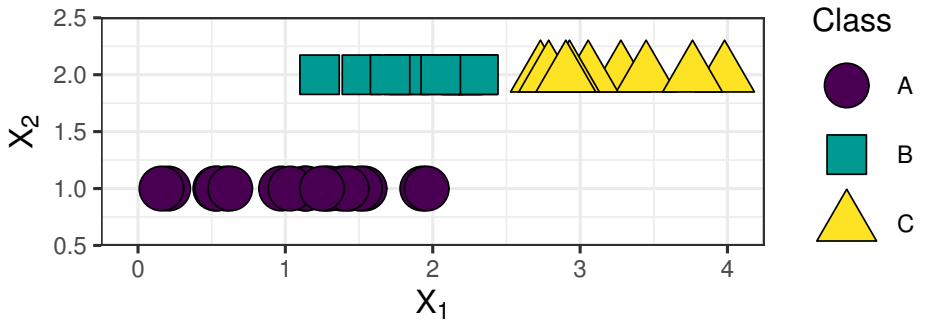

Figure 3.4: A simulated pattern used in Section 3.2. The stepwise LDA based on Wilks' Λ selects X_2 and ignores X_1, leading to suboptimal performance.

partial F-statistic follows an F-distribution (Rencher and Christensen, 2002). However, the original proof assumes that variables are ordered randomly, rather than selected through a stepwise process. Intuitively, if we maximize the F-statistic at each step, the result will be biased, as noted in (Rencher and Larson, 1980).

The simulation setup is as follows: in each round, we simulate $N = 150$ observations from $J = 3$ classes, with each class having the same sampling probability of $1/3$. We simulate X_1 from a standard normal distribution in the one-variable scenario and simulate X_1 and X_2 from independent standard normal distributions in the two-variable scenario. We simulate 10,000 rounds and record the partial F-statistic from each round at each step. We then compare the simulated F-statistic with the theoretical distribution, as summarized in Figure 3.5.

The upper plot corresponds to the one-variable scenario. Since we have only one variable, X_1, no selection occurs, and the simulated distribution matches the theoretical distribution closely. The lower plot shows the two-variable scenario. Here, the stepwise selection first chooses $X^{(1)}$, which has a larger partial F-statistic (and a smaller partial Λ-statistic) compared to the other variable, resulting in an upward bias in the first partial F-statistic. The second partial Λ is the ratio of the overall Wilks' Λ (with two variables) to the first (partial) Wilks' Λ, so its partial Λ-statistic is biased upwards and its partial F-statistic is biased downwards. Note that the theoretical distributions

for the first and second partial F are different ($F_{2,147}$ and $F_{2,146}$), but the difference is negligible, so we assign the same color to both distributions in the plot.

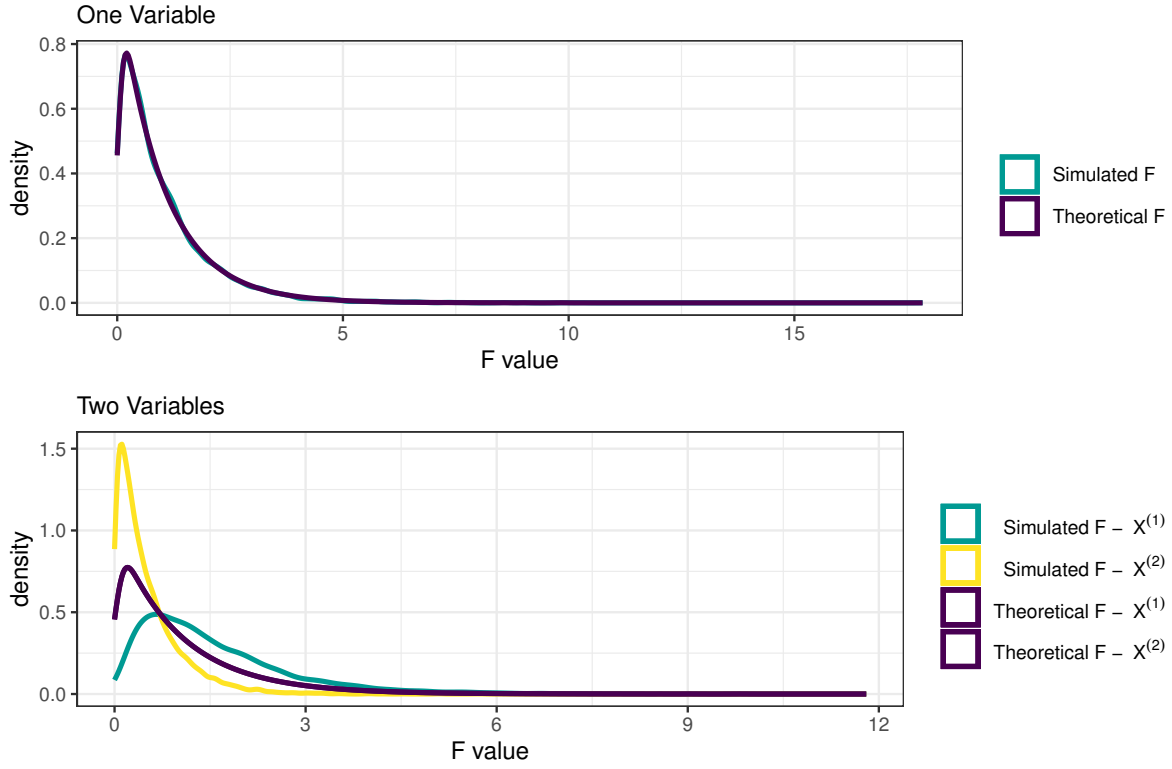

Figure 3.5: The partial F-statistic does not follow an F-distribution under the stepwise selection framework (Section 3.2). When there are two variables (lower plot), the partial F for the first variable $X^{(1)}$ is biased upwards, while the partial F for the second variable $X^{(2)}$ is biased downwards.

The third problem is the inflated Type I error associated with the threshold. In most programs, a fixed threshold of 4 is applied to the partial F-statistic. Another possible criterion is comparing the p-value of the partial F-statistic with the predefined α. However, both methods fail to account for the number of variables screened.

Proposed Improvements in Stepwise LDA

We now derive the distribution of the test statistics used in the proposed stepwise LDA/GSVD framework. Without loss of generality, we assume $\mathbf{S}_T$ is invertible through-

out this section, as redundant columns that cause $\mathbf{S}_T$ to be non-invertible have no discriminative power and can always be removed.

Theorem 3.1. *Pillai's trace is non-decreasing when new variables are added to the model.*

Proof. Suppose $\mathbf{X} \in \mathbb{R}^{N\times K}$ have been included in the model, and the new variable to be added is $\mathbf{z} \in \mathbb{R}^N$. The new between-class and total scatter matrices with $(K+1)$ variables can be written as:

$$\mathbf{S}_B = \begin{pmatrix} \mathbf{B}_x & \mathbf{b}_x \\ \mathbf{b}_x' & b_z \end{pmatrix} \quad \mathbf{S}_T = \begin{pmatrix} \mathbf{T}_x & \mathbf{t}_x \\ \mathbf{t}_x' & t_z \end{pmatrix}, \tag{3.14}$$

where $\mathbf{B}_x$ and $\mathbf{T}_x$ are the previous between-class and total scatter matrices for $\mathbf{X}$. If $K = 0$, the difference in Pillai's trace will be b_z/t_z. This value is non-negative since the between-class scatter matrix is positive semi-definite ($b_z \geq 0$) and the total scatter matrix is positive definite ($t_z > 0$). For $K \geq 1$, we aim to show

$$\operatorname{trace}\left(\mathbf{S}_T^{-1}\mathbf{S}_B\right) - \operatorname{trace}\left(\mathbf{T}_x^{-1}\mathbf{B}_x\right) \geq 0. \tag{3.15}$$

Since $\mathbf{S}_T$ and $\mathbf{T}_x$ are invertible, we have

$$\mathbf{S}_T^{-1} = \begin{pmatrix} \mathbf{T}_x^{-1} + \mathbf{T}_x^{-1}\mathbf{t}_x\mathbf{t}_x'\mathbf{T}_x^{-1}(t_z - \mathbf{t}_x'\mathbf{T}_x^{-1}\mathbf{t}_x)^{-1} & -\mathbf{T}_x^{-1}\mathbf{t}_x(t_z - \mathbf{t}_x'\mathbf{T}_x^{-1}\mathbf{t}_x)^{-1} \\ -(t_z - \mathbf{t}_x'\mathbf{T}_x^{-1}\mathbf{t}_x)^{-1}\mathbf{t}_x'\mathbf{T}_x^{-1} & (t_z - \mathbf{t}_x'\mathbf{T}_x^{-1}\mathbf{t}_x)^{-1} \end{pmatrix}. \tag{3.16}$$

According to the block matrix multiplication and the properties of the trace, we have

$$\begin{aligned}\operatorname{trace}\left(\mathbf{S}_T^{-1}\mathbf{S}_B\right) &= \operatorname{trace}\left(\mathbf{T}_x^{-1}\mathbf{B}_x\right) \\ &\quad + (\mathbf{t}_x'\mathbf{T}_x^{-1}, -1)\mathbf{S}_B(\mathbf{t}_x'\mathbf{T}_x^{-1}, -1)' \\ &\quad \times (t_z - \mathbf{t}_x'\mathbf{T}_x^{-1}\mathbf{t}_x)^{-1}. \end{aligned} \tag{3.17}$$

The Schur complement of the block $\mathbf{T}_x$ in the matrix $\mathbf{S}_T$ is given by $(t_z - \mathbf{t}_x'\mathbf{T}_x^{-1}\mathbf{t}_x)^{-1}$. Since $\mathbf{T}_x$ and $\mathbf{S}_T$ are both positive definite, we have $(t_z - \mathbf{t}_x'\mathbf{T}_x^{-1}\mathbf{t}_x)^{-1} > 0$. The quadratic form $(\mathbf{t}_x'\mathbf{T}_x^{-1}, -1)\mathbf{S}_B(\mathbf{t}_x'\mathbf{T}_x^{-1}, -1)'$ is also non-negative because the middle matrix is positive semi-definite. □

At its core, LDA/GSVD seeks to maximize Pillai's trace $V = \operatorname{trace}\left(\mathbf{S}_T^{-1}\mathbf{S}_B\right)$. According to Theorem 3.1, with each variable added, the current Pillai's trace increases (or remains the same). Let $V^{(M)}$ denote the Pillai's trace with all M variables included. The goal of stepwise selection is to approximate $V^{(M)}$ using $V^{(K)}$, where $K \ll M$.

Suppose the variable set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{K-1}\}$ has been selected after the first $(K-1)$ steps, and the Pillai's trace of that variable set is $V_{\max}^{(K-1)}$. Here, the subscript indicates that this Pillai's trace is not of $(K-1)$ randomly selected variables but is instead maximized at each step through stepwise forward selection. At step K, we calculate $V_{(1)}^{(K)}, V_{2}^{(K)}, \ldots, V_{(M-K+1)}^{(K)}$, where $V_{(i)}^{(K)}$ denotes the Pillai's trace of the variable set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{K-1}, \mathbf{x}_{(i)}\}$. Let $k = \arg\max_i V_{(i)}^{(K)}$. We then select $\mathbf{x}_{(k)}$ as the best candidate at step K, and $V_{\max}^{(K)} = V_{(k)}^{(K)}$. To establish an inclusion criterion, we must measure the marginal effect of the added variable $\mathbf{x}_{(k)}$, which corresponds to $V_{\max}^{(K)} - V_{\max}^{(K-1)}$.

Theorem 3.2. *At step K, let t_K be the $(1-\alpha)^{1/(M-K+1)}$ quantile of $\mathrm{B}\left(\frac{J-1}{2}, \frac{N-J}{2}\right)$. Then, $P(V_{\max}^{(K)} - V_{\max}^{(K-1)} \geq t_K) \leq \alpha$ as $N \to \infty$ under H_0, where the newly added variable $\mathbf{z}$ is normally distributed and independent of both $\mathbf{X}$ and $\mathbf{y}$.*

Proof. When $K = 1$, $V^{(1)}$ under H_0 follows a beta distribution $\mathrm{B}(\frac{J-1}{2}, \frac{N-J}{2})$ (Pillai, 1955).

Since $V_{\max}^{(1)} - V_{\max}^{(0)} = V_{\max}^{(1)}$ is the maximum of $V_1^{(1)}, V_2^{(1)}, \ldots, V_M^{(1)}$, which are M i.i.d. random variables from the beta distribution, its CDF can be written as $I_x^M(\frac{J-1}{2}, \frac{N-J}{2})$ where $I_x(\frac{J-1}{2}, \frac{N-J}{2})$ is the CDF of $\mathrm{B}(\frac{J-1}{2}, \frac{N-J}{2})$. To control the type I error below α, the threshold t must satisfy $I_t^M(\frac{J-1}{2}, \frac{N-J}{2}) \leq 1-\alpha$, which is equivalent to $I_t(\frac{J-1}{2}, \frac{N-J}{2}) \leq (1-\alpha)^{1/M}$. Then t is the $(1-\alpha)^{1/M}$ quantile of $\mathrm{B}\left(\frac{J-1}{2}, \frac{N-J}{2}\right)$.

When $K > 1$, from equation (3.17) we know that

$$\begin{aligned} V_{(i)}^{(K)} - V_{\max}^{(K-1)} &= (\mathbf{t}_x' \mathbf{T}_x^{-1}, -1)\mathbf{S}_B(\mathbf{t}_x' \mathbf{T}_x^{-1}, -1)' \\ &\quad \times (t_z - \mathbf{t}_x' \mathbf{T}_x^{-1} \mathbf{t}_x)^{-1}. \end{aligned} \tag{3.18}$$

This equation still holds if we replace $\mathbf{S}_B$ and $\mathbf{S}_T$ with $\mathbf{S}_B/(N-J)$ and $\mathbf{S}_T/(N-J)$, which are the least squares estimators of the between-class and total covariance matrices. Since $\mathbf{X}$ and $\mathbf{z}$ are independent, their covariance $\mathbf{t}_x \to \mathbf{0}$ as $N \to \infty$. Note that $\mathbf{S}_B/(N-J)$ and $\mathbf{S}_T/(N-J)$ are finite as $N \to \infty$. Substituting $\mathbf{t}_x = \mathbf{0}$ into equation (3.18), we get

$$\begin{aligned} V_{(i)}^{(K)} - V_{\max}^{(K-1)} &= (\mathbf{0}, -1)\mathbf{S}_B(\mathbf{0}, -1)'(t_z - 0)^{-1} \\ &= b_z/t_z \end{aligned} \tag{3.19}$$

b_z/t_z is Pillai's trace for $\mathbf{z}$. Therefore, the distribution of $V_{(i)}^{(K)} - V_{\max}^{(K-1)}$ can be approximated by $V^{(1)}$, and the rest follows the scenario where $K = 1$. □

Based on our experience, this asymptotic approximation sometimes leads to a very conservative threshold, with the type I error falling well below the predefined α. Therefore, we introduce an empirical approximation to mitigate this problem and

achieve higher power. Suppose we have already added $K-1$ variables and the current Pillai's trace is $V_{\max}^{(K-1)}$. Since the maximum Pillai's trace for J classes is $J-1$, the maximum Pillai's trace that can be added is bounded by $J-1-V_{\max}^{(K-1)}$, which can be viewed as the maximum Pillai's trace for a classification problem with $J-V_{\max}^{(K-1)}$ classes. Thus, at the k-th step, the threshold becomes the quantile from $\mathrm{B}\left(\frac{J'-1}{2}, \frac{N-J'}{2}\right)$ instead of $\mathrm{B}\left(\frac{J-1}{2}, \frac{N-J}{2}\right)$, where $J' = J - V_{\max}^{(K-1)}$.

Next, we analyze the type I error under the stepwise LDA/GSVD framework and demonstrate that the family-wise error rate is controlled at the nominal level α. Suppose we have M variables in total, some of which are noise variables ($\mathbf{x} \in S_n$) and some are informative ($\mathbf{x} \in S_i$). At each step, there are three possible outcomes: a noise variable is selected, the selection stops, or an informative variable is selected. The entire process is illustrated in Figure 3.6.

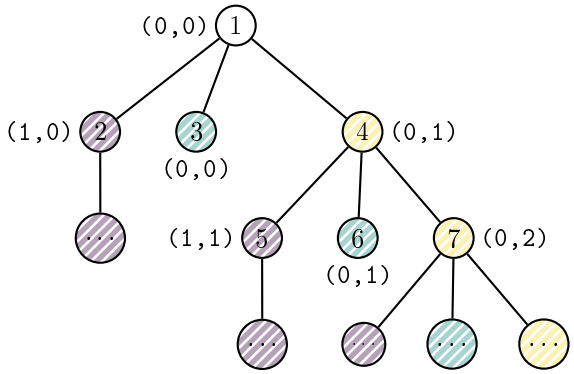

Figure 3.6: Illustration of stepwise forward selection (Section 3.2). (n_n, n_i) next to each node represents the number of noise and informative variables included so far. A yellow background indicates an informative variable being selected, a green background indicates the selection stops, and a purple background indicates that at least one type I error is made.

Suppose $\mathbf{x}$ is the variable with the largest Pillai's trace and is selected at the K-th step. Conditional on whether $\mathbf{x} \in S_n$ or $\mathbf{x} \in S_i$, there are four possible outcomes:

$$
\begin{aligned}
p_{K1} &= P(\mathbf{x} \text{ is added} \mid \mathbf{x} \in S_n) \\
p_{K2} &= P(\mathbf{x} \text{ is not added} \mid \mathbf{x} \in S_n) \\
p_{K3} &= P(\mathbf{x} \text{ is added} \mid \mathbf{x} \in S_i) \\
p_{K4} &= P(\mathbf{x} \text{ is not added} \mid \mathbf{x} \in S_i)
\end{aligned} \tag{3.20}
$$

In situations with p_{K2} and p_{K4}, $\mathbf{x}$ is not added, and stepwise selection stops. Therefore, no type I error is made or will be made. This corresponds to the green regions in Figure 3.6. For the situation with p_{K3}, since an informative variable is added, no type I error is made at the current step, corresponding to the yellow regions in Figure 3.6. The purple region in Figure 3.6 reflects scenarios where a type I error is made, with p_{K1} being the only situation that results in such an error. Theorem 3.2 shows that under H_0, $p_{K1} \leq \alpha$, meaning that at each step, the probability of branching into the purple region is controlled at α. Now, we aim to show that, overall, the probability of ending up in any purple region is controlled at α.

The probability of reaching node 2 is $p_{11} \leq \alpha$. The probability of reaching node 5 is $p_{13} \times p_{21} \leq p_{21} \leq \alpha$. The reason we can use the product of p_{13} and p_{21} to calculate this probability is that under H_0, the variable selected in the first step is assumed to be independent of the variable selected in the second step. For nodes like node 2 and node 5, where the first noise variable is added in the current step, the probability of reaching them can be written as

$$
p_{K1} \prod_{k=1}^{K-1} p_{k3} \leq p_{K1} \leq \alpha. \tag{3.21}
$$

Meanwhile, the probability of reaching their child nodes is also controlled at α, because reaching these nodes requires first reaching their parent node. All purple nodes fall into one of these two scenarios, so the family-wise type I error rate is controlled at

α. This means that if the stepwise LDA/GSVD selects K variables $\{\mathbf{x}_{(1)}, \mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(K)}\}$, then the probability that at least one $\mathbf{x}_{(i)}$ is a noise variable is controlled at α.

The stepwise forward selection framework is summarized in Algorithm 2.

Algorithm 2 Proposed stepwise selection based on Pillai's trace

Require: Data matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$ with N observations and M features. Class label vector $\mathbf{y} \in \mathbb{R}^N$ containing J classes
Ensure: A reduced data matrix $\mathbf{X}_{\text{new}} \in \mathbb{R}^{N \times K}$ where $K \leq M$

index_in $\Leftarrow \{\}$
index_pool $\Leftarrow \{1, 2, \ldots, M\}$
previous_pillai $\Leftarrow 0$
while index_pool is not empty **do**
    $l \Leftarrow$ length(index_pool)
    for $i \Leftarrow 1$ to l **do**
        pillai_saved[i] $\Leftarrow$ `get_pillai`($\mathbf{X}$, $\mathbf{y}$, (index_in, index_pool[i])) ▷ Calculate Pillai's trace for each feature
    end for
    i_selected $\Leftarrow \arg\max_i$ pillai_saved[i]
    $J' = J -$ previous_pillai
    threshold $\Leftarrow I^{-1}_{(1-\alpha)^{1/l}}(\frac{J'-1}{2}, \frac{N-J'}{2})$
    if pillai_saved[i_selected] - previous_pillai $\leq$ threshold **then**
        break ▷ Stop if the improvement is below threshold
    else
        index_selected $\Leftarrow$ index_pool[i_selected]
        index_in $\Leftarrow$ (index_in, index_selected)
        index_pool $\Leftarrow$ index_pool $\setminus$ {index_selected}
        previous_pillai $\Leftarrow$ pillai_saved[i_selected]
    end if
end while
if length(index_in) $= 0$ **then**
    $\mathbf{X}_{\text{new}} \Leftarrow \mathbf{X}$ ▷ No variable is significant, return all variables
else
    $\mathbf{X}_{\text{new}} \Leftarrow \mathbf{X}$(:, index_in)
end if

Simulation

In this section, we use simulation to showcase the performance of three forward LDA variants:

1. `Pillai`: the proposed variant using Pillai's trace (see Algorithm 2).

2. `Wilks`: the original variant based on Wilks' Λ. The inclusion criterion is based on p-value, with a variable included if the p-value is below the predefined α.

3. `Wilks-Bonferroni`: This variant applies an additional Bonferroni correction to the p-value compared to the second variant. If there are $(M - K + 1)$ variables to choose from at the K-th step, the p-value is multiplied by $(M - K + 1)$ to adjust for the multiple testing.

Note that these variants are for selection alone. They help check the type I error and power (whether the desired variables are included). To further compare the testing accuracy, we apply LDA/GSVD to the selected variables.

First, we analyze the type I error rate. We use the same simulation settings from Section 3.2 to compare the three forward LDA variants, and the results are summarized in Figure 3.7. `Pillai` and `Wilks-Bonferroni` successfully control the type I error in both scenarios. In contrast, `Wilks` suffers from an inflated type I error rate due to multiple testing. These results validate Theorem 3.2, demonstrating that the type I error rate is well-controlled under H_0, where the noise variables are normally distributed and independent of both the informative variables and the response.

