Sparse Discriminant Analysis with LASSO in Ultra-High Dimensions

Names:

Yunzhao Li

Instructor: Dehan Kong

STA315H5 S2025

Advanced Statistical Learning

Contents

1 Introduction **2**

2 Existing Methods **3**

2.1 Nearest Shrunken Centroids (NSC) 3

2.2 Feature Annealed Independence Rule (FAIR) 3

2.3 ℓ_1-Constrained Fisher Discriminant 3

2.4 ℓ_1-Penalized LDA 4

2.5 Motivation for SDA 4

3 Our Method **5**

3.1 Overview 5

3.2 Least-Squares Formulation of LDA 5

3.3 LASSO-Penalized Discriminant Function 6

3.4 Connection to Bayes Optimality 7

4 Simulation **8**

5 Results **10**

6 Conclusions **11**

7 References **12**

8 Appendix **13**

8.1 Codes 13

1 Introduction

For high-dimensional classification with way more features (p) than samples (n), it usually causes problems for traditional methods like Linear Discriminant Analysis (LDA), due to the difficulty of estimating a covariance matrix in this scenario. Some new methods that assume only few features are useful, still don't work well when there's too much noise. So, it is important to focus on how to select the few features that actually help with classification.

Some existing methods, such as Nearest Shrunken Centroids (NSC) and FAIR assume that features are independent. They can work for things like gene data, but ignoring how features are correlated might miss important patterns. Other methods try adding a penalty to LDA, so that they can include the correlation. However, considering the methods from Witten and Tibshirani (2011) or Wu et al. (2008), they are with great computational expenses and the optimization problems are still hard to solve.

To fix these problems, Mai, Zou, and Yuan (2012) proposed a method called Sparse Discriminant Analysis (SDA). The idea is about turning LDA into a regression problem and using LASSO to select the important features. In this paper, we'll first introduce some common methods with their pros and cons. Then, we'll explain how SDA works and how it connects to LDA and the Bayes rule. Finally, we'll repeat some simulations as the original paper to compare SDA with other methods, and show its advantages.

2 Existing Methods

2.1 Nearest Shrunken Centroids (NSC)

Tibshirani et al., 2002 (NSC): This method finds the average value of each feature for each class, then shrinks the difference between them. If the difference between the class means for a feature is small, it sets it to zero. Only the features with large differences are kept. It assumes that features are independent by using a diagonal covariance matrix.

Pros: It's simple and works well when there are lots of features. It also does feature selection automatically.

Cons: It can miss important combinations of features that are only useful when considered together, and also keep features that are similar to each other. This makes it less accurate when it comes to feature correlation.

2.2 Feature Annealed Independence Rule (FAIR)

Fan & Fan, 2008: This approach relies on a naive Bayes independence model. It ranks features by a marginal statistic and includes them in a classifier in order of importance. The classification rule is with a weighted voting:

$$\hat{y} = \text{sign}\Big(\sum_{j=1}^{p} w_j x_j\Big)$$

where weights w_j are nonzero for selected features only.

Pros: Simple and efficient. It uses one feature at a time to make decisions, so it avoids the difficulty of estimating the full covariance matrix.

Cons: It assumes that features are independent. FAIR might miss signals that only show up when features work together