Secondly, we illustrate the primary advantage of our proposed method over the original framework: its ability to handle scenarios where Wilks' $\Lambda = 0$. We use a simulated dataset, which contains 2,000 observations. The response variable is randomly selected from 10 classes, each with an equal probability of $1/10$. We then create a dummy matrix (one-hot encoding) of the response, resulting in 10 columns, each consisting of 1s or 0s. These 10 columns are used as our features. Ideally, these 10 features can perfectly predict the response, and a robust forward selection method should select them all. However, `Wilks` stops after selecting only one feature, $I_{\text{Class One}}$, the indicator of the first class. Here, $I_{\text{Class One}} = 1$ for observations from class one and

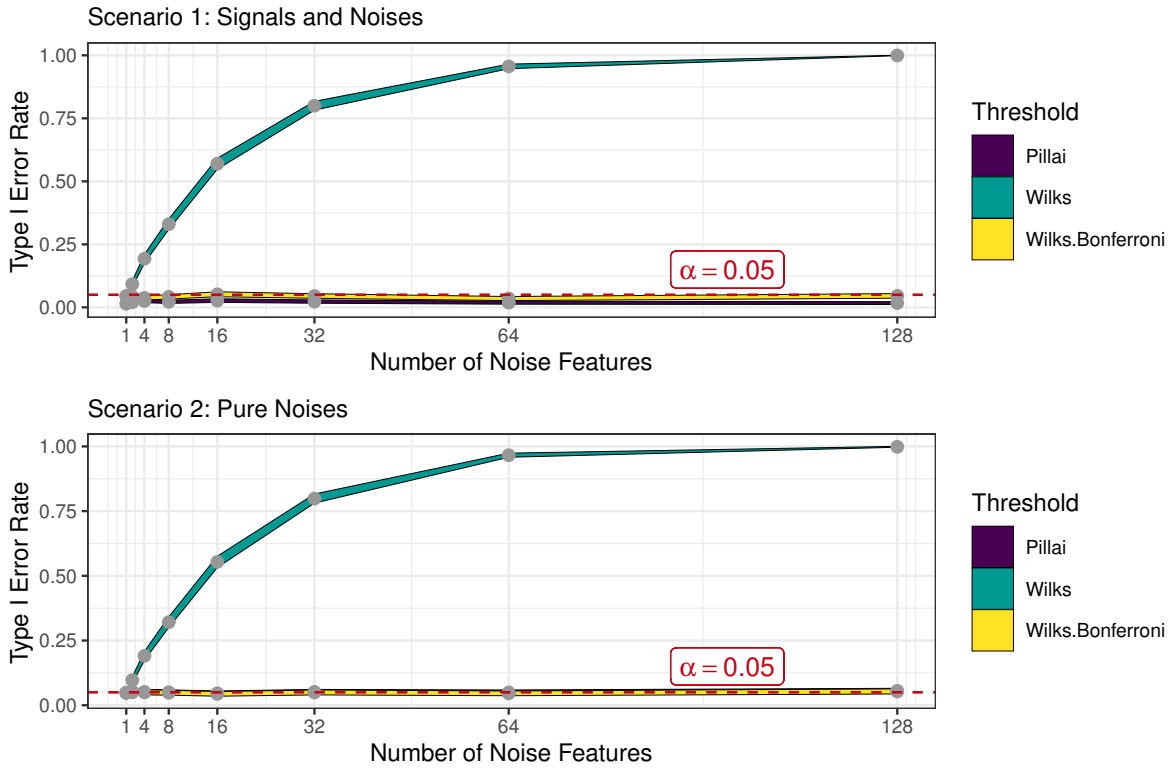

Figure 3.7: Type I error rate from three forward LDA variants on the iris dataset (Section 3.2). The ribbon width represents the 95% confidence interval. The ribbon for `Pillai` overlaps with the ribbon for `Wilks-Bonferroni` in the lower plot.

$I_{\text{Class One}} = 0$ for the other classes. Therefore, the within-class variance is 0, leading to Wilks' $\Lambda = 0$. Meanwhile, with `Pillai`, the feature $I_{\text{Class One}}$ contributes a value of 1 to the overall Pillai's trace, which does not trigger a stop. `Pillai` continues adding features, with each feature $I_{\text{Class i}}$ contributing a value of 1 to the overall Pillai's trace. It selects 9 features and then stops, as the maximum Pillai's trace of $J - 1 = 9$ is reached. With these 9 features, we have enough information to perfectly predict the response.

Real Data Analysis

We tested the stepwise LDA on the two datasets mentioned in Section 3.2, and the results are summarized in Table 3.3. As seen, when LDA/GSVD tends to overfit, stepwise LDA can help it recover, often outperforming the DLDA variant, which is considered effective in these scenarios.

dataset	method	trainAcc	testAcc
CE-[B]REF_RACE	DLDA	0.702 (0.009)	0.575 (0.011)
CE-[B]REF_RACE	LDA/GSVD	0.999 (0.001)	0.439 (0.018)
CE-[B]REF_RACE	stepLDA	0.783 (0.01)	**0.614 (0.011)**
Parkinson-class	DLDA	0.819 (0.005)	0.834 (0.012)
Parkinson-class	LDA/GSVD	1 (0)	0.66 (0.021)
Parkinson-class	stepLDA	0.853 (0.006)	**0.844 (0.015)**

Table 3.3: Results of three LDA variants in Section 3.2. Two standard deviations are recorded in parentheses. Data corresponds to CE-[B]REF_RACE and Parkinson-class detailed in Table A.1.

However, stepwise LDA does not always outperform LDA/GSVD. We assessed stepwise LDA on the same 49 datasets mentioned in Section 3.1 and included its performance in the comparison. According to Figure 3.8, on average, stepwise LDA and LDA/GSVD show similar testing accuracies across different datasets. The inferior performance of stepwise LDA compared to LDA/GSVD on some datasets can be attributed to its one-variable-at-a-time approach, which can be shortsighted and miss out on detecting multiple weak signals that work together. Furthermore, stepwise LDA is slower than LDA/GSVD, as the selection process becomes time consuming with an increasing number of variables.

Stepwise LDA is inherently effective at distinguishing signal from noise. To further illustrate this advantage, we selected four small datasets, each with fewer than 1000 observations and less than 10 variables. We then added 500 Gaussian noise variables to each dataset and assessed the impact on their performance. The results are summarized in Table 3.4. It is obvious that the testing accuracies of DLDA and LDA/GSVD are significantly impacted by the added noise, showing decreases ranging from 30.9% to 58.2%. In contrast, stepwise LDA remains relatively stable, with differences in testing accuracies before and after adding noise variables all being less than 10.4%. In our simulations, this pattern is consistent across multiple datasets.

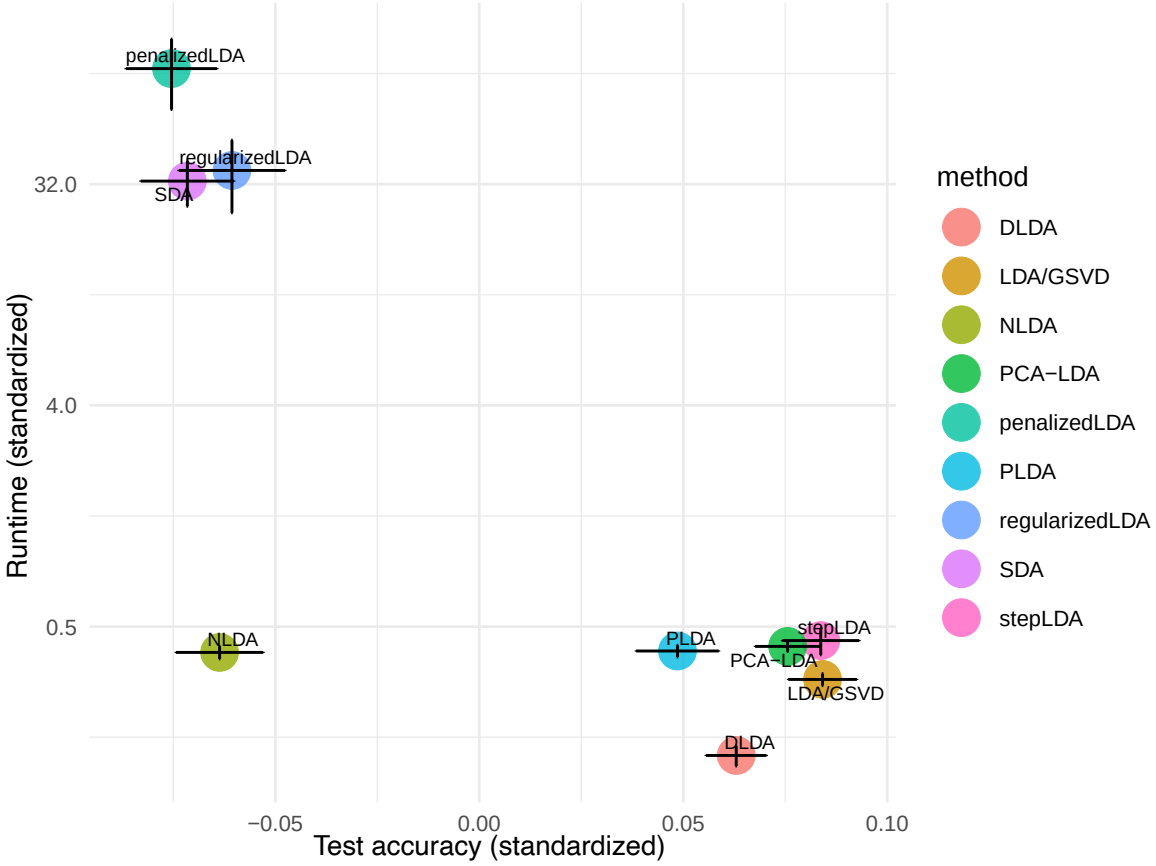

Figure 3.8: Real data analysis results in Section 3.2: Average runtime vs. average testing accuracy (stepLDA added). Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

		iris-Species	fishcatch-Species	Vowel-y	Digits-Digit
DLDA	Before	0.926 (0.022)	0.808 (0.025)	0.593 (0.014)	0.718 (0.023)
	After	0.616 (0.032)	0.391 (0.031)	0.235 (0.015)	0.192 (0.017)
	Diff	-0.309 (0.053)	-0.416 (0.056)	-0.357 (0.028)	-0.526 (0.04)
LDA/GSVD	Before	0.975 (0.008)	0.965 (0.019)	0.593 (0.014)	0.718 (0.023)
	After	0.603 (0.044)	0.383 (0.028)	0.223 (0.012)	0.177 (0.022)
	Diff	-0.372 (0.052)	-0.582 (0.047)	-0.37 (0.026)	-0.541 (0.045)
stepLDA	Before	0.958 (0.014)	0.971 (0.016)	0.598 (0.013)	0.722 (0.022)
	After	0.938 (0.015)	0.975 (0.009)	0.494 (0.017)	0.685 (0.026)
	Diff	**-0.02 (0.028)**	**0.004 (0.025)**	**-0.104 (0.029)**	**-0.038 (0.048)**

Table 3.4: Testing accuracies of three LDA variants in Section 3.2 after adding 500 Gaussian noise variables. Two standard deviations are recorded in parentheses. Detailed data descriptions can be found in Table A.1.

Discussion and Conclusion

Based on the analysis results, LDA/GSVD and stepwise LDA have been selected as the best LDA variants to serve as node models in the subsequent decision tree. Next, we discuss some failed attempts to help future researchers. Stepwise LDA selects a subset of variables from the original data. We also tried variable selection in the transformed space $\mathbf{Xw}$ (discriminant scores) but found it ineffective. The reason might be that $\mathbf{Xw}$ has already excluded the effect of noise variables, making de-noising in subsequent selection less effective. Additionally, the final discriminant scores have at most $J-1$ dimensions, leaving not much room for subsequent variable selection, especially in binary classification where there is only one discriminant score.

We also note some instability in R during our implementation of stepwise LDA. Calculating Wilks' Λ involves determinants of the scatter matrices. But different implementations will return different determinants, especially for nearly singular matrices. Although the condition number of a matrix can indicate singularity, calculating it via `kappa` is very time consuming, sometimes even more than a complete QR decomposition. Fortunately, Pillai's trace doesn't require determinant calculation. In Section 3.1, we noted that scaling can improve robustness. After finding the discriminant direction $\mathbf{w}$ on the scaled data, we have two options for prediction. We can either transform $\mathbf{w}$ back to its original scale and apply it to the testing data, or we can scale the testing data and use $\mathbf{w}$ directly. Theoretically, both methods should yield identical results, but in practice, they often differ, sometimes significantly, with discriminant scores differing by as much as e^{40}. For now, we decided to use the second method as it proved to be more accurate.

3.3 Stopping Rule

Decision trees work by recursively dividing the sample space into smaller subspaces and creating different decision rules for each subspace. This process of splitting cannot continue indefinitely due to the finite sample size and variable space. Therefore, it's crucial to establish a stopping rule for the tree-generating process. This rule ensures that the tree is not overly complex and doesn't just work well for the data it was trained on, but can also make good predictions for new data. By specifying an appropriate stopping rule, we effectively select the right-size tree from numerous possible candidates. Therefore, the stopping rule can be viewed as the model selection tool for the decision tree.

Note that forest methods, either using bagging or bootstrap, typically have less strict stopping rules. They continue splitting until a node becomes trivial, such as when only one class remains or there are no variables left to split. In forest methods, the regularization effect is implicitly achieved through the ensemble of trees. However, since our focus is on developing a single-tree algorithm, we will focus on more sophisticated stopping rules that better satisfy our needs. All simulation results in this section are based on the LDA Splitting described in Section 3.4.

Existing Stopping Rules

The stopping rule in decision trees is also referred to as pruning, which can be categorized into two main types: pre-pruning and post-pruning. Pre-pruning involves stopping the tree's growth early to prevent overcomplexity, whereas post-pruning removes branches from a fully grown tree. Below are some commonly used methods from both categories. For a more detailed discussion, please refer to (Mingers, 1989; Esposito et al., 1997).

Reduced Error Pruning (Quinlan, 1987) This technique requires a separate valida-

tion set. In this method, after growing the tree to its full size, we start at the bottom and, at every intermediate node, attempt to cut its branches and turn it into a terminal node that predicts all instances to the plurality class. If such pruning reduces the classification errors on the validation set, we then keep this pruning, hence the name *Reduced Error Pruning*.

Pessimistic Error Pruning (Quinlan, 1987) This technique has an advantage over Reduced Error Pruning as it does not require a separate validation set, allowing for the efficient use of all available samples for training. However, since the resubstitution error (the testing error when the test set is the same as the training set) tends to optimistically underestimate the true classification error, two modifications are made. Firstly, every node incurs a cost of $1/2$ in the overall classification error, thereby penalizing model complexity. Secondly, a split is kept not just when it has a lower resubstitution error, but it must also be at least 1SD below that error. The SD is calculated based on the binomial assumption of the resubstitution error.

Minimum Error Pruning (Niblett and Bratko, 1987) This technique focuses on minimizing the expected error rate, using the entire training set for both building and pruning the tree. The expected error rate at any node is calculated as one minus a modified incidence rate of the majority class. This modified incidence rate is estimated through a weighted average of the observed proportion and its prior probability of each class. The weight is predefined and assumed to be consistent across all nodes. In the pruning process, a branch is pruned from a specific node if it leads to an increase in the expected error rate.

Critical Value Pruning (Mingers, 1987) This method provides a flexible framework for a range of pruning methods, each defined by a unique test statistic to evaluate the strength of splits in the tree. The possible choice of the test statistic can be

the chi-squared statistic from the contingency table or the Gini Impurity. The pruning process works by examining the tree from the bottom up and pruning branches that fail to meet a specified threshold.

Occasionally, a situation arises where a node has a small test statistic, but one of its child nodes exceeds the threshold with a higher statistic. In such cases, this method will keep the parent node, thereby implicitly enforcing monotonicity throughout the tree. Typically, this method involves testing many thresholds, which creates a series of trees of different sizes. Notably, the larger the threshold, the smaller the resulting tree. The final step involves selecting the best tree from these candidates; e.g., a separate validation set can be used.

Cost-Complexity Pruning (Breiman et al., 1984) This method, like Pessimistic Error Pruning, considers the number of child nodes in a split and penalizes a split that results in many child nodes with little improvement in training accuracy. More specifically, let $R(t)$ be the resubstitution error at the current node, and $R(T_t)$ the resubstitution error of the subtree rooted at this node. Let $|\widetilde{T_t}|$ denote the number of terminal nodes in the subtree T_t, then the measure α of the current split is defined as follows:

$$\alpha = \frac{R(t) - R(T_t)}{|\widetilde{T_t}| - 1} \tag{3.22}$$

This measure effectively estimates the reduction in resubstitution error per terminal node, which is positively related to the strength of a split. After calculating α for all intermediate nodes, pruning begins from the smallest to the largest α, pruning the tree from its full size to a single root node. And more importantly, each α in the range $[\min(\alpha), \max(\alpha)]$ corresponds to a specific tree in the pruning process.

After we get a series of trees, this method uses a smarter approach than Critical Value Pruning for selecting the best tree. It divides the training set into K folds and applying K-fold cross-validation. For the i-th model, the i-th fold is excluded, and a series of trees are built using the remaining $K - 1$ folds. The i-th fold then serves as the testing set, and the testing error for each tree in the series is calculated, with each error corresponding to a specific α. The next step involves combining information from all K folds to select the best α based on the smallest average testing errors across all folds. Finally, the original tree is pruned using the best α obtained from the cross-validation.

Problems with Cost-Complexity Pruning

Implemented in CART, Cost-Complexity Pruning is among the most popular methods, renowned for its promising theoretical properties and strong practical performance. However, when attempting to apply this method in our research, we found several drawbacks, indicating that it might not be the ideal approach for our needs. In Cost-Complexity Pruning, α plays two critical roles:

1. **Determining the Pruning Order**: Consider a full-size binary tree with three levels (8 terminal nodes). To prune this tree back to its root node, 7 cuts are necessary. Arranging these cuts in different ways results in varied cutting paths. In this example, there are 80 potential cutting paths leading to 26 unique subtrees. We can view α as indicating the strength of a split. With the help of α, we can pick the best path from these 80 possibilities, ideally resulting in a path with a monotone increasing split strength from bottom to top. Along this path, we get 7 subtrees and then use Cross-Validation (CV) to pick the most accurate one.

2. **Establishing a Mapping Rule**: Given that our tree is constructed using all the data, an external referee is necessary to select the best subtree. In Cost-Complexity

Pruning, this referee is provided by cross-validation (CV). We assume the α values in the main and CV trees are consistent, meaning they both reflect the same split strength. The best empirical cutting α is chosen using the CV trees, and subsequently, this α value is applied to prune the main tree.

There are issues with both outlined objectives. For the first, the concern is relatively minor. Consistently removing the weakest link does not always guarantee the most accurate subtree. Moreover, for decision trees with a non-trivial node model, the α value might not necessarily be increasing monotonically from bottom to top, which brings us to the second objective.

There's an underlying assumption with the α link: its behavior should be consistent between CV trees and the main tree. Since we prune the CV tree and the main tree using the same α, if that α gives us a big CV tree, then the main tree corresponding to this α should be big as well. This association becomes meaningless when the α is non-monotonic; a bigger main tree may correspond to a smaller CV tree, while a smaller main tree may correspond to a bigger CV tree. Note that once some nodes are pruned, the α for the remaining parts of the tree will change, and that is why we have non-increasing α (from bottom to top) in the main tree.

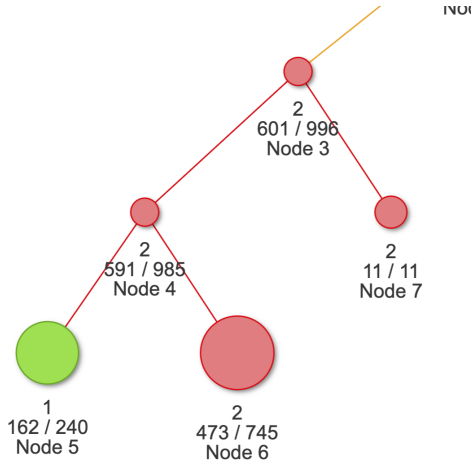

Figure 3.9: A partial plot from LDATree, used in Section 3.3. The number below each node shows the ratio of correctly classified training samples vs. total node size.

treeeNo	nodeCount	meanMSE	seMSE	α	testAcc
0	31	414.1	5.757	3.464	0.429
1	25	413.5	5.706	12.49	0.429
2	23	413.6	5.546	10.392	0.429
3	21	413.4	5.564	27.495	0.429
4	19	412.1	5.604	18.166	0.429
5	17	412.1	5.604	40.89	0.429
6	15	412.4	5.327	44.497	0.43
7	13	412.2	5.672	7.483	0.431
8	11	412.2	5.672	63.937	0.431
9	9	412.3	5.489	73.498	0.431
10	7	411.8	5.475	60.374	0.436
11	5	411.8	5.475	95.247	0.443
12	3	413.8	5.948	143	0.444
13	1	419.2	6.135	NA	0.424

Table 3.5: The CVpruning table from LDATree using NHDS-discharge.status data in Section 3.3. Detailed data descriptions can be found in Table A.1.

Here is an example showing that the α might not be monotonically increasing using the NHDS-discharge.status data (see Table A.1). Table 3.5 provides the pruning summary. The α value in the k-th row indicates the α required to prune the tree from the k-th row to the $(k+1)$-th row. Ideally, the α column should show a monotonic increase from the top to the bottom. Yet, there's a pronounced discrepancy between the 6th and 7th trees. The corresponding cuts for this gap can be seen in nodes 4 and 3, as shown in Figure 3.9. Node 4 represents a strong split since the count of correctly classified samples rises from 591 to $162 + 473 = 635$, leading to an $\alpha = 44.497$. Once this node is pruned, node 3 becomes the next candidate. It's a weak split, with a slight increase from 601 to $591 + 11 = 602$. Such non-monotonic behavior can make the mapping confusing and meaningless.

Proposed Stopping Rules

Based on previous works, we added our own insights and identified two potential pre-pruning candidates, detailed below:

Method 1: Validation Set Our first modification involves using the α from the Cost-Complexity Pruning but replacing the resubstitution error with the validation error. Many traditional methods rely on resubstitution error or its variants to measure the performance of a split. Doing so will have low overfitting risks only with trivial node models. Particularly in nodes with data from multiple classes, prediction by majority vote is less prone to overfitting. However, in the case of an LDA model, due to its strong discriminant capability, overfitting can become a significant problem, making resubstitution error a less reliable indicator of split efficacy. Using a validation set offers a more objective estimate. Although reserving part of the data for validation results in a smaller training set, this is less of an issue for LDA, which primarily depends on class centroids and covariance matrix estimates. Additional data points may not substantially enhance these estimates, making the exclusion of some data more acceptable in this context.

Another adaptation is the introduction of a *kStepAhead* hyperparameter, enabling lookahead capability. A key limitation of pre-pruning compared to post-pruning is its potential to overlook higher-order interactions due to its stringent inclusion of new splits. A beneficial split may follow one or two weaker splits, a pattern typically detected only in post-pruning. Allowing the stopping rule to look ahead beyond a hard threshold might address this issue.

Our simulation results indicate that the train-validation ratio significantly influences model performance, meaning careful tuning is crucial. Besides, setting the *kStepAhead* value to either 0 or 1 often yields satisfactory results, suggesting that looking many steps ahead might not be necessary for this collection of datasets. Overall, the model's performance is comparable to that achieved with Cost-Complexity Pruning, but with a substantially reduced runtime.

Method 2: Out-of-Bag with p-value Threshold This method addresses the limitations

of the first method. First, we replace the separate validation set with Bootstrap sampling. Our simulations showed a performance decline when using smaller training sets, leading us to avoid excluding any training data during model fitting. The Bootstrap method is better as it can approximate the observed sample effectively, with the Out-of-Bag (OOB) samples serving as a convenient validation set. Moreover, a train-validation split at the root node could significantly change the tree shape, while bootstrap sampling occurs within each node and ensures consistency in data size.

We also adapted and modified the binomial distribution concept from Pessimistic Error Pruning. The original Pessimistic Error Pruning compares the current split to predicting everyone as the plurality class. However, it is optimistic and may be overly sensitive to randomness. To address this, let p_1 represent the prediction accuracy before the split, and p_2 the accuracy after the split. We want to test the hypothesis $H_0 : p_1 = p_2$ against the alternative $H_1 : p_1 < p_2$. We then calculate the p-value from this one-sided z-test using OOB samples and compare it to the predefined Type I error rate. One advantage of this p-value thresholding is that it reflects differences in sample size. For example, a 10% increase in OOB testing accuracy has different implications depending on whether the node has 10 or 10,000 samples.

The workflow is as follows: At each node, we initially use all data points to find the split, ensuring full utilization of the training set for tree structure formation. Within that node, we generate a bootstrap and an OOB sample. The bootstrap sample is used to fit the LDA model at that node. We then apply the split to the bootstrap data and fit LDA models in the child nodes. Finally, the OOB sample assesses the change in testing accuracy. Splitting stops if this change lacks statistical significance. Our simulations indicate that this stopping rule often yields better results than Method 1, and the test accuracies from separate test

sets closely align with OOB test accuracies.

After an extensive examination of Method 2, we decided to remove the bootstrap step and instead directly used changes in resubstitution error for the two-sample z-test (Modified Method 2). This simplification of the algorithm led to reduced runtime while maintaining comparable performance on most datasets. Additionally, we investigated the differences between pre-pruning and post-pruning in this p-value threshold method. In pre-pruning, the tree stops growing as long as the p-value is not significant. In contrast, post-pruning allows the tree to grow to full depth and then prunes from the bottom up, removing branches with non-significant p-values. In practice, we found that the post-pruning method had better performance on synthetic datasets featuring high-order interactions but insignificant main effects.

We also examined post-pruning with cross-validation, where we used CV trees to choose the best p-value threshold for the main tree. We found that post-pruning is not very helpful when there is no overfitting. Unlike typical models, the LDATree does not have the usual convex testing error pattern, where errors decrease as underfitting is resolved and then increase due to overfitting. Most of the time, it shows a monotone trend. We tested post-pruning on the 49 datasets, and summarized the results in Figure 3.10. In each small plot, the x-axis represents the p-value defined above, also the α for cost-complexity pruning. A larger p-value threshold implies fewer constraints, allowing the tree to grow deeper. The y-axis measures testing accuracy. Averaging the 49 plots yields Figure 3.11. This indicates that in general, LDATree benefits from additional splits and is robust to overfitting. However, due to the minor changes in testing accuracy, we set $\alpha = 0.01$ to achieve a shorter tree and potentially improved interpretability.

Overall, we claim that the post-pruning with cross-validation is by far the most accurate stopping rule. It is recommended if runtime is not a big concern. However, in our real data analysis, the pre-stopping rule shows good testing accuracies within

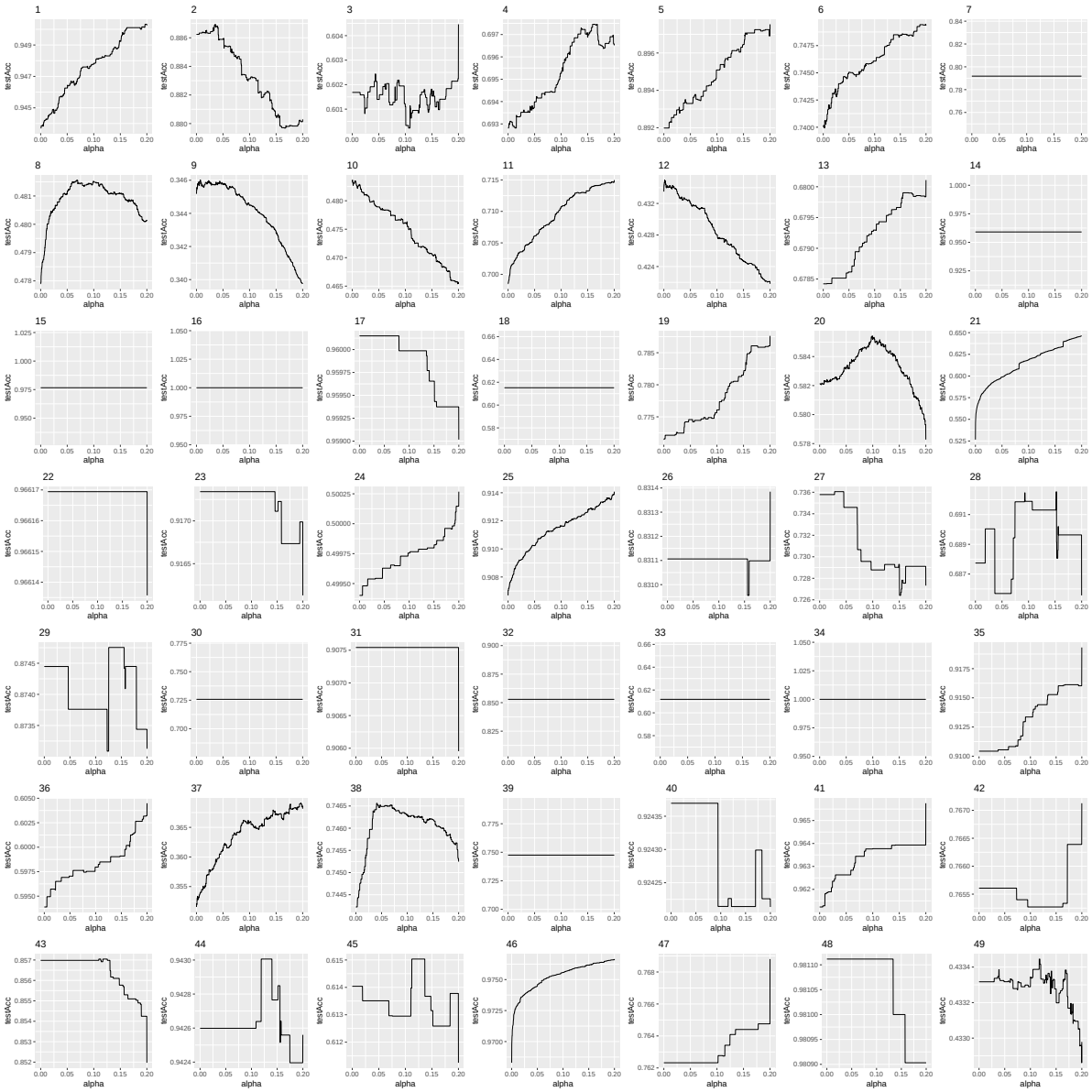

Figure 3.10: Test Accuracy vs. α: the performance of stepLDATree on 49 datasets. The individual plot title indicates the index of the dataset. α is the p-value cutoff during tree construction.

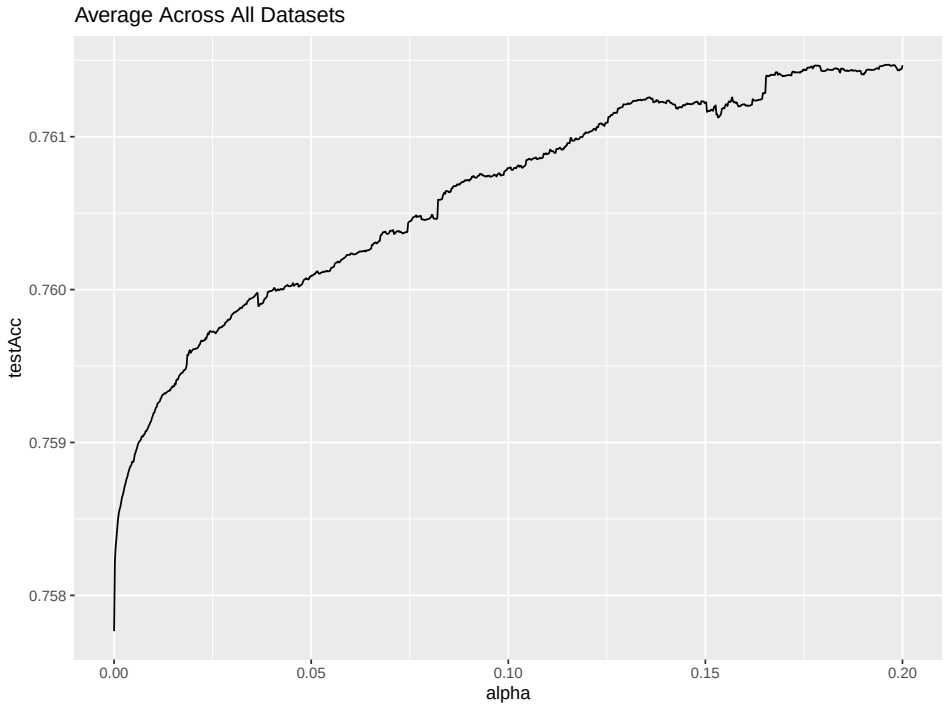

Figure 3.11: Test Accuracy vs. α: the performance of stepLDATree on 49 datasets and averaged. α is the p-value cutoff during tree construction.

the time limit. On most datasets, the differences are not significant compared to the post-pruning with cross-validation. Therefore, we set the Modified Method 2 as the default stopping rule in our implementation, and use it for the rest of our analysis unless otherwise specified. Below is a detailed description.

Suppose there are n samples in the current node. We fit the LDA model within the node and record the number of correctly predicted observations as n_1. Next, we use all data points to find the split, distribute the data points, and fit LDA models in all child nodes. We then sum the number of correctly predicted observations across child nodes and record this as n_2. Thus, we have $\hat{p_1} = n_1/n$ and $\hat{p_2} = n_2/n$. The z-statistic is calculated as

$$z = \frac{n_2 - n_1}{\sqrt{n\hat{p_1}(1-\hat{p_1}) + n\hat{p_2}(1-\hat{p_2})}}.$$

The p-value is then given by $1 - \Phi(z)$, where $\Phi(x)$ is the CDF of the standard normal

distribution.

3.4 Splitting Rule

Introduction: Classical Methods and Their Challenges

Now, let us turn our attention to another crucial aspect of decision tree architecture: the splitting rule. This rule determines how samples are distributed from a parent node to its child nodes. A good splitting rule can effectively divide the sample space so that each resulting subset is as internally homogeneous as possible (for the simple node model). Conversely, a bad splitting rule may fail to define the correct decision boundaries or might provide misleading information, potentially due to issues such as selection bias (Loh and Shih, 1997).

For many splitting methods, the process can be divided into two stages. Initially, an objective function is defined, which the method aims to optimize. This applies to both univariate splits, where the rule is based on a single variable, and to forms like linear combination (or oblique) splits. In these cases, every potential split point is exhaustively evaluated to select the one that optimizes the objective function. For example, if the split involves a numerical variable with 100 distinct values, there are 99 possible split points, and this number increases linearly with the sample size. Traditional methods that focus on univariate splits use objective functions like the Gini Index (Breiman et al., 1984) or information gain (Quinlan, 1986). In the last two decades, several methods have emerged targeting oblique splits, including Forest-RC (Breiman, 2001), Random Rotation Random Forest (Blaser and Fryzlewicz, 2016), the Canonical Correlation Forest (Rainforth and Wood, 2015), and Sparse Projection Oblique Randomer Forests (Tomita et al., 2020). While these methods vary in how they determine the linear combination of variables, they all use the same objective function within an exhaustive search framework. However, this framework might not be the best for our LDATree.

Most of the methods mentioned above rely on a simple node model that predicts all instances to be the majority class. This approach has zero training error in nodes where all instances belong to a single class. Consequently, measures of node impurity, like the Gini Index or information gain, are used as effective criteria for splitting. For LDA, a pure node is beneficial but it's not necessary. LDA is most effective when different classes are linearly separable. When pure nodes are unattainable, we can focus on another metric that characterizes linear separability, potentially enhancing LDA's performance.

Exploration of Various Splitting Methods

Leading Discriminant Score Splitting Our first method coincides with the approach outlined in (Padmanabhan et al., 1999), using the leading discriminant score (LD1 score) as the splitting criterion. The LD1 score, being a linear combination of variables, offers advantages over univariate splits, particularly in handling non-axis-aligned decision boundaries common in real data. As shown in Figure 1.1, univariate splits are less effective; they require multiple splits, forming a staircase function to approximate the boundary line. In contrast, LDA can achieve this with a single split, perpendicular to the LD1 score.

Moreover, other methods using linear combination splits either generate directions randomly (lacking guaranteed effectiveness) or require extensive time to find an effective combination due to the infinite possible directions. The LD1 score, derived from LDA, can find a powerful split efficiently without reliance on random choices or exhausted searches. It represents the best single direction to maximize Fisher's criterion. Following this, we only need to perform one exhaustive search on the LD1 score (or on sample quantiles for larger datasets), selecting the cut that maximizes the reduction in the Gini Index.

LDA Splitting This method is adapted from the LDA split introduced in FACT (Loh and Vanichsetakul, 1988), with some modifications. Within each node, we first fit an LDA model using the data in that node. Then, we create J' subnodes, where J' represents the number of distinct predicted classes. Each observation is distributed to a branch along with others in the same predicted class. Similar to the LD1 score method, this approach also originates from LDA, offering the advantage of quickly identifying powerful linear combinations of variables for splitting. However, they are different in several aspects. Firstly, this method does not always produce a binary tree, except in binary classification problems. The use of multi-way splits typically results in a shorter tree, as the sample size in each node decreases more rapidly with branching. More importantly, this method generates splits in one shot, eliminating the need for exhaustive searching. While LD1 score splitting divides the sample space into halves to maximize the Gini Index reduction, LDA splitting partitions the space into J' parts, with decision boundaries determined by LDA. Another benefit of using LDA for decision tree splitting is its guaranteed superiority over a stand-alone LDA. It becomes equivalent to a stand-alone LDA when the LDATree consists of only one LDA split and a trivial node model.

Another distinction from the FACT split concerns the hidden class scenario. Occasionally, we encounter $J' < J$, indicating that some classes are not predicted (hidden), yet these classes might appear in predictions of new cases. In such instances, we assign the observation to the class with the highest posterior probability among the non-hidden classes. However, this approach can sometimes lead to premature stopping, as illustrated in Figure 3.12. For instance, Node 3, containing 498 observations, might benefit from further splitting. In this node, LDA predicted all observations as class B and the posterior probability plot indicated the red curve is below the blue curve almost everywhere. While this pattern is

good for a prediction task, it's less effective for finding a splitting rule aimed at efficiently dividing the sample space. The lower testing accuracy should not be a big concern, since it will recover from subsequent splits. To address this, we adjust our strategy by assigning equal priors to all classes whenever there exists a dominant class. This change enables LDA to identify a cut between the centroids of the classes, thereby facilitating continued splitting. In our implementation, we switch to equal priors when the Gini Index in a particular node falls within the range $(0, 0.1]$.

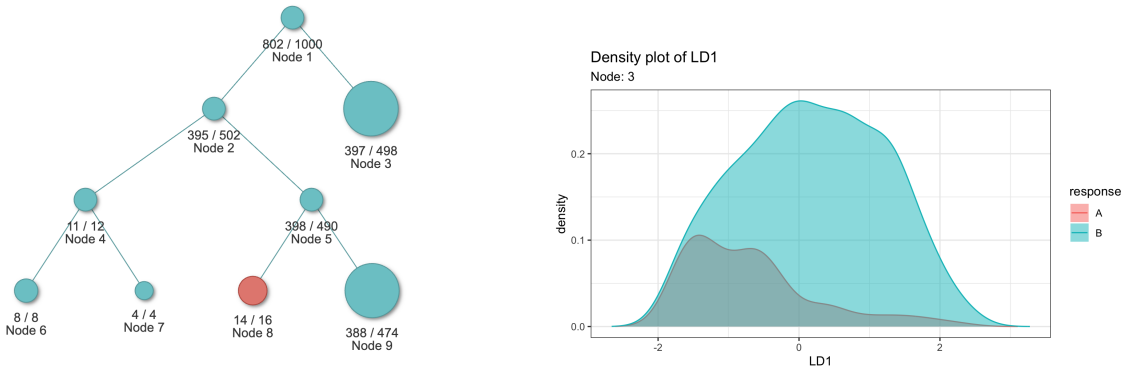

Figure 3.12: A toy example demonstrating the ineffectiveness of the LDA split in the presence of a class with a dominant prior. The right plot shows the posterior probability of data in node 3 from the left plot (Section 3.4).

Scatter Trace Splitting This method draws inspiration from the XOR (or chessboard) problem. Figures 3.13 and 3.14 demonstrate how the previous two splitting methods perform on this XOR problem. Although the results are generally fine, there is room for improvement. The primary challenge is that both univariate splits and linear combination splits struggle to identify this pattern. In symmetrical patterns like the XOR, LDA does not work effectively.

We believe that the best splitting method should complement LDA. So far, the two splitting criteria we have used seem to conflict with the LDA node model, each striving to explain more of the data variance. Our aim is for the split to play a supportive role, enhancing the effectiveness of the subsequent LDA node

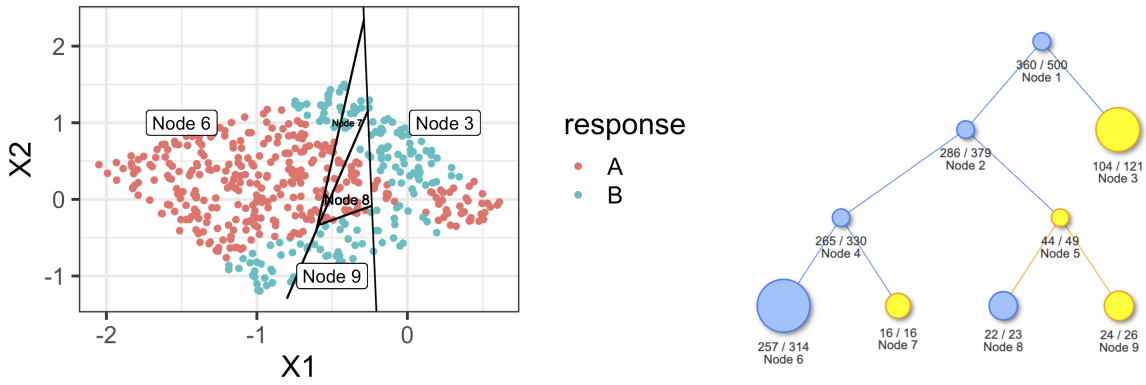

Figure 3.13: LDA splitting on XOR data (Section 3.4).

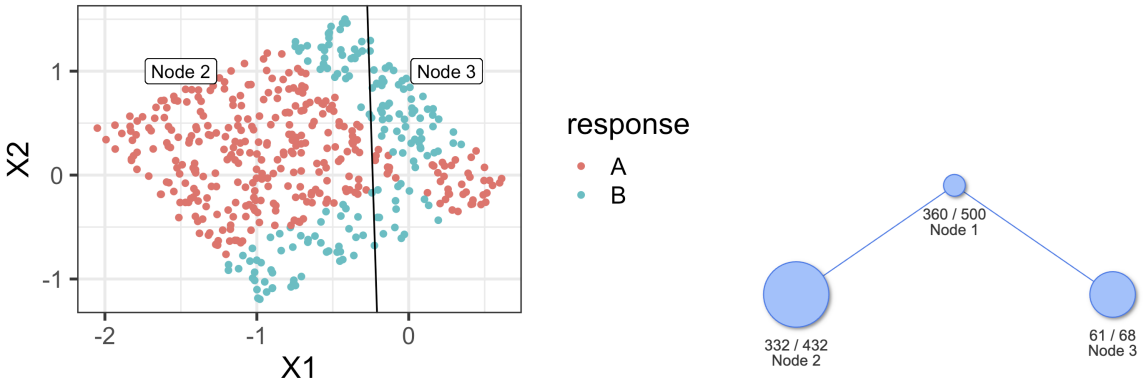

Figure 3.14: LD1 score splitting on XOR data (Section 3.4).

model. In this context, Pillai's trace, which we extensively discussed in Section 3.2, is particularly useful as it serves as a measure of LDA performance. However, we have not derived an analytical solution for this metric at this moment, so we turn to simulated annealing, using the `optim` function in R to iteratively find a numerical solution. The algorithm is outlined as follows:

- Initialize a direction vector $v_{p\times 1}$ and a scalar h. Partition the data into two groups (A and B) based on whether $Xv \geq h$ or $Xv < h$.
- For groups A and B, with n_A, n_B samples respectively, calculate $n_A \text{tr}(\mathbf{S}_B^{(A)}) + n_B \text{tr}(\mathbf{S}_B^{(B)})$. A larger value indicates better separation. The original Pillai's trace is defined as $\text{tr}(\mathbf{S}_T^{-1}\mathbf{S}_B)$. To increase speed, I disregard the correlation structure and scale all columns to have unit variance before calculating the

metric.

- Use simulated annealing to find the next v and h until convergence is reached or the iteration limit (200) is met.

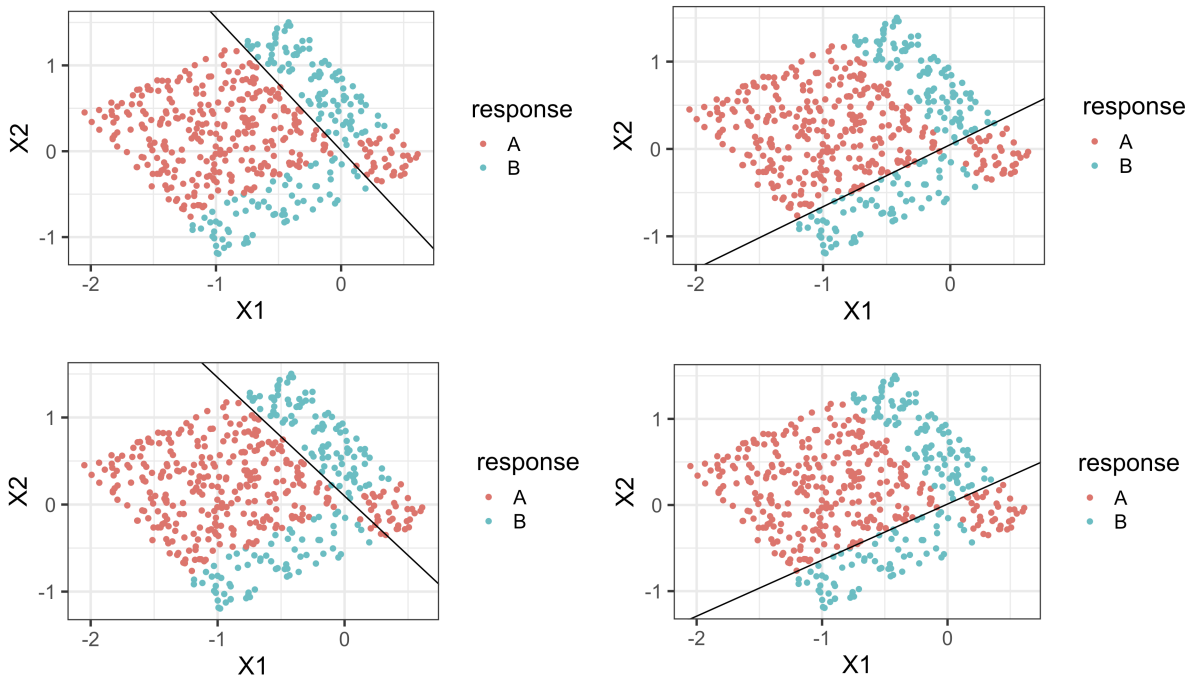

Figure 3.15: Illustration of the scatter trace splitting rule on XOR data, executed four times (Section 3.4).

As Figure 3.15 demonstrates, the scatter trace splitting effectively solves the XOR problem. However, due to the inherent randomness in the simulated annealing process and the initial value selection, the split might not always be unique.

Random Splits Random splits are widely used as splitting criteria in forest methods. These splits are fast and can help escape local optima due to their inherent randomness. We propose two candidates for random splits:

- The first, inspired by (Tomita et al., 2020), is completely random. At each node, we randomly select $\sqrt{M}$ variables and randomly assign them coefficients of ± 1 (after scaling) to form a linear combination split. We then

project all observations along this direction and split at the median. This approach creates a balanced decision tree with half of the data on each side.

- The second approach is inspired by the XOR pattern discussed earlier. In some cases, different classes might exhibit different patterns, but their centroids overlap when averaged, leading to ineffective LDA splits. A line (or hyperplane) passing through all group means (centroids) can mitigate this issue. The challenge arises with many variables, where multiple hyperplanes are possible, or with fewer variables, where it's impossible for a hyperplane to intersect all group means. For the former scenario, we rank the variables by a stepwise process. For the latter, we use the least square estimate (LSE) due to its uniqueness. Let's say we have J classes, resulting in a $J \times M$ matrix of group means. We rank the M variables by their within-to-between-class variance ratio, selecting the top J variables. If fewer than J variables are available, we choose all of them. The top-ranked variable is treated as the response variable, and we perform a linear regression of that variable on the intercept and the remaining variables. This linear regression's $\mathbf{X}$ and $\mathbf{y}$ are based on the group mean matrix ($J \times M$), not the original dataset ($N \times M$). We can't include more than J variables in order to do linear regression. And in cases where $\mathbf{X}'\mathbf{X}$ is not invertible, we remove the least important variable until it becomes invertible. The identified hyperplane is then used for splitting. Interestingly, as this hyperplane usually passes through the group means (and, by nature, near the group medians), it often results in a balanced split with about half the data on each side.

Probabilistic Adaptive Splits Random splits can also be viewed as useful alternatives when the LDA splits are ineffective. Consequently, we try to combine these two types of splitting criteria: using the original splitting rule when the LDA node model performs well, and resorting to random splits when it does not. In each

node, we calculate Pillai's trace and its associated p-value as indicators of LDA performance. The threshold for p-value is set at $0.05/100$, applying the Bonferroni correction with $\alpha = 0.05$, and considering that LDATree typically has fewer than 100 nodes.

Real Data Analysis and Conclusion

To evaluate the performance of these splitting variants, we used the same testing framework as outlined in Section 3.1. In this comparison, we evaluate a total of 22 methods, comprising 11 for LDATree and 11 for stepLDATree. The distinction between LDATree and stepLDATree lies in their approach to LDA modeling: LDATree uses LDA/GSVD for all LDA models used in both node model and splitting rules, while stepLDATree uses stepwise LDA instead. For both LDATree and stepLDATree, we test the following:

- Single splitting rule (5 methods): This category includes LD1 splitting (LD1), LDA splitting (LDA), scatter trace splitting (Trace), totally random splits (Random), and group mean splitting (GM).
- Probabilistic adaptive splits (6 methods): Here, we use a p-value threshold to determine whether to use definitive splits or random splits. With three definitive splits and two random splits, there are six methods in total.

The results for LDATree are presented in Figure 3.16, and the results for stepLDATree are summarized in Figure 3.17. The first observation from both plots is their wide confidence intervals. This can be attributed to the inherently higher variance of LDATree and stepLDATree compared to previous LDA variants. Another possible reason for these performance variations could be the inherent variability in datasets. Certain methods may excel with specific datasets while underperforming with others.

Overall, we anticipate that differences in testing accuracies across splitting methods would become statistically significant with a larger dataset collection.

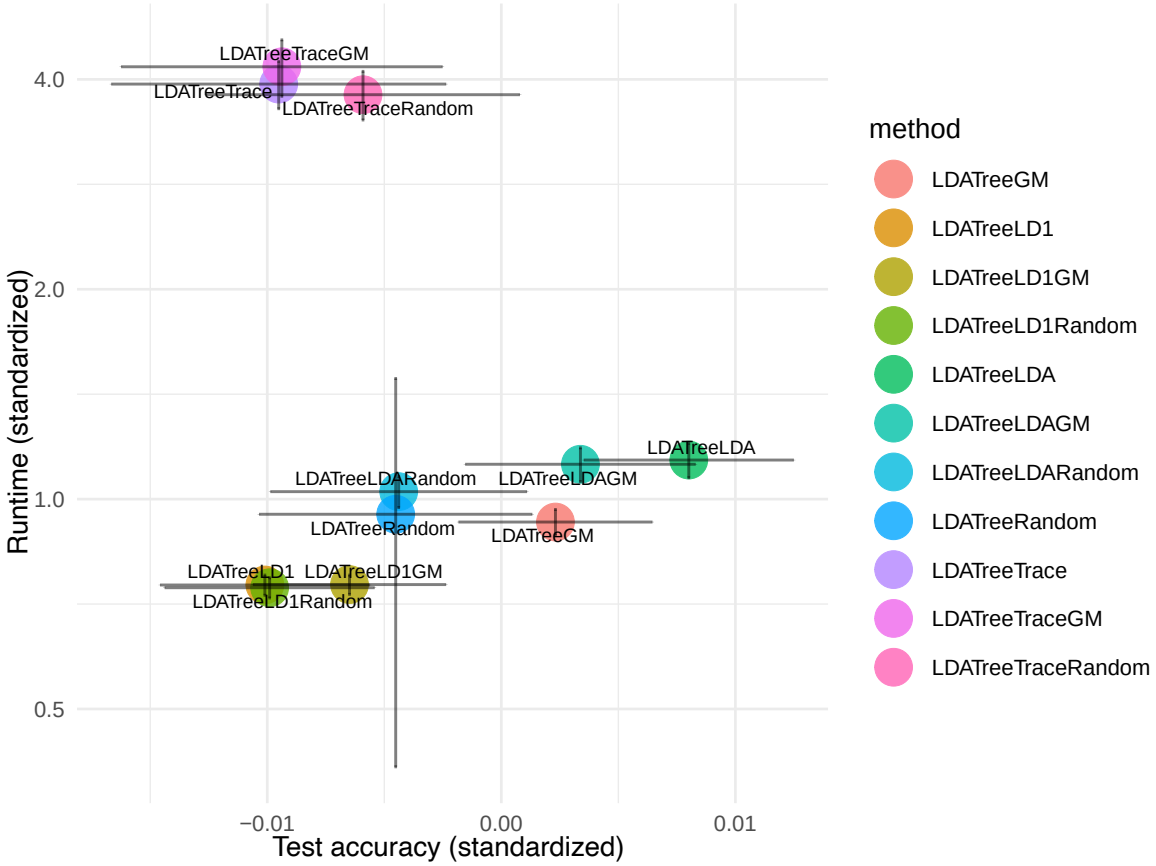

Figure 3.16: Real data analysis results in Section 3.4: Average runtime vs. average testing accuracy for LDATree. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

The methods from the Trace family suffer from their speed — they fail to complete within five minutes on over half of the datasets. Given sufficient time, their performance could potentially match or surpass other methods. For LDATree, the top-tier methods include LDA, GM, and LDAGM, with LDA being the most accurate and GM the fastest. In the case of stepwise LDA, the best performers are LDA, LDAGM, and LDARandom. Analyzing their performance across various datasets, we conclude that the LDA split is the most effective overall. Consequently, it will be adopted as the default splitting rule in both LDATree and stepLDATree.

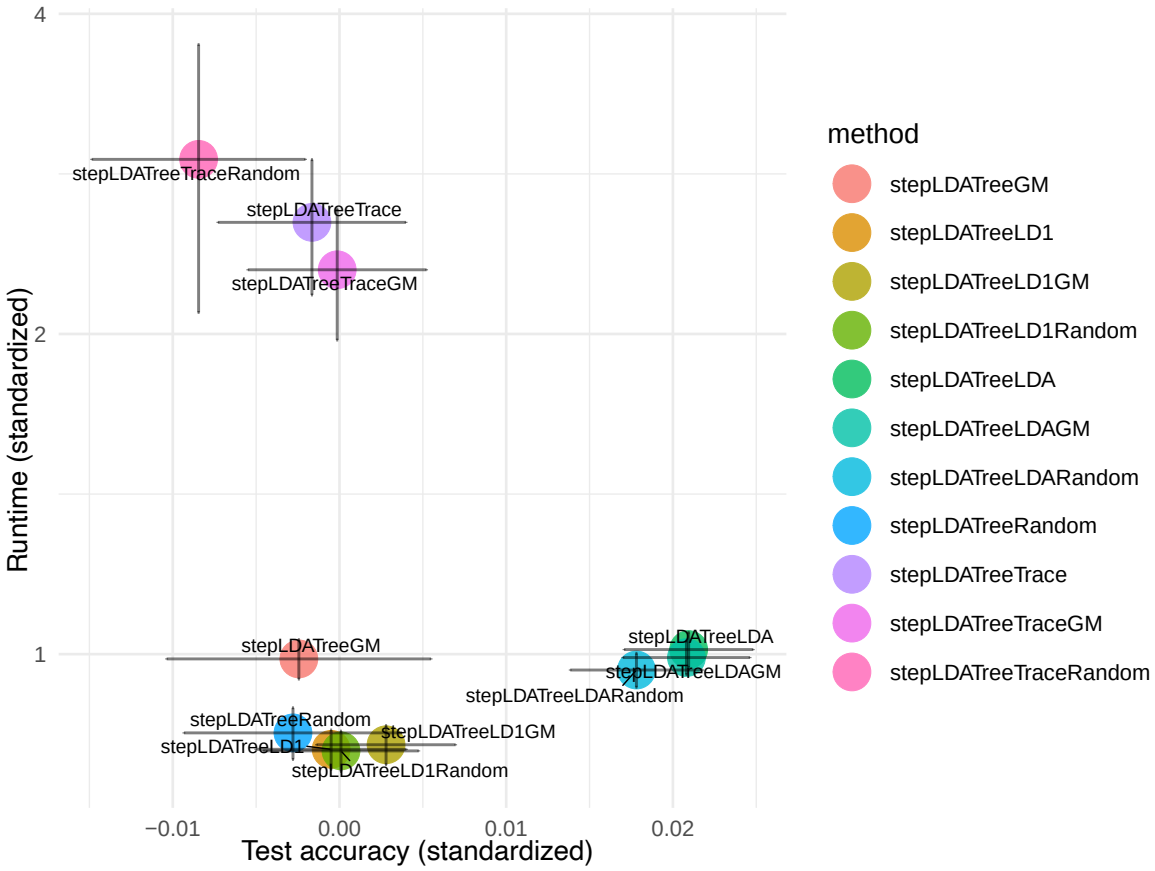

Figure 3.17: Real data analysis results in Section 3.4: Average runtime vs. average testing accuracy for stepLDATree. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

4 MISSING VALUE SOLUTIONS

The LDATree algorithm and its derivation have been covered in previous sections. All previous simulation results were based on complete datasets; this section will focus on missing values. Missing values are common in machine learning, with numerous methods developed to address them. This section will explore some of these methods within the LDATree framework. Node-wise simple imputation with missing value indicators has shown to be the most effective method in terms of runtime and prediction accuracy.

4.1 Existing Methods

The decision tree family has a special way to handle missing values using the decision tree structure. As observations move down the tree, the splitting variable may sometimes be missing. Instead of imputing these missing values, some methods can bypass them. Here are four popular methods:

- The most famous one is the **surrogate splits** Breiman et al. (1984). When the target variable for splitting is missing, it will use another backup variable to split the data.
- **Probabilistic split**, like the one in C4.5 Quinlan (1993), which distributes the current observation to all child nodes with certain weights. The weights are proportional to the relative frequencies of the child nodes and the sum of the weights is one.
- Treat missing values as a separate category, as implemented in GUIDE Loh (2002). All missing values will be distributed to either left or right, depending on which loss is smaller.

- The **internal node strategy**, which is used mainly in cost-sensitive trees Ling et al. (2004). If the target variable is missing, the observations will stay in this internal node, and the prediction is made using the training data in this node.

However, these methods are not applicable to our LDATree because LDA itself cannot handle missing values. This means that even if we bypass the missing value imputation during the splitting, we must still address them when fitting an LDA node model. Therefore, we select and list some popular methods (and their R imputations) below:

Simple Imputation Impute numerical variables using the mean or median, and categorical variables using the mode or a new level. Variants of this framework include adding missing value indicators to save the missing information or adding Gaussian errors to the imputed values to retain the variance.

Node-Wise Simple Imputation Compared to the simple imputation, this method differs in the timing of the imputation. Simple imputation imputes all missing values at the root node, treating these values as observed in all subsequent nodes. However, node-wise simple imputation handles missing values separately in each node. Once the observation is passed to the child nodes, imputed missing values from the current node are removed and re-imputed using the child nodes. This method can be viewed as conditional imputations based on the samples in the node.

Matrix Completion Matrix completion assumes that a low-rank matrix can well approximate the data matrix. This method fits patterns from observed data using SVD and applies them to impute missing values. We use the R package `softImpute` Hastie et al. (2015), which claims to combine two popular approaches into a more efficient method using fast alternating ridge regression.

Hot-Deck Imputation Hot-deck imputation replaces missing values with observed responses from similar units, termed donors, grouped into imputation cells. These cells are constructed based on the similarity of observations. The standard approach involves fitting logistic regression on propensity scores and creating imputation cells based on posterior probabilities. Besides, decision trees are also ideal for the hot-deck method, as each terminal node forms an imputation cell.

missForest Many packages use random forest for imputation; here, we select the R package `missForest` Stekhoven and Bühlmann (2012). This package uses random forests to impute missing values in all variables. After each iteration, the package records a measure of imputation performance and stops if the performance drops.

GUIDE We use a predictive model from GUIDE to impute the missing values. GUIDE can handle missing values, offering it a significant advantage over other methods. When both predictors and responses have missing values, GUIDE can directly output a predictive model, unlike most methods that must first address the missing values in predictors.

We also tested the `mice` package, but its runtime (several days) forced us to abandon it. `Amelia`, another missing value solution using the Expectation-Maximization (EM) algorithm, consistently crashed during our tests.

4.2 Proposed Methods

Inspired by the methods mentioned in the previous section, we propose two new solutions for handling missing values. The first solution allows LDA to directly handle missing values, while the second uses class-wise imputation.

To find decision boundaries, LDA requires two components: the group centroids and the scatter matrices. When faced with missing values, we can estimate the group

centroids using non-missing entries. In R, this can be done using `mean(..., na.rm = TRUE)`. The scatter matrix, which is proportional to a covariance matrix, can handle missing values using pairwise deletion. This involves using only complete pairs of variables to calculate each entry in the covariance matrix. We use the following strategies to address potential problems:

- Sometimes, the correlation may be amplified with few observations. If there are fewer than four observations, we manually set the correlation to zero. With more than four observations, we calculate Fisher's confidence interval for the pairwise correlation and set the correlation to zero if the interval covers zero.

- The covariance matrix derived from pairwise deletion may not be semi-positive definite. In such cases, we use the R function `Matrix::nearPD()` to approximate it with the nearest positive definite matrix.

However, this method has intrinsic problems that cannot be resolved. For example, using only complete observations to estimate the group centroids could result in bias if the data is not Missing Completely at Random (MCAR). Therefore, to improve imputation accuracy, we have to use information from other variables, leading to our second method. This second method involves conditional imputation based on the response variable. The basic steps for this method are as follows:

1. For each numerical variable in the current node, save the class-wise means and standard deviations (SD). If the variable is categorical, then save the frequency for each level.

2. Then, predict the class label for each observation, regardless of whether the true class label is known. The prediction model here is probability-based. For each variable, based on the previously saved information, we calculate a posterior probability table including all classes; we use normal density with the saved

mean and SD if the variable is numeric, and the saved frequency table if the variable is categorical. For a numerical variable x, the posterior probability for each class j is calculated as

$$p_j = \widehat{\pi_j} \frac{1}{\sqrt{2\pi\widehat{\sigma_j^2}}} \exp\left(\frac{(x_{\text{obs}} - \bar{x_j})^2}{2\widehat{\sigma_j^2}}\right),$$

then normalized so that $\sum_{j=1}^{J} p_j = 1$, where $\widehat{\pi_j}$, $\bar{x_j}$, and $\widehat{\sigma_j^2}$ represent the estimated class prior, mean, and variance for class j, respectively. For a factor variable x, posterior probability follows the Bayes formula, approximated as

$$p(j|x) = \frac{p(x|j)p(j)}{p(x)} \approx \frac{n_{xj}}{n_{x+}},$$

then normalized so that $\sum_{j=1}^{J} p(j|x) = 1$. Here, n_{xj} means the observed frequency of the combination $\{x = x\} \cap \{y = j\}$, and n_{x+} means the observed frequency of $\{x = x\}$. Suppose there are a total of M variables, and the current observation has K observed and $M - K$ missing. Then we have K sets of posterior probability tables. There are three methods to determine the final predicted class:

a) **Vote**: Directly determine the class using each of the K variables separately to obtain K predicted classes. Then ensemble the results, and the final predicted class will be the one with the most votes.

b) **ProdPost**: Multiply the K sets of posterior probability tables and standardize them to obtain a final posterior probability table. We then classify it to be the class with the largest posterior probability.

c) **MeanPost**: Sum the K sets of posterior probability tables and standardize them to obtain a final posterior probability table. We then classify it to be the class with the largest posterior probability.

3. Finally, impute each of the $M - K$ missing entries using the class-wise mean or

mode, where the class is the predicted class from the previous step.

4.3 Simulation

Simulation Setting

We use 25 real datasets, each containing missing values. The data description is provided in Table A.2. The last column of Table A.2 shows the percentage of observations missing at least one covariate. Notably, most datasets do not have a single complete observation, highlighting the complexity of this challenge. The testing framework is the same as in Section 3.1. For each dataset, we randomly sample 70% for training and use the remaining 30% to assess testing accuracy. This process is repeated 20 times to obtain confidence intervals for the testing accuracies.

We tested two versions of most methods: one with missing value indicators and one without. To avoid overlapping different methods, we grouped and displayed the results separately. We then put together the best method from each group for a more comprehensive understanding. We evaluated the performance of these missing value imputation methods on both LDATree and stepLDATree. However, because they are very similar, we only present the results for stepLDATree here.

Simple Methods

In this subsection, we evaluated sixteen simple imputation methods. We impute all missing entries at the root node using column statistics such as the mean and median. To differentiate these methods, we adopt an `A..B..CD` syntax for the methods' names, which denotes:

- `A`: `mean` or `median`, used to impute missing values in numerical variables.

- `B`: `mode` or `newLevel`, used for imputing missing values in categorical variables. Here, `mode` refers to the most frequent level, while `newLevel` means assigning an additional level beyond the original levels for missing entries.

- `C`: 0 or 1. If `C` equals 0, no extra steps are taken. If `C` equals 1, we add Gaussian noise to the deterministic imputation for numerical variables. This Gaussian noise has a mean of zero, and its SD is estimated from the column's SD. Consider using median imputation for column X. The column median and SD are calculated as m_X and s_X respectively, using the complete entries. For each missing entry, if `C` equals 0, it is imputed with m_X; if `C` equals 1, it is imputed with $m_X + \varepsilon_i$, where ε_i is drawn from a normal distribution $N(0, s_X^2)$.

- `D`: empty or `_IND`, indicating the use of missing value indicators. This indicator is added for every column with missing values, whether numerical or categorical. If there are M variables and K have missing values, then for `D = _IND`, the total will be $M + K$ columns.

The final results are summarized in Figure 4.1. Sixteen methods are approximately grouped into four clusters based on the missing value indicator and the use of extra Gaussian noise. It is clear that the missing value indicator improves the testing accuracy, as methods ending with `_IND` are positioned to the right of those without it, indicating higher testing accuracies. When using missing value indicators, other variants appear to have not much effect. Gaussian noise (methods with `..1` before the `_IND`) seems to speed up the program because they have shorter trees compared to those without Gaussian noise. However, the group without Gaussian noise has slightly better testing accuracy because it performs significantly better in one of the datasets compared to the group with Gaussian noise. Therefore, we consider `median..newLevel..0_IND` and `median..mode..0_IND` to be the best candidate from

this group. We favor `median..newLevel..0_IND` over `median..mode..0_IND` because `newLevel` has slightly higher testing accuracies than `mode` across all four clusters.

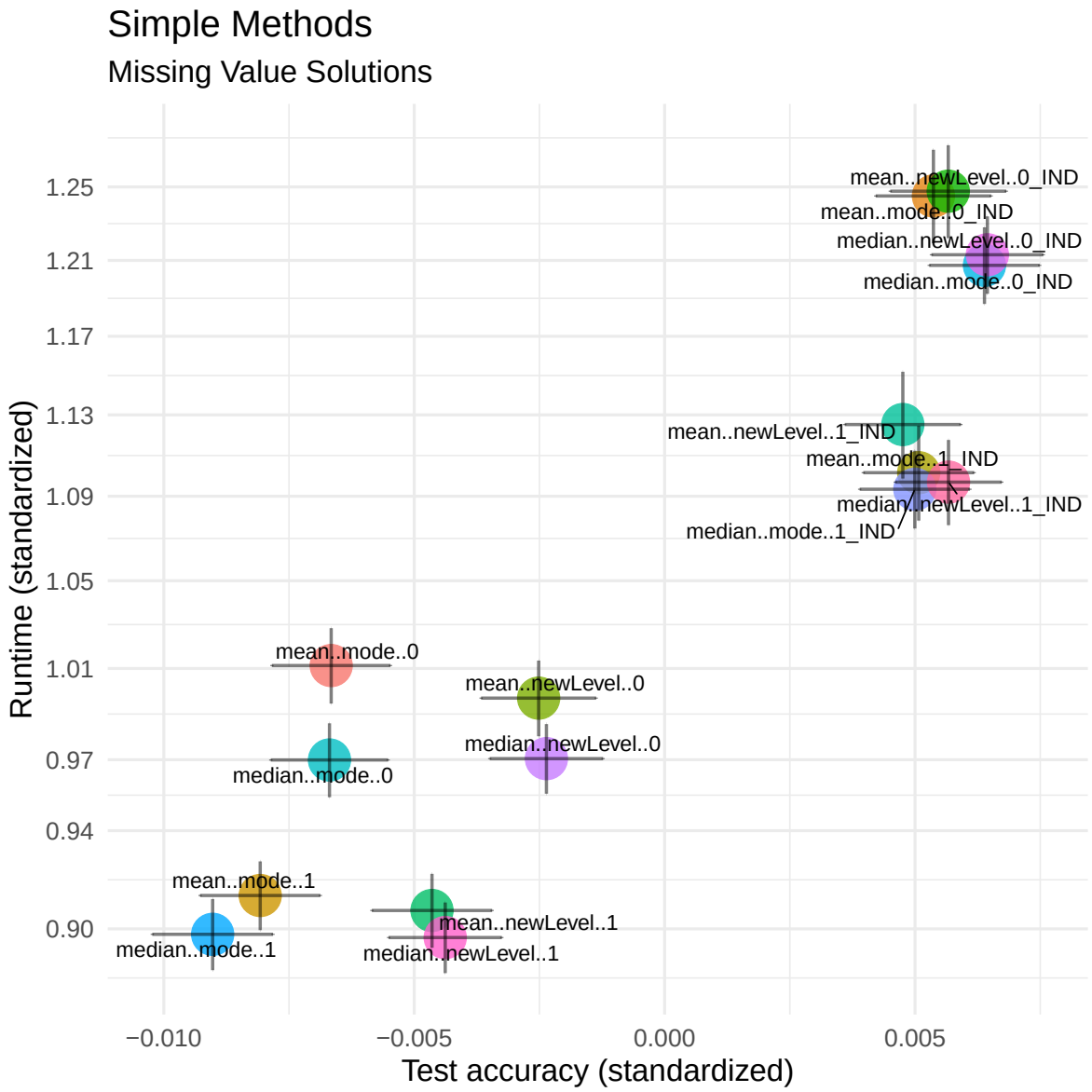

Figure 4.1: Real data analysis results in Section 4.3 using 25 datasets: Average runtime vs. average testing accuracy for stepLDATree. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

Hot-deck Methods

In this subsection, we evaluated eight hot-deck imputation methods. All of these methods are based on the GUIDE decision tree. Compared to logistic regression and other methods, using a decision tree as the hot-deck method offers several advantages:

- The key aspect of hot-deck imputation is identifying donor groups, also known as imputation cells. In logistic regression with the propensity scores as the response, the outcomes are posterior probabilities that must be manually divided into five groups. This process is somewhat arbitrary compared to decision trees, where each terminal node naturally serves as an imputation cell.

- Another advantage of decision trees is that they do not rely on assumptions about the distribution, unlike logistic regression, which assumes an S-curve on the probability distribution, limiting its generality.

- A major drawback of logistic regression and other methods is that they can not handle missing values. However, the GUIDE method does not require any pre-processing before fitting the model.

The methods we tested include:

GPTHD GUIDE Propensity Score Tree with Hot-Deck Imputation: Assume we have M variables and K of them contain missing values. For the i-th variable among the K, we define the observed flag $\mathbf{Z}$ as $Z = I(X^{(i)} \neq \text{NA})$ and fit a GUIDE propensity score tree using the available covariates $\{X^{(1)}, \cdots, X^{(M)}\} \setminus \{X^{(i)}\}$. This GUIDE method applies an internal check and ensures that the predicted $\hat{Z} \in (0, 1]$, indicating that each terminal node of the GUIDE tree contains at least one observed response. Missing values in $\mathbf{X^{(i)}}$ are then imputed using the hot-deck method, with imputation cells corresponding to the tree's terminal nodes.

GRTHD GUIDE Regression Tree with Hot-Deck Imputation: Similar to `GPTHD`, but replaces the propensity score tree with a regression or classification tree, depending on the variable type. It uses $\mathbf{X}^{(\mathbf{i})}$ as the response variable in the GUIDE model, fitted only with rows where $\mathbf{X}^{(\mathbf{i})}$ is observed.

GCTHD GUIDE Combined Tree with Hot-Deck Imputation: We want to use both the information from the missingness and the values themselves, so we combine the propensity score tree with the regression tree. This approach creates two trees, `GPTHD` and `GRTHD`. For each missing entry, one donor group is obtained from each method, with the final donor group being the intersection of these two groups. If the intersection is empty, the final donor group is the union of the two.

GCT2HD The second version of GUIDE Combined Tree with Hot-Deck Imputation: The key difference is that when the intersection is empty, the final donor group defaults to that from `GRTHD`, which, based on our experience, is more accurate than `GPTHD`. However, this method and `GCTHD` have the downside of being twice as slow.

For each method, we test two versions, one with missing value indicators and one without. The missing value indicators have nothing to do with the hot-deck imputation. They are added after all missing values have been imputed and are used only when fitting the LDATree model.

The final results are summarized in Figure 4.2. Again, this shows the necessity of missing value indicators. Although `GPTHD_IND` is twice as fast as the other three methods, we selected `GRTHD_IND` as the best candidate from this group because of its slightly higher testing accuracy.

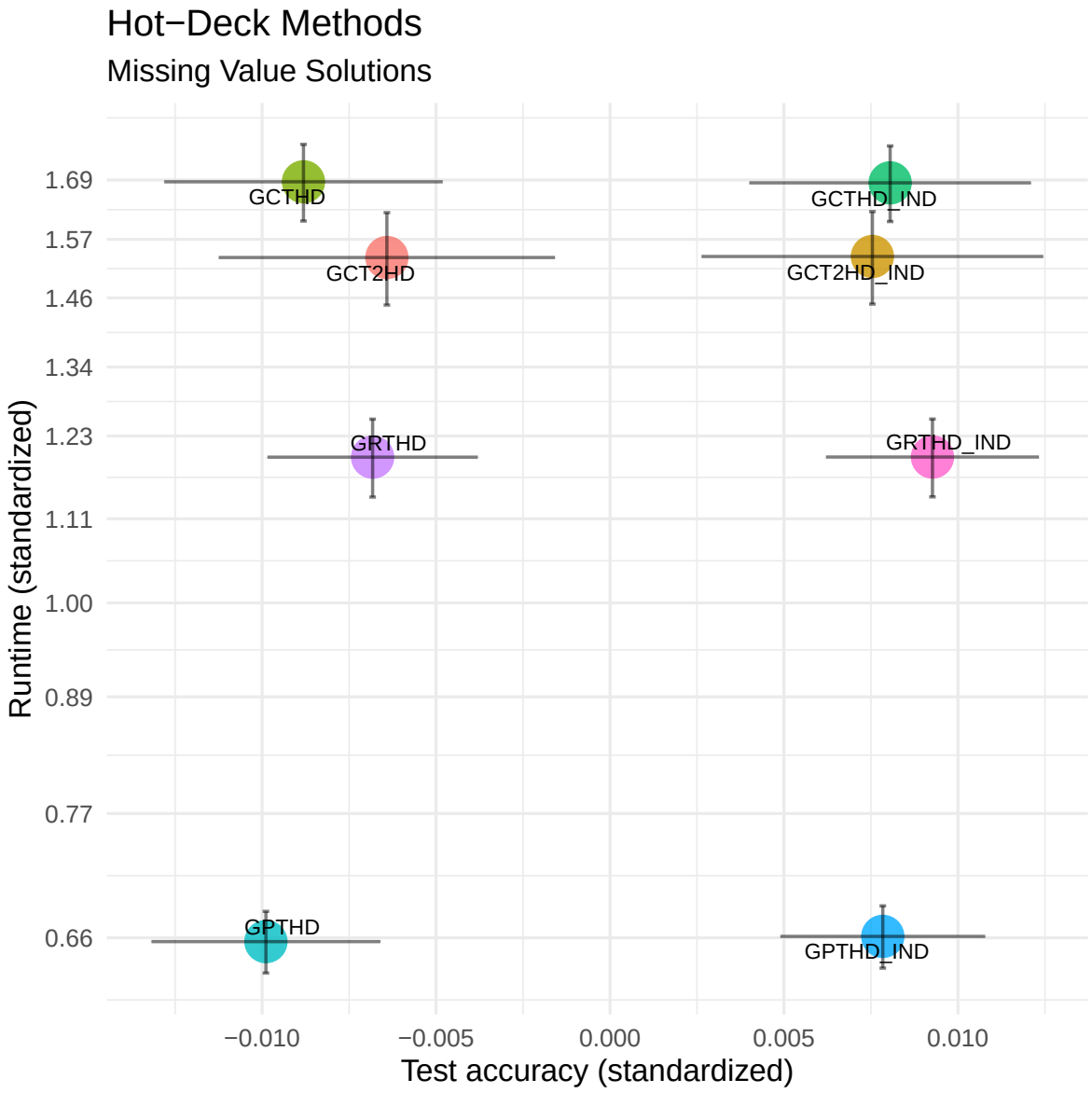

Figure 4.2: Real data analysis results in Section 4.3 using 25 datasets: Average runtime vs. average testing accuracy for stepLDATree. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

LDA-Exclusive Methods

In this subsection, we evaluate eight LDA-exclusive imputation methods. The main idea is to use complete entries to estimate group centroids and pairwise complete entries to estimate scatter matrices. More details can be found in Section 4.2. We adopt the `posAfisherBC` syntax for the methods' names, which denotes:

- `A`: `Y` or `N`, used to denote whether to use R function `Matrix::nearPD()` to approximate the scatter matrix with the nearest positive definite matrix.
- `B`: `Y` or `N`, used to denote whether to use Fisher's confidence interval for the pairwise correlation and set the correlation to zero if the interval covers zero.
- `C`: empty or `_IND`, indicating the use of missing value indicators. The missing value indicators are added before calculating the group centroids and scatter matrices.

The final results are summarized in Figure 4.3. The eight methods are approximately grouped into four clusters. Missing indicators are necessary, and approximating the scatter matrices should not be done even if they are not semi-positive definite. We selected `posNfisherN_IND` as the best candidate because of its slightly higher testing accuracy than `posNfisherY_IND`.

Class-Wise Methods

In this subsection, we evaluate six class-wise imputation methods. The main idea is to predict the class for each observation, and then use the class-wise mean or mode to impute these values. Although the predicted class may not always be correct, it is expected to be more accurate than class-blind imputations. More details, including the three methods we propose, can be found in Section 4.2. Given our results in previous sections on the importance of missing value indicators, all methods tested here have

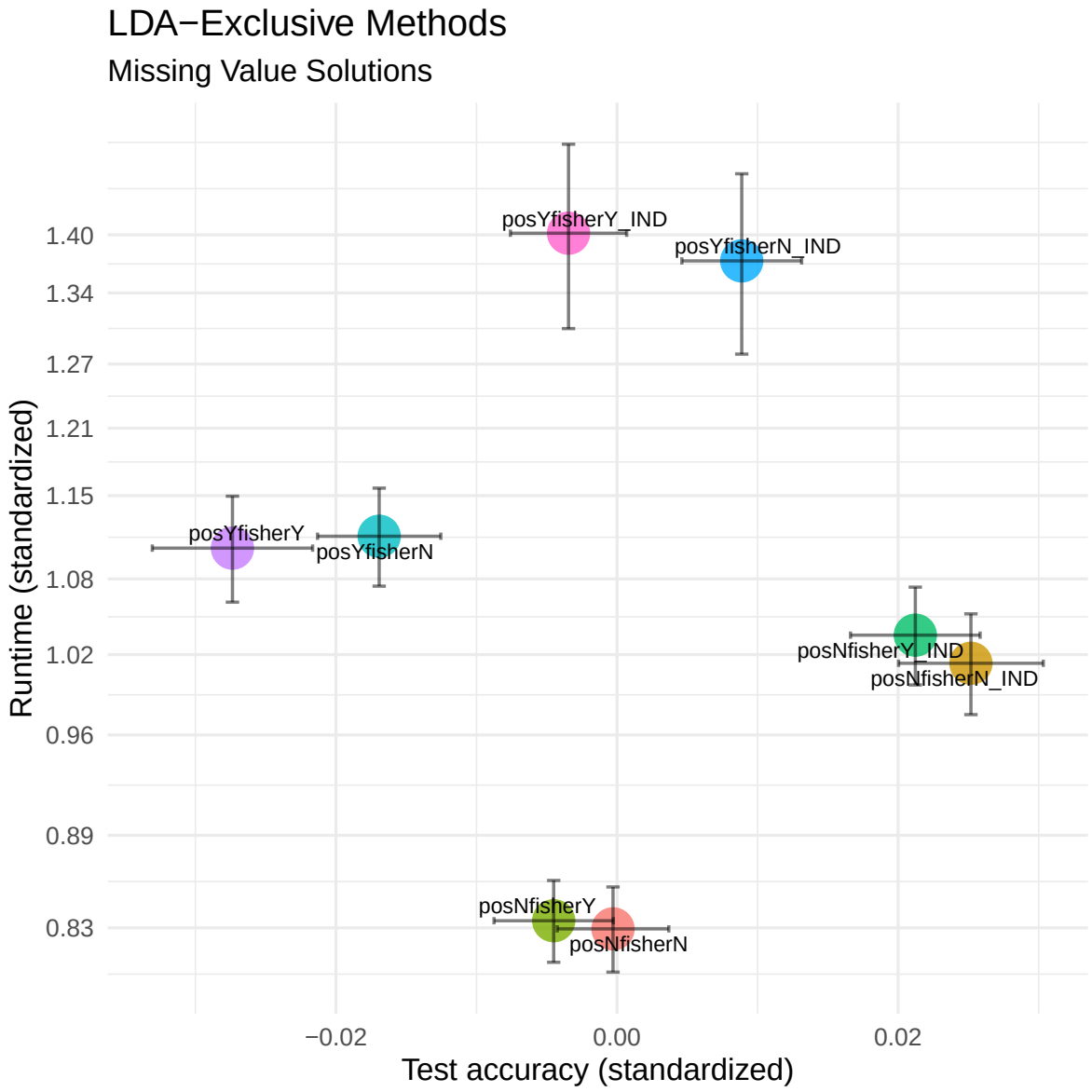

Figure 4.3: Real data analysis results in Section 4.3 using 25 datasets: Average runtime vs. average testing accuracy for stepLDATree. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

them. If the method's name includes `_chi`, this indicates that an additional chi-squared-based variable selection is performed for class prediction. This step aims to eliminate noise variables that might affect prediction accuracy.

The final results are summarized in Figure 4.4. Except for `prodPost_IND`, all other methods had similar performance. We selected `meanPost_chi_IND` as the best candidate due to its slightly higher testing accuracy.

4.4 Combined Result and Conclusion

Now, let's combine the best candidate from each method group with methods that do not belong to any category for a comprehensive comparison. The descriptions of methods are summarized in Table 4.1, and the final result is summarized in Figure 4.5. Based on the runtime and testing accuracies, `nodeWise_IND` and `median..newLevel..0_IND` appear to be the best candidates. We will compare these two methods more carefully in Section 4.4.

Below are some comments during the simulation:

- The matrix completion method has three issues. First, it requires transforming categorical variables into dummy variables before fitting the model, so the imputed values in dummy columns may not be strictly 0 or 1. The second, and the more critical issue is that the fitted model cannot predict new observations since it modifies rows and columns simultaneously. Our tentative solution is to vertically combine the training and testing sets and perform matrix completion a second time for imputations in the testing set. Third, the tuning process for the key parameter λ is both time-consuming and ill-defined. The package authors suggest manually tuning the parameter through trial and error. Therefore, we skip this step, select the highest possible rank, and set $\lambda_{\text{used}} = \sqrt{\lambda_{\max}}$.

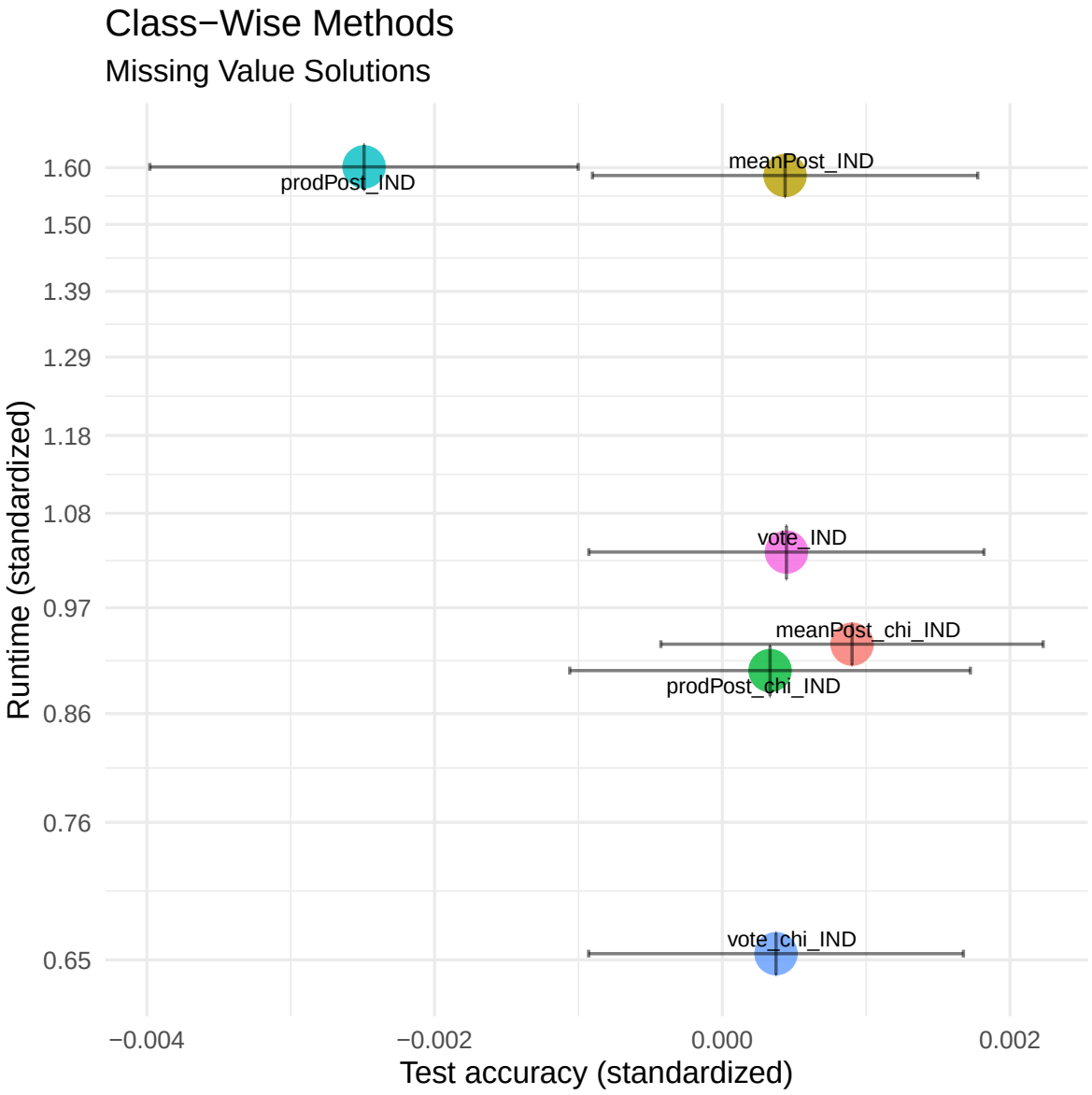

Figure 4.4: Real data analysis results in Section 4.3 using 25 datasets: Average runtime vs. average testing accuracy for stepLDATree. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

method	description
median..newLevel..0_IND	Use the median for numerical variables and a new level for categorical variables. Add missing value indicators for numerical variables. Imputation occurs only once in the root node.
nodeWise_IND	Use the median, a new level, and missing indicators. Imputation occurs at each intermediate node.
meanPost_chi_IND	We apply chi-squared-based variable selection to eliminate noise variables and calculate posterior probabilities to predict the class for each observation. We then average these probabilities across all predictors, predict the class, and impute missing values using the class-wise mean or mode. Missing value indicators are added. More details can be found in Section 4.2.
GRTHD_IND	GUIDE Regression Tree with Hot-Deck Imputation. Missing value indicators are added. More details can be found in Section 4.3.
GUIDE_IND	This method is similar to `GRTHD` from Section 4.3, but it does not use hot-deck. For each variable with missing values, a GUIDE regression or classification tree is fitted according to its type, and predicted values are used directly for imputation.
posNfisherN_IND	Use complete entries to estimate group centroids and pairwise complete entries for scatter matrices. Missing value indicators are added. More details can be found in Section 4.3 and Section 4.2.
softImpute_IND	It is the matrix completion method that uses the R package `softImpute`. Missing value indicators are added. More details can be found in Section 4.2.
missForest_IND	It is the `missForest` method that is introduced in Section 4.2. Missing value indicators are added.

Table 4.1: Descriptions of missing value methods tested in Section 4.4.

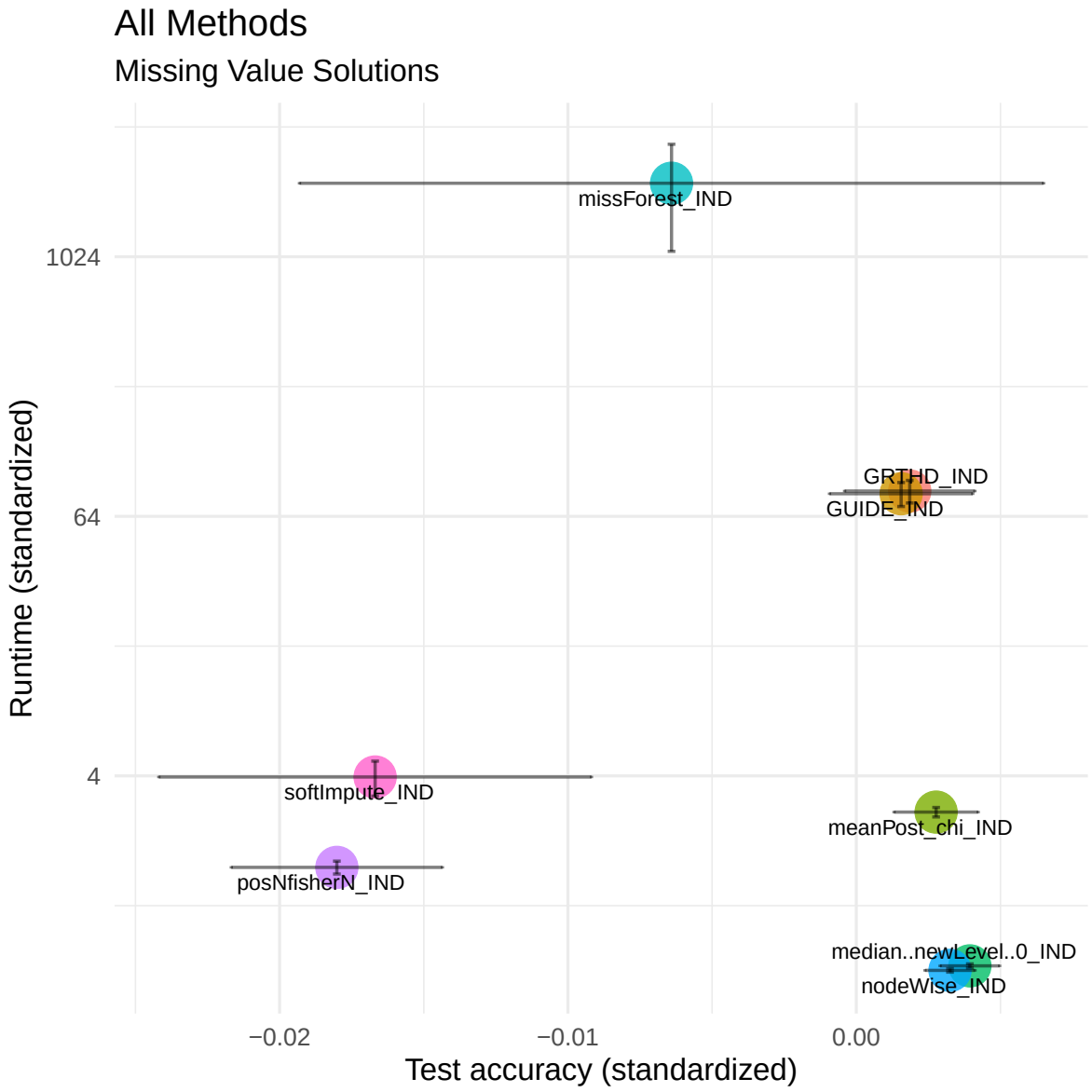

Figure 4.5: Real data analysis results in Section 4.4 using 25 datasets: Average runtime vs. average testing accuracy for stepLDATree. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

- Like `softImpute`, `missForest` cannot save the fitted model for future prediction. Again, we vertically combine the training and testing sets and rerun `missForest` to impute missing values in the testing set. Additionally, as it uses `randomForest` backend, which cannot handle factors with more than 53 levels, we delete such columns.

- For the `GUIDE_IND` method, only one iteration occurs over all variables, unlike `mice`. GUIDE can directly handle missing values, thereby eliminating the need for multiple imputations and avoiding the randomness propagated by `mice`.

Root-Node and Node-Wise Simple Imputation

In previous simulations, `nodeWise_IND` and `median..newLevel..0_IND` turned out to be the best candidates. This section will compare their differences. The method `median..newLevel..0_IND`, which is our root-node simple imputation method, imputes only once at the root node by imputing the sample median for numerical variables, adding missing value indicators, and imputing categorical variables with a new level. On the other hand, `nodeWise_IND` uses the same imputation techniques for numerical and categorical variables but performs these at every intermediate node. We denote this specific imputation technique as **M1**, and we have the following results.

Lemma 4.1. *For a numerical predictor* $\mathbf{X}$ *containing missing values, we impute with a constant* C *and add the missing value indicator* $X^- = I(X = NA)$*. Then, the column spaces of* $\{\mathbf{X}, \mathbf{X}^-\}$ *does not depend on the choice of* C*.*

Proof. We want to show that the column spaces of $\{\mathbf{X}_{c_1}, \mathbf{X^-}_{c_1}\}$ and $\{\mathbf{X}_{c_2}, \mathbf{X^-}_{c_2}\}$ are the same for any c_1, c_2.

Any element $\mathbf{E}$ in the column space of $\{\mathbf{X}_{c_1}, \mathbf{X^-}_{c_1}\}$ can be expressed as $\mathbf{E} = a * \mathbf{X}_{c_1} + b * \mathbf{X^-}_{c_1}$. For the i-th entry that is not missing in $\mathbf{X}$, $E_i = a \cdot X_i + b \cdot 0$. For the j-th entry that is missing in $\mathbf{X}$, $E_j = a \cdot c_1 + b \cdot 1 = a \cdot c_2 + (b + a \cdot c_1 - a \cdot c_2) \cdot 1$.

Thus, $\mathbf{E}$ can be rewritten as $\mathbf{E} = a \cdot \mathbf{X}_{c_1} + b \cdot \mathbf{X}^-{}_{c_1} = a^* \cdot \mathbf{X}_{c_2} + b^* \cdot \mathbf{X}^-{}_{c_2}$, where $a^* = a$ and $b^* = b + a \cdot c_1 - a \cdot c_2$. Therefore, $\mathbf{E}$ belongs to the column space of $\{\mathbf{X}_{c_2}, \mathbf{X}^-{}_{c_2}\}$, showing that $\text{span}(\mathbf{X}_{c_1}, \mathbf{X}^-{}_{c_1}) \subseteq \text{span}(\mathbf{X}_{c_2}, \mathbf{X}^-{}_{c_2})$. Similarly, we have $\text{span}(\mathbf{X}_{c_2}, \mathbf{X}^-{}_{c_2}) \subseteq \text{span}(\mathbf{X}_{c_1}, \mathbf{X}^-{}_{c_1})$. Eventually, this shows that $\text{span}(\mathbf{X}_{c_1}, \mathbf{X}^-{}_{c_1}) = \text{span}(\mathbf{X}_{c_2}, \mathbf{X}^-{}_{c_2})$. □

Lemma 4.2. *For K numerical predictors $\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_K$ containing missing values, the combined column spaces of $\{\mathbf{X}_1, \mathbf{X}_1^-, \mathbf{X}_2, \mathbf{X}_2^-, \cdots, \mathbf{X}_K, \mathbf{X}_K^-\}$ does not depend on the choice of $C_1, C_2, \cdots, C_K$.*

Proof. This result is a simple extension of Lemma 4.1. □

Lemma 4.3. *The decision rules from the LDA/GSVD model do not depend on the choice of $C_1, C_2, \cdots, C_K$.*

Proof. Suppose we have two design matrices $\mathbf{X}_1$ and $\mathbf{X}_2$ where the only difference is their choices of C_i. Based on Lemma 4.2 we know that they share the same column space, which means $\mathbf{X}_2 = \mathbf{X}_1\mathbf{R}$ where $\mathbf{R}$ is full rank square matrix. LDA/GSVD tries to find the transformation matrix $\mathbf{W}$ that can maximize the criterion Ye and Yu (2005)

$$F_{\mathbf{X}}(\mathbf{W}) = \text{trace}\left((\mathbf{S}_W)^+ \mathbf{S}_B\right)$$

where $(\mathbf{S}_W)^+$ denotes the pseudo-inverse of the within-class scatter matrix.

Suppose the transformation matrices we find for $\mathbf{X}_1$ and $\mathbf{X}_2$ are $\mathbf{W}_1$ and $\mathbf{W}_2$, respectively. Then we have

$$F_{\mathbf{X}_2}(\mathbf{W}_2) = F_{\mathbf{X}_1}(\mathbf{R}\mathbf{W}_2) \leq F_{\mathbf{X}_1}(\mathbf{W}_1).$$

By similar argument, we have $F_{\mathbf{X}_1}(\mathbf{W}_1) \leq F_{\mathbf{X}_2}(\mathbf{W}_2)$. Therefore, $F_{\mathbf{X}_1}(\mathbf{W}_1) = F_{\mathbf{X}_2}(\mathbf{W}_2)$, and their decision rules are the same. Here, the same decision rule means

the same discriminant power, but the transformation matrix can differ by a full-rank rotation matrix. □

Theorem 4.4. *For the LDATree model, the node-wise imputation and the root-node imputation will lead to the same decision rule.*

Proof. The difference between these two imputation methods is their choice of C_i. For categorical variables, they both impute with a new level. For numerical variables, the root-node imputation always imputes with the median from the root node, while the node-wise imputation imputes with the median from the current node, leading to a different set of C_i.

In the LDATree model, we use LDA/GSVD for both the node model and split. As long as the current LDA/GSVD models are the same for both methods, they will share the same split and child node structure, and the rest will follow. Therefore, we only need to prove that in one particular node, these two imputations yield the same LDA/GSVD model while having different choices of C_i, which is proved in Lemma 4.3. □

Generally speaking, node-wise imputation should yield a different solution compared to root-node imputation. However, we do two things differently: we add the missing value indicators and use the LDA split. Together, these make the two methods equivalent under the LDATree model. Although the two trained models are the same, the testing results could differ if both of the following occur:

1. There exists perfect linear dependency in the design matrix.

2. The new testing data does not comply with the same linear dependency.

During our simulation, this seldom happens. When it does, the difference is negligible.

When fitting a stepLDATree model, the difference between the two imputation methods becomes much harder to investigate, as taking a subset of variables changes the column space of the design matrix. The only scenario in which these two methods yield different results is when all of the following occur:

1. The stepLDATree should have at least two levels (depth > 2), since the node-wise imputation at the root node is the same as the root-node imputation. This also means that the LDA at the root node cannot be so powerful that subsequent splits are rendered useless.

2. To find a difference in one specific node, the two imputation methods should impute different values. This means the node-wise median should differ from the overall median.

3. To find a difference in one specific node, the stepwise LDA should select different sets of variables. This means that for at least one variable, one imputation method will treat it as significant while the other will not.

The trickiest part is the third point above. The significance of the variable after imputation depends on the class structure, so neither of these two methods is theoretically admissible. Based on the testing accuracies on real datasets, the difference between the two methods when fitting stepLDATree is negligible. Due to the three constraints above, it is hard to find a simulation example to illustrate the difference between these two methods. We favor root-node imputation for the following reasons:

1. It is computationally faster. Root-node imputation imputes once at the root node, while node-wise imputation imputes at every node.

2. It provides more stable results when certain nodes have very few observations. The median estimate in that particular node will have a much higher variance.

3. Node-wise imputation might generate some unreasonable decision paths, which are hard to interpret. Suppose one particular observation has missing values on X_1, and has been imputed with a, b, and c in the 1st, 2nd, and 3rd split using node-wise imputation. To end up in the terminal nodes it reaches, the value of X_1 takes three different values. However, in real life, there is only one realization of X_1. The three splits that this observation passes through might never form a feasible decision path if the imputation remains the same in all three splits. Root-node imputation, on the other hand, does not have this type of logical issue.

4. It has a slightly better testing accuracy compared to the node-wise imputation based on our simulation.

Therefore, we decide to use root-node imputation as our default missing value solution.

5 SIMULATION

In this section, we use synthetic datasets to illustrate the strengths of stepLDATree and LDATree. First, we demonstrate their performances on understandable 2D patterns. Second, we identify two scenarios where the performance of stepLDATree surpasses that of the random forest. Details about the other methods used for comparison can be found in Table 5.1. Note that we use the default parameters in all methods without special tuning. For LDATree and stepLDATree, we use the post-pruning stopping rule in this section.

5.1 Some 2D Patterns

3X3 Square

The first pattern is a 3x3 square with three classes. The shape is carefully designed to be symmetric, with the class centroids of all three classes being the same. Each small square contains 1000 points, resulting in a total of 9000 points. We split the data 50:50 for training and testing purposes, and the results are summarized in Figure 5.1. Except for the original pattern, other plots show the prediction regions.

Due to the symmetric pattern, LDA struggles and classifies all observations into the majority class due to its higher prior. All other methods perform well. Notice that

method	description
LDA	The LDA/GSVD method, which is introduced in Section 3.1
SVM	Support vector machine from the R package `e1071`.
rpart	CART method from the R package `rpart`.
GUIDE	GUIDE classification tree, introduced in Section 4
GF	GUIDE forest, the forest version of the GUIDE classification tree.
ranger	The random forest method from the R package `ranger`.

Table 5.1: Descriptions of the methods tested in Section 5.

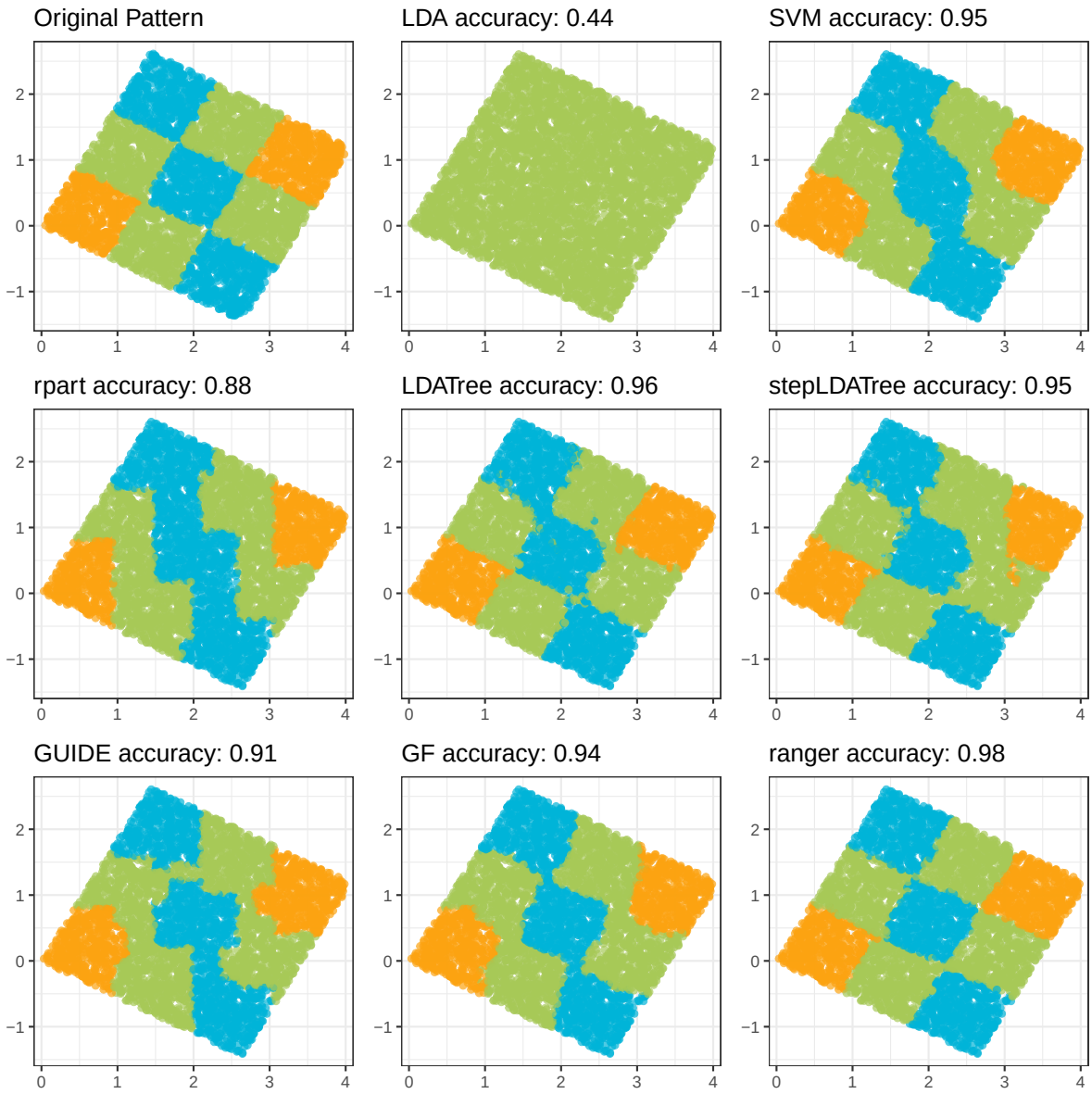

Figure 5.1: The first simulation results (3X3 square) in Section 5.1. Except for the original pattern, other plots show the prediction regions.

methods `rpart` and `GUIDE` have obvious axis-orthogonal decision boundaries, which limits their performance.

Spiral Shape

The second pattern is spiral shapes with three classes. Each class contains 5000 points, resulting in a total of 15000 points. We split the data 50:50 for training and testing purposes, and the results are summarized in Figure 5.2.

Again, LDA struggles because it only supports linear boundaries. Spiral decision boundaries are needed here, and SVM also has difficulty finding effective boundaries. `rpart`, `GUIDE`, and `GF` perform poorly in this scenario, partly due to their axis-orthogonal splits. In contrast, LDATree, stepLDATree, and `ranger` yield satisfying results.

Concentric Circles

The third pattern contains concentric circles from four classes. Each class contains 2000 points, resulting in a total of 8000 points. We split the data 50:50 for training and testing purposes, and the results are summarized in Figure 5.3.

Here, SVM and `ranger` have the best performance. Other methods have similar performances, except for LDA.

5.2 Use Case for StepLDATree

Robustness to Noise Variables

In this section, we aim to show that stepLDATree is robust to noise variables and can simultaneously be used as a variable selection tool. This is a useful property, especially if you have many variables.

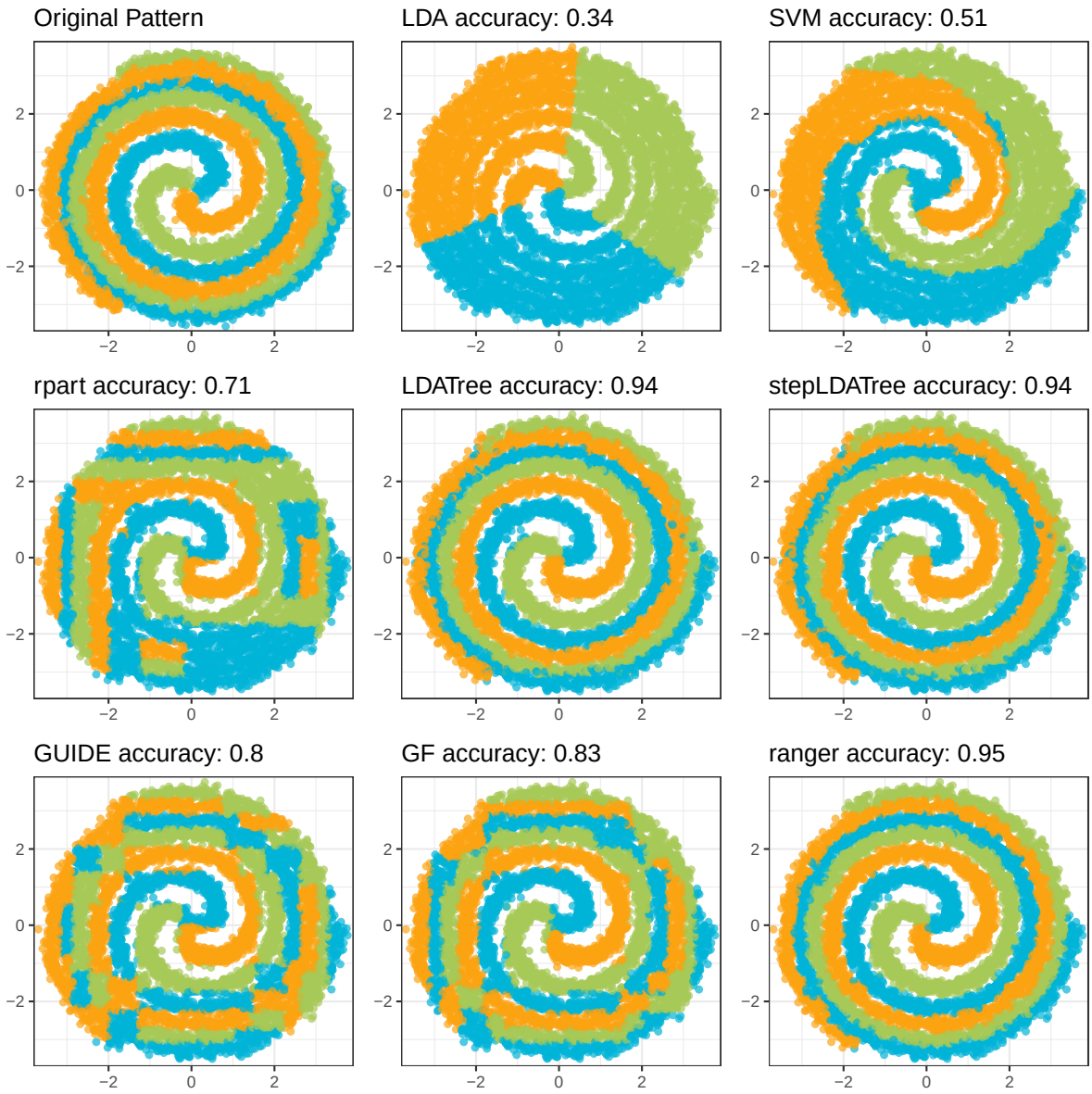

Figure 5.2: The second simulation results (spiral) in Section 5.1. Except for the original pattern, other plots show the prediction regions.

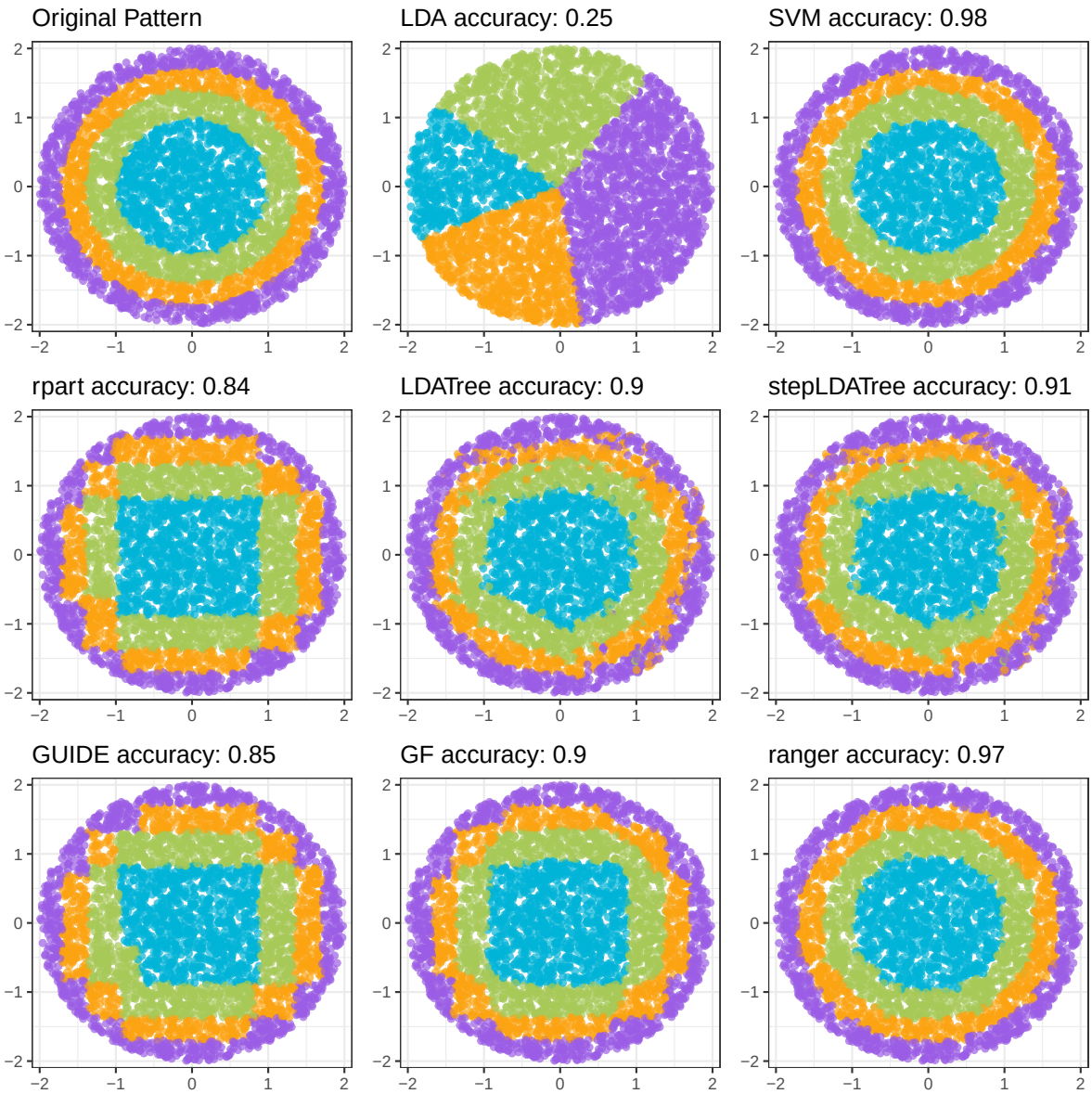

Figure 5.3: The third simulation results (concentric circles) in Section 5.1. Except for the original pattern, other plots show the prediction regions.

We use the 3X3 square data from Section 5.1. In addition to the two informative variables, we add 300 pure Gaussian noise variables to both the training and testing sets. Results are summarized in Figure 5.4.

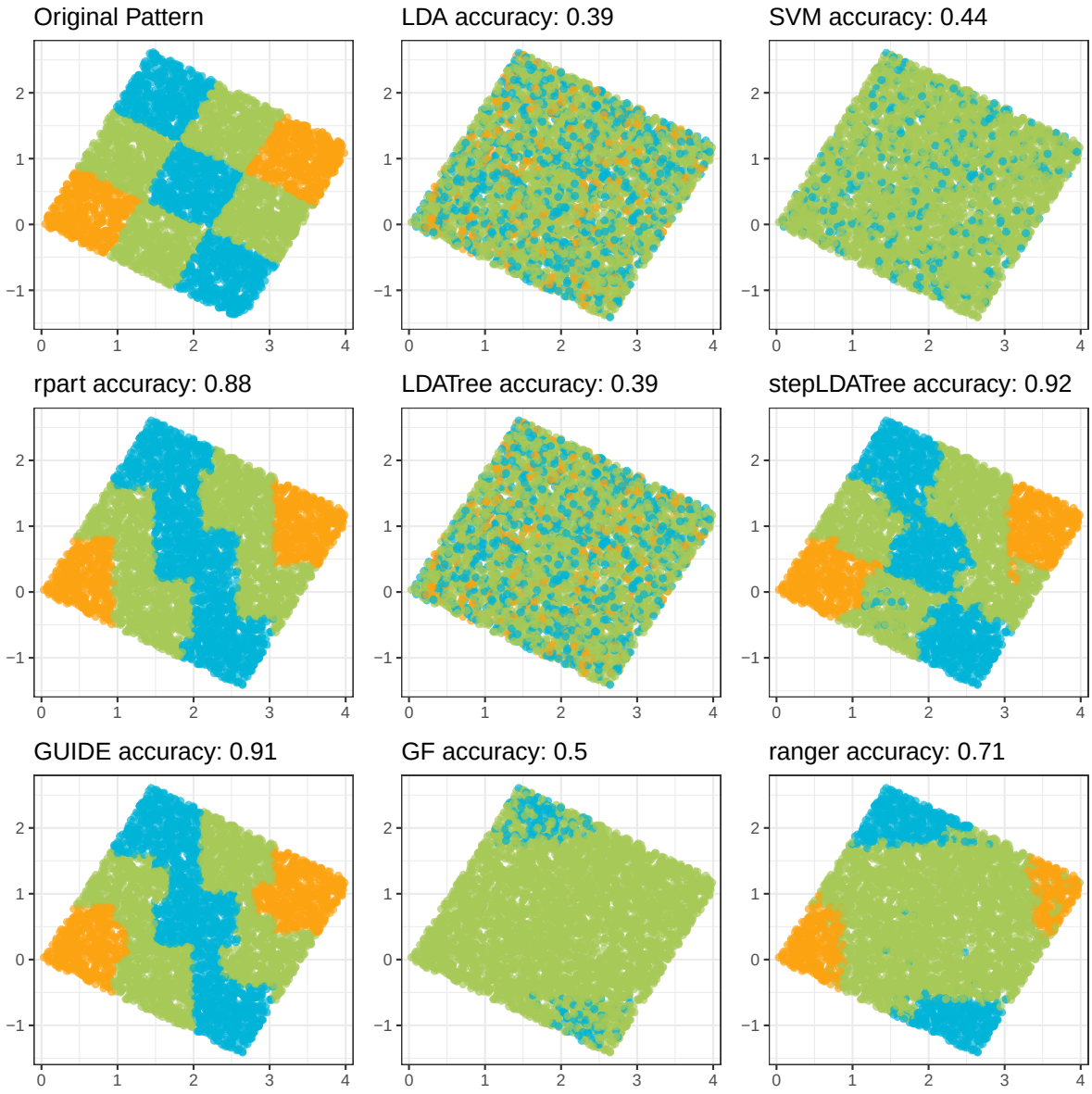

Figure 5.4: The simulation results (3X3 square + noises) in Section 5.2. Except for the original pattern, other plots show the prediction regions.

Most methods perform much worse compared to the previous situation without noise variables. Notably, `ranger` is also heavily affected by noise variables. When performing classification problems, the random forest randomly chooses $\sqrt{M}$ variables

(where M is the number of available variables) in each split, so the chance of selecting the informative variables becomes much lower. In this example, the signal-to-noise ratio is $2:300$. `GF`, SVM, and LDATree also suffer from the same problem. However, stepLDATree, `GUIDE`, and `rpart` achieve decent testing accuracy due to their ability to ignore useless variables. This result shows that stepLDATree has a higher chance of outperforming the random forest when there are many noise variables.

Look-Ahead Splitting

In this section, we will show that both LDATree and stepLDATree have the ability to look ahead when splitting the tree, giving them an advantage over other decision tree methods with more greedy strategies.

Our simulation result is based on the XOR shape in five-dimensional space. The standard XOR problem is in two-dimensional space with two classes. $(0,0)$ and $(1,1)$ return 0, while $(0,1)$ and $(1,0)$ return 1. In a 2D plot, this looks like a chessboard pattern. The reason methods like CART or random forest struggle with this pattern is that there is no single cut that can significantly decrease the impurity. They need to look one step ahead, make one trivial cut, and then make the powerful cut. For a 3D XOR, it requires looking two steps ahead. As you can imagine, the difficulty of the problem increases with dimensionality.

To set up the simulation, we first find the 32 centers in the five-dimensional space. Suppose we denote the five variables as $X_1, X_2, \cdots, X_5$. Each variable can take either 0 or 1, leading to $2^5 = 32$ centers. We define our response variable Y as $Y = (X_1 + X_2 + X_3 + X_4 + X_5) \mod 2$. We sample 200 data points around each center. For a specific center $\mathbf{C} = (c_1, c_2, \cdots, c_5)$, the points are sampled from $\mathcal{N}(\mathbf{C}, 0.2 \times \mathbf{I}_5)$. This results in a total of $200 \times 32 = 6400$ points. We split the data 50:50 for training and testing purposes, and the results are summarized in Figure 5.5. Note that the confidence intervals are calculated based on 96 replications.

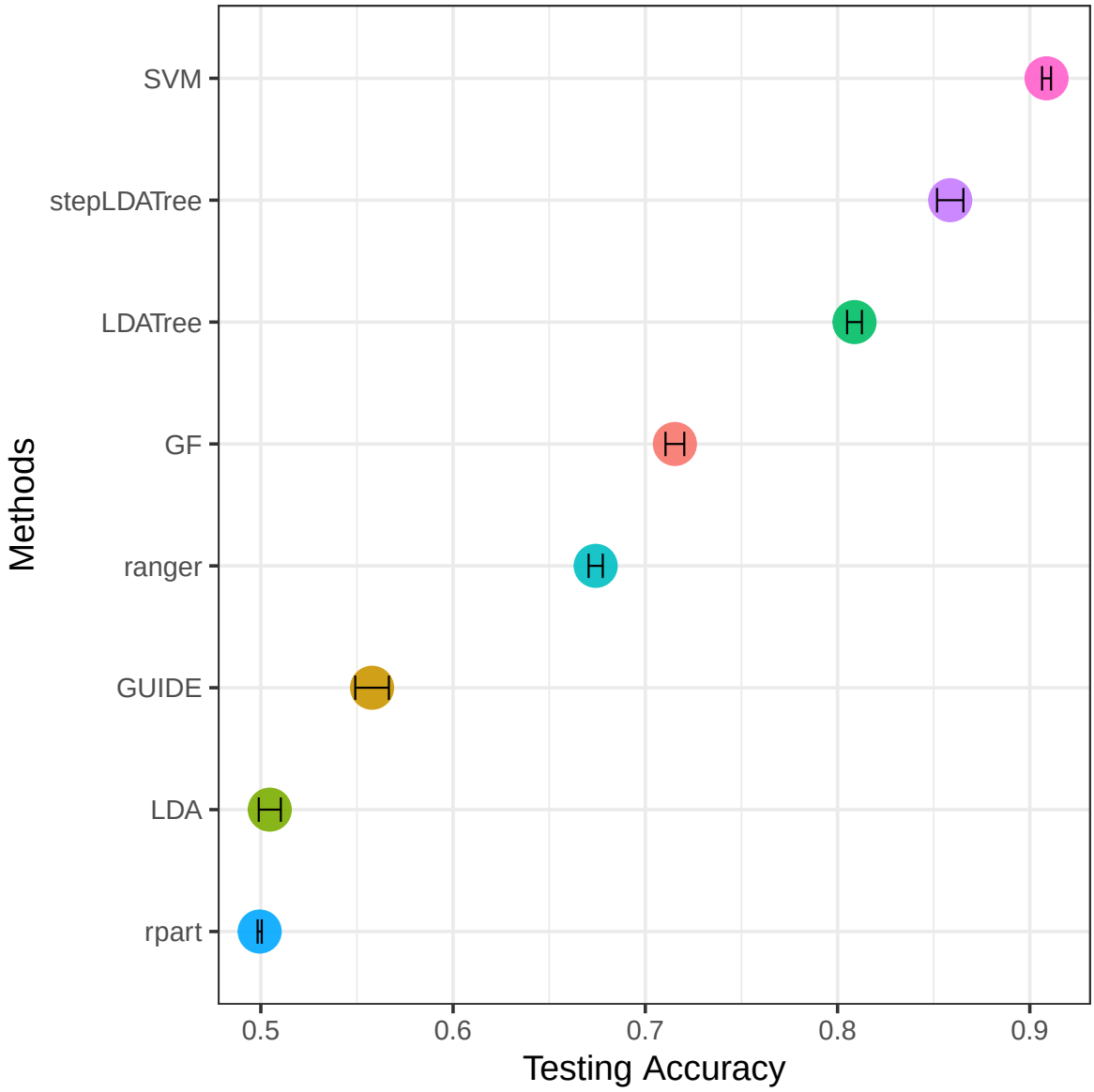

Figure 5.5: The testing results (5D XOR) in Section 5.2. Methods are ordered by their accuracies, with confidence intervals for accuracies shown in 2SD error bars.

SVM has the highest testing accuracy, followed by stepLDATree and LDATree. Single-tree methods like `rpart` and `GUIDE` have the lowest testing accuracies. Forest methods like `ranger` and `GF` are in the middle. We believe that for problems requiring proactive searching, stepLDATree and LDATree perform better compared to random forest and single-tree methods.

6 REAL DATA ANALYSIS

The purpose of this section is to validate the performance of stepLDATree and LDATree using real datasets and to compare them against existing methods. First, we show the average performance across all datasets. Then, we examine specific datasets and discuss the potential use cases for stepLDATree and LDATree. For LDATree and stepLDATree, we use the pre-pruning stopping rule in this section.

6.1 Performance Across 49 Datasets

Once again, we use the same datasets, method processing, and graphing techniques as in Section 3.1. Details about the other methods used for comparison can be found in Table 6.1. Note that for methods that cannot handle missing values, we apply the root-node imputation, as mentioned in Section 4.4. The final results are summarized in Figure 6.1.

Among all machine learning methods, the random forest (`ranger`) achieves the highest testing accuracy. GUIDE forest (`GF`) and stepLDATree form the second tier in terms of testing accuracy, but stepLDATree is much faster. LDATree and `GUIDE`

method	description
LDA	The LDA/GSVD method, which is introduced in Section 3.1
stepLDA	The stepwise LDA/GSVD method, which is introduced in Section 3.2.
ctree	Conditional tree method Hothorn et al. (2006) from the R package `partykit`.
rpart	CART method from the R package `rpart`.
GUIDE	GUIDE classification tree, introduced in Section 4
GF	GUIDE forest, the forest version of the GUIDE classification tree.
ranger	The random forest method from the R package `ranger`.

Table 6.1: Descriptions of the methods tested in Section 6.1.

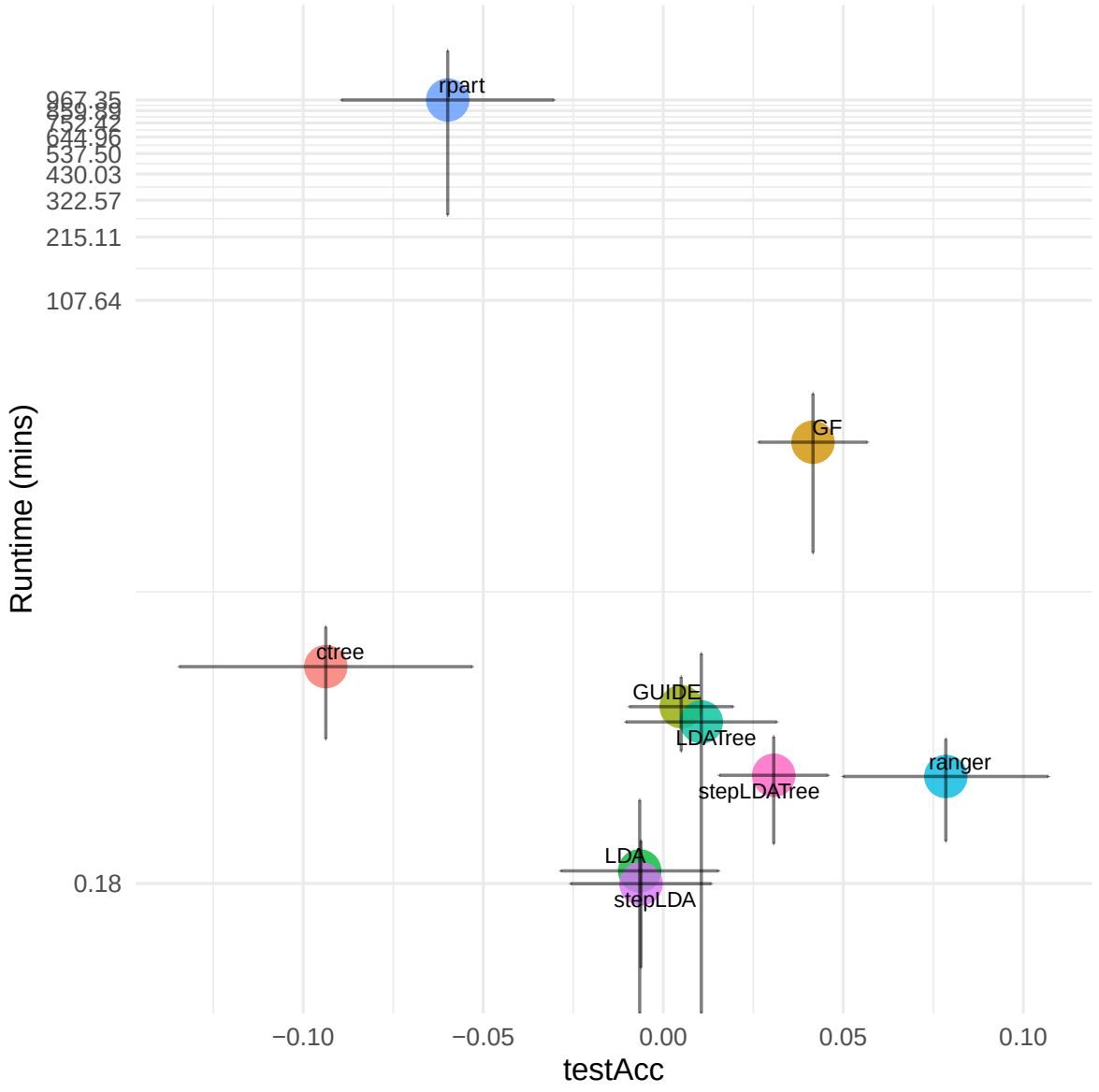

Figure 6.1: Real data analysis results in Section 6.1 using 49 datasets: Average runtime vs. average testing accuracy. Confidence intervals for runtimes (on a logarithmic scale) and standardized accuracies are presented with 2SD error bars.

have similar performance in both runtime and testing accuracy. On the other side, `rpart` and `ctree` have poor testing accuracy, with `rpart` being particularly slow on certain datasets. Based on these results, we conclude that stepLDATree, on average, outperforms all other single-tree methods but is not as effective as the random forest.

6.2 Use Case for LDATree

Among the 49 tested datasets, there are some datasets where LDATree has better performance compared to the random forest. We list three of them below and illustrate the potential use case for LDATree.

fishcatch–Species It is introduced in Section 2.2.

Arcene–Class The data set, obtained from the UC Irvine Machine Learning Repository (`https://archive.ics.uci.edu/dataset/167/arcene`), contains ten thousand mass-spectrometry measurements from 200 patients. Our task is to use those features to separate cancer patients from healthy patients.

Digits_CART–Digit This dataset, along with the following descriptions, is borrowed from Section 2.6.1 in Breiman et al. (1984). Digits are ordinarily displayed on electronic watches and calculators using seven horizontal and vertical lights in on-off combinations (see Figure 6.2). Number the lights as shown in Figure 6.3, and the scheme for all ten digits is shown in Figure 6.4. However, the data for the example are generated from a faulty calculator. Each of the seven lights has a probability of 0.1 of not doing what it is supposed to do. The final dataset contains 200 data points of ten digits, and our task is to predict the digit class given the seven lights.

Figure 6.2: Electronic digits from the Digits dataset in Section 6.2.

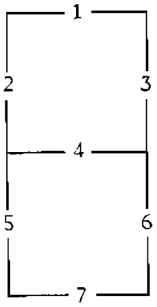

Figure 6.3: Number the electronic lights from the Digits dataset in Section 6.2.

Digit	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	0	0	1	0	0	1	0
2	1	0	1	1	1	0	1
3	1	0	1	1	0	1	1
4	0	1	1	1	0	1	0
5	1	1	0	1	0	1	1
6	1	1	0	1	1	1	1
7	1	0	1	0	0	1	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
0	1	1	1	0	1	1	1

Figure 6.4: Notation of the ten electronic digits from the Digits dataset in Section 6.2.

The results are summarized in Figure 6.5. On all three datasets, the LDATree is one of the best methods, while `ranger` seems to struggle in these cases. We suspect the potential reasons are:

- Small sample size. To make the random forest effective, one must build many trees, each requiring substantial data to approximate the pattern well enough through a non-parametric way. However, there are at most 200 observations in all three examples, which significantly limits its power. In contrast, for LDATree and stepLDATree, the tree size is adaptive, so they tend to choose simpler models when few samples are available. Additionally, since LDA is a parametric model, it does not require many data points to perform well.

- Useful interactions between variables. The shape is the most important information when identifying fish. Therefore, for the fishcatch data, knowing the length of a fish is not enough unless you also know the width. For the digits dataset, one particular light cannot fully decide which number it is unless you know all other lights. The same occurs for the Arcene data, where there are interactions between different proteins. In each split, the random forest will search for the best single variable and its cut point. However, the LDATree uses the LDA split, which is the linear combination of variables, and the decision boundary is not restricted to dichotomous splits (unless it is a binary classification problem). Sometimes, one variable might be marginally insignificant but is significant in interactions with other variables. This explains why stepLDATree is not as effective as the LDATree in the Arcene data, as stepLDATree has a variable selection step.

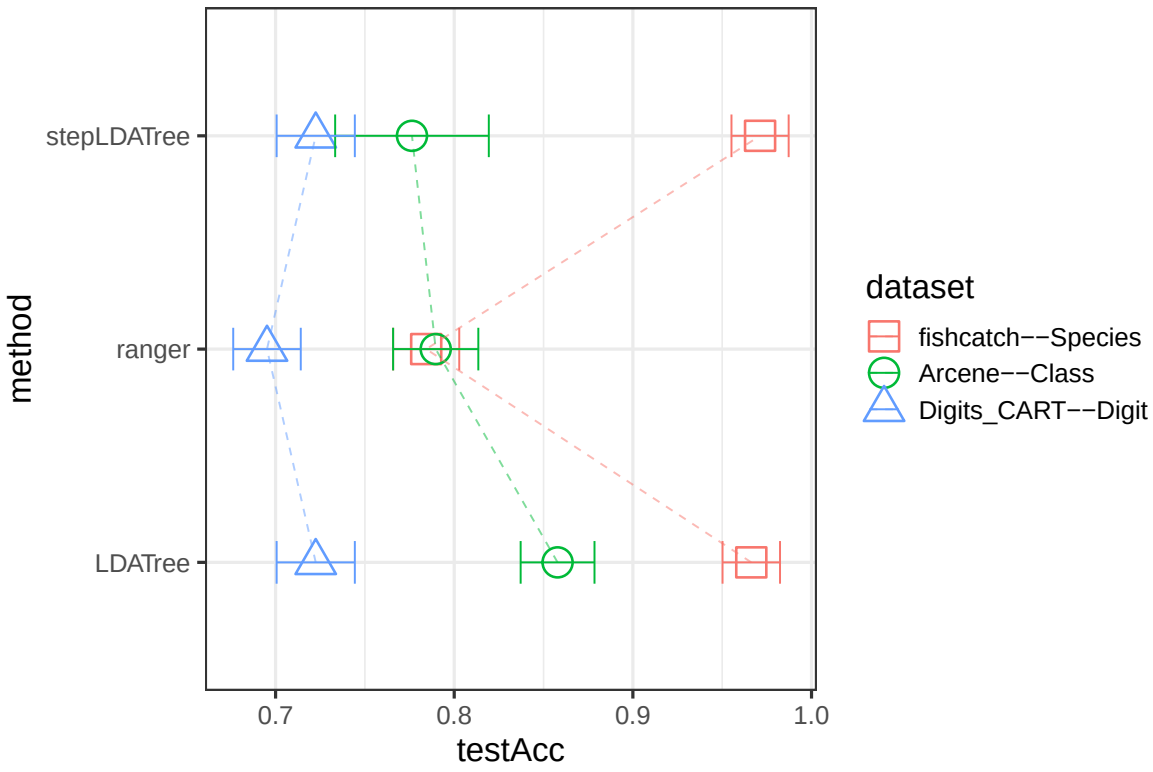

Figure 6.5: Real data analysis results in Section 6.2 on three datasets: Confidence intervals for testing accuracies are presented with 2SD error bars.

7 CONCLUSION & FUTURE WORK

In this thesis, we focus on the integration of LDA and the decision tree, introducing stepLDATree and LDATree. We aim to make them powerful single-tree classifiers that address most of the well-known weaknesses of both LDA and decision trees. Additionally, we touch on the problem of missing values, seeking the best solution within our framework. Throughout our research, the metric we care about the most is testing accuracy.

Section 3 mainly focuses on algorithm development and documents our efforts in revising and improving each part of the tree structure. We review some traditional LDA methods that can deal with the small-sample-size problem in Section 3.1, and select LDA/GSVD as our best candidate. It can retain all discriminant information without the necessity of tuning. To address the potential overfitting problem with LDA, we review the most popular stepwise LDA method in Section 3.2 and propose an improved version. This new version uses Pillai's trace instead of Wilk's Λ, solving the problem of premature stopping and better integrating with our LDA/GSVD framework. We then review several stopping rules in Section 3.3 and present a new direct-stopping rule using the p-value from the z-test of comparing pre-split and post-split training errors. We also recommend traditional cost-complexity pruning when time cost is not a concern and look-ahead splitting is needed. In Section 3.4, we explore several new splitting rules and choose the LDA split as our best candidate. The LDA split is not forced to be axis-orthogonal, making it more general. It uses the prediction region from the LDA, making it a powerful discriminative tool if we have a good LDA fit. If not, the split serves as a random split, which can break the symmetry pattern and facilitate subsequent splits. Additionally, the LDA split is fast since there is no searching involved, and it has already been fitted in the previous step.

Section 4 is about missing values. We explored new missing value solutions and

applied them, along with several well-known missing value solutions, to our stepLDATree and LDATree models. We found that the simple missing value solution is sufficient in terms of testing accuracy. We prove in Section 4.4 that simple imputation at the root node is equivalent to node-wise simple imputation within the LDATree framework. Section 5 presents results from synthetic datasets, illustrating the methods' performance and providing use cases for stepLDATree. In Section 6, we analyze real datasets and present use cases for LDATree.

We claim that both LDATree and stepLDATree generally outperform single-tree classifiers in terms of testing accuracy. Compared to ensemble methods like random forests, there are potential use cases where both methods may outperform. StepLDATree is preferable when there are many noise variables or when look-ahead splitting is needed. LDATree is more effective when the sample size is small and there are interactions between variables. Overall, stepLDATree is more accurate, more general, and can handle more scenarios, while LDATree might be a better choice when noise variables are not a concern and the sample size is not large.

Next, we discuss the limitations from two perspectives. First, regarding the algorithm itself, each part can be improved. The LDA/GSVD in Section 3.1 has decent accuracy, but it can be very slow when there are many columns. Another notorious problem with LDA is its sensitivity to outliers. Currently, this is not a huge problem since the tree structure helps mitigate it, but it would be better if LDA could handle this independently. The splitting rule in Section 3.4 might be modified to make the decision boundary clearer and more intuitive, unlike those in Figure 5.1 and Figure 5.4. Value grouping might be helpful. Additionally, LDA only considers class centroids, and decision boundaries are calculated solely based on the normality assumption. Borrowing ideas from SVM to make it more data-driven could be helpful. In the future, we can integrate importance scores into our algorithm since stepLDATree and stepwise LDA have generated them implicitly.

Second, from the dataset perspective, collecting additional datasets is necessary, as the current set of 49 datasets may not provide sufficient diversity or representation. It is important to have data with various missing patterns. It will be interesting to see if our current solution remains effective with datasets having diverse MAR patterns.

A SUPPLEMENTARY DATA DESCRIPTION

dataset	colName	rowSize	colSize	nLevels	maxProp	missing
CE	EARNCOMP	4693	916	8	0.274	Yes
CE	REF_RACE	4693	914	6	0.888	Yes
CE	[B]REF_RACE	1051	889	6	0.5	Yes
NHTSA	TRANSM	3270	920	9	0.412	Yes
NHTSA	COLMEC	3028	920	9	0.603	Yes
NHTSA	ENGINE	3273	911	18	0.382	Yes
COVID	DeathIntube	54313	48	2	0.792	Yes
COVID	[B]DeathIntube	22606	47	2	0.5	Yes
ACS	CIT	79060	114	5	0.2	Yes
ACS	RELSHIPP	49039	99	19	0.053	Yes
NHDS	marital.status	4374	59	6	0.167	No
NHDS	admission.type	29235	61	5	0.2	No
NHDS	discharge.status	10829	59	7	0.143	No
birthwt	DBWT	90000	139	3	0.33	Yes
iris	Species	150	4	3	0.33	No
fishcatch	Species	159	7	7	0.352	Yes
DummyMatrix	Class	2000	20	10	0.1	No
DummyMatrix[M]	Class	2000	20	10	0.1	Yes
Vowel	y	990	10	11	0.09	No
PrelimSim1	response	16000	87	8	0.125	Yes
PrelimSim2	response	16000	4	8	0.125	Yes
Internet	adOrNot	3279	1558	2	0.86	Yes
Internet	[B]adOrNot	919	1470	2	0.5	Yes
Indoor	Space	21048	522	124	0.053	No
Indoor	BF	21048	522	13	0.131	No
Parkinson	class	756	753	2	0.746	No
Parkinson	[B]class	384	753	2	0.5	No
Arcene	Class	200	9961	2	0.56	No
peptide	bind	310	254	2	0.584	No
Digits_CART	Digit	200	7	10	0.14	No
golubdata	Y	72	3571	2	0.653	No
CapitalOne	dissatisfied	22242	25	2	0.852	Yes
CapitalOne	[B]dissatisfied	6576	25	2	0.5	Yes
DummyAndSim1	response	16000	28	9	0.498	No
spam	spam	4601	57	2	0.606	No
simSurface2	Y	5000	10	2	0.5	No
simSurface4	Y	5000	10	4	0.25	No
jobSatis	DV_JobSatis	50225	74	5	0.466	Yes
Diabetes	Diabetes_binary	70692	21	2	0.5	No
SUPPORT2	death	9105	64	2	0.681	Yes
SUPPORT2	hospdead	9105	64	2	0.741	Yes
MI	ZSN	1700	111	2	0.768	Yes
MI	LET_IS	1700	111	8	0.841	Yes
APS_Failure	class	2750	169	2	0.5	Yes
SECOM	result	208	468	2	0.5	Yes
Motion_Capture	Class	78095	39	5	0.21	Yes
Polish_bankruptcy	class	4182	64	2	0.5	Yes
Mice_Protein	class	1080	77	8	0.139	Yes
Diabetes130	readmitted	34071	1983	3	0.33	Yes

Table A.1: Data description for the dataset used in Section 3.1. Here, `nLevels` represents the number of levels of the response variable, and `maxProp` represents the proportion of the plurality class in the response variable.

dataset	colName	rowSize	colSize	nLevels	maxProp	naProp	naAnyProp
CE	EARNCOMP	4693	916	8	0.274	0.148	1
CE	REF_RACE	4693	914	6	0.888	0.146	1
CE	[B]REF_RACE	1051	889	6	0.5	0.147	1
NHTSA	TRANSM	3270	920	9	0.412	0.093	1
NHTSA	COLMEC	3028	920	9	0.603	0.093	1
NHTSA	ENGINE	3273	911	18	0.382	0.093	1
ACS	CIT	79060	114	5	0.2	0.31	1
ACS	RELSHIPP	49039	99	19	0.053	0.366	1
MI	ZSN	1700	111	2	0.768	0.084	1
MI	LET_IS	1700	111	8	0.841	0.084	1
SECOM	result	208	468	2	0.5	0.047	1
Motion_Capture	Class	78095	39	5	0.21	0.328	1
COVID	[B]DeathIntube	22606	47	2	0.5	0.075	0.988
COVID	DeathIntube	54313	48	2	0.792	0.075	0.986
SUPPORT2	death	9105	64	2	0.681	0.115	0.966
SUPPORT2	hospdead	9105	64	2	0.741	0.115	0.966
APS_Failure	class	2750	169	2	0.5	0.129	0.957
birthwt	DBWT	90000	139	3	0.33	0.035	0.894
Polish_bankruptcy	class	4182	64	2	0.5	0.019	0.663
fishcatch	Species	159	7	7	0.352	0.069	0.553
jobSatis	DV_JobSatis	50225	74	5	0.466	0.028	0.508
Mice_Protein	class	1080	77	8	0.139	0.017	0.489
Internet	adOrNot	3279	1558	2	0.86	0.001	0.281
CapitalOne	dissatisfied	22242	25	2	0.852	0.003	0.079
CapitalOne	[B]dissatisfied	6576	25	2	0.5	0.003	0.078

Table A.2: Data description for the dataset used in Section 4.3, sorted by `naAnyProp`. Here, `nLevels` represents the number of levels of the response variable, `maxProp` represents the proportion of the plurality class in the response variable, `naProp` represents the proportion of the missing entries across all entries, and `naAnyProp` is defined as one minus the proportion of complete cases.

REFERENCES

Belhumeur, Peter N., Joao P Hespanha, and David J. Kriegman. 1997. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence* 19(7):711–720.

Blaser, Rico, and Piotr Fryzlewicz. 2016. Random rotation ensembles. *The Journal of Machine Learning Research* 17(1):126–151.

Breiman, L, JH Friedman, R Olshen, and CJ Stone. 1984. Classification and Regression Trees.

Breiman, Leo. 2001. Random forests. *Machine learning* 45:5–32.

Chen, Li-Fen, Hong-Yuan Mark Liao, Ming-Tat Ko, Ja-Chen Lin, and Gwo-Jong Yu. 2000. A new LDA-based face recognition system which can solve the small sample size problem. *Pattern recognition* 33(10):1713–1726.

Clemmensen, Line, Daniela Witten, Trevor Hastie, and Bjarne Ersbøll. 2011. Sparse discriminant analysis. *Technometrics* 406–413.

Dixon, Wilfrid Joseph. 1990. *Bmdp statistical software manual: to accompany the 1990 software release*, vol. 1. University of California Press.

Esposito, Floriana, Donato Malerba, Giovanni Semeraro, and J Kay. 1997. A comparative analysis of methods for pruning decision trees. *IEEE transactions on pattern analysis and machine intelligence* 19(5):476–491.

Friedman, Jerome H. 1989. Regularized discriminant analysis. *Journal of the American Statistical Association* 84(405):165–175.

Fukunaga, Keinosuke. 2013. *Introduction to statistical pattern recognition*. Elsevier.

Gao, Hui, and James W Davis. 2006. Why direct LDA is not equivalent to LDA. *Pattern Recognition* 39(5):1002–1006.

Habbema, JDF, and J Hermans. 1977. Selection of variables in discriminant analysis by F-statistic and error rate. *Technometrics* 19(4):487–493.

Hastie, Trevor, Rahul Mazumder, Jason D Lee, and Reza Zadeh. 2015. Matrix completion and low-rank SVD via fast alternating least squares. *The Journal of Machine Learning Research* 16(1):3367–3402.

Hastie, Trevor, Robert Tibshirani, and Andreas Buja. 1994. Flexible discriminant analysis by optimal scoring. *Journal of the American Statistical Association* 1255–1270.

Hermans, J, and JDF Hobbema. 1976. *Manual for the alloc discriminant analysis programs: A package of fortran computer programs*. University of Leiden, Department of Medical Statistics.

Hothorn, Torsten, Kurt Hornik, and Achim Zeileis. 2006. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics* 15(3):651–674.

Howland, Peg, Moongu Jeon, and Haesun Park. 2003. Structure preserving dimension reduction for clustered text data based on the generalized singular value decomposition. *SIAM Journal on Matrix Analysis and Applications* 25(1):165–179.

IBM Corp. 2021. IBM SPSS Statistics for Windows, Version 28.0. *Armonk, NY: IBM Corp*.

Jennrich, Robert I. 1977. Stepwise discriminant analysis. *Statistical methods for digital conputers* 76.

Ji, Shuiwang, and Jieping Ye. 2008. Generalized linear discriminant analysis: a unified framework and efficient model selection. *IEEE Transactions on Neural Networks* 19(10): 1768–1782.

Jombart, Thibaut, Sébastien Devillard, and François Balloux. 2010. Discriminant analysis of principal components: a new method for the analysis of genetically structured populations. *BMC genetics* 11(1):1–15.

Kim, Hyunjoong, and Wei-Yin Loh. 2003. Classification trees with bivariate linear discriminant node models. *Journal of Computational and Graphical Statistics* 12(3):512–530.

Ling, Charles X, Qiang Yang, Jianning Wang, and Shichao Zhang. 2004. Decision trees with minimal costs. In *Proceedings of the twenty-first international conference on machine learning*, 69.

Loh, Wei-Yin. 2002. Regression tress with unbiased variable selection and interaction detection. *Statistica Sinica* 361–386.

Loh, Wei-Yin, and Yu-Shan Shih. 1997. Split selection methods for classification trees. *Statistica sinica* 815–840.

Loh, Wei-Yin, and Nunta Vanichsetakul. 1988. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association* 83(403): 715–725.

Mai, Qing. 2013. A review of discriminant analysis in high dimensions. *Wiley Interdisciplinary Reviews: Computational Statistics* 5(3):190–197.

McCabe, George P. 1975. Computations for variable selection in discriminant analysis. *Technometrics* 17(1):103–109.

Mingers, John. 1987. Expert systems—rule induction with statistical data. *Journal of the operational research society* 38:39–47.

———. 1989. An empirical comparison of pruning methods for decision tree induction. *Machine learning* 4:227–243.

Niblett, Tim, and Ivan Bratko. 1987. Learning decision rules in noisy domains. In *Proceedings of expert systems' 86, the 6th annual technical conference on research and development in expert systems iii*, 25–34.

Padmanabhan, Mukund, Lalit R Bahl, and David Nahamoo. 1999. Partitioning the feature space of a classifier with linear hyperplanes. *IEEE Transactions on Speech and Audio Processing* 7(3):282–288.

Park, Haesun, Barry L Drake, Sangmin Lee, and Cheong Hee Park. 2007. Fast linear discriminant analysis using qr decomposition and regularization. Tech. Rep., Georgia Institute of Technology.

Pillai, KC Sreedharan. 1955. Some new test criteria in multivariate analysis. *The Annals of Mathematical Statistics* 117–121.

Quinlan, J. Ross. 1986. Induction of decision trees. *Machine learning* 1:81–106.

———. 1987. Simplifying decision trees. *International journal of man-machine studies* 27(3):221–234.

Quinlan, J Ross. 1993. C4.5: Programs for machine learning. *The Morgan Kaufmann Series in Machine Learning*.

Rainforth, Tom, and Frank Wood. 2015. Canonical correlation forests. *arXiv preprint arXiv:1507.05444*.

Rencher, Alvin C, and WF Christensen. 2002. *Methods of multivariate analysis*. 3rd ed. John Wiley & Sons.

Rencher, Alvin C, and Steven F Larson. 1980. Bias in Wilks' Λ in stepwise discriminant analysis. *Technometrics* 22(3):349–356.

SAS Institute Inc. 2014. SAS/ETS$^{\circledR}$ 13.2 User's Guide: High-Performance Procedures. *Cary, NC: SAS Institute Inc.*

Stekhoven, Daniel J, and Peter Bühlmann. 2012. MissForest—non-parametric missing value imputation for mixed-type data. *Bioinformatics* 28(1):112–118.

Tharwat, Alaa, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien. 2017. Linear discriminant analysis: A detailed tutorial. *AI communications* 30(2):169–190.

Tomita, Tyler M, James Browne, Cencheng Shen, Jaewon Chung, Jesse L Patsolic, Benjamin Falk, Carey E Priebe, Jason Yim, Randal Burns, Mauro Maggioni, et al. 2020. Sparse projection oblique randomer forests. *The Journal of Machine Learning Research* 21(1):4193–4231.

Venables, W. N., and B. D. Ripley. 2002. *Modern applied statistics with s*. 4th ed. New York: Springer. ISBN 0-387-95457-0.

Witten, Daniela M, and Robert Tibshirani. 2011. Penalized classification using Fisher's linear discriminant. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73(5):753–772.

Ye, Jieping, Ravi Janardan, Cheong Hee Park, and Haesun Park. 2004. An optimization criterion for generalized discriminant analysis on undersampled problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26(8):982–994.

Ye, Jieping, and Tao Xiong. 2006. Null space versus orthogonal linear discriminant analysis. In *Proceedings of the 23rd international conference on machine learning*, 1073–1080.

Ye, Jieping, and Bin Yu. 2005. Characterization of a family of algorithms for generalized discriminant analysis on undersampled problems. *Journal of Machine Learning Research* 6(4).

Yu, Hua, and Jie Yang. 2001. A direct LDA algorithm for high-dimensional data—with application to face recognition. *Pattern recognition* 34(10):2067–2070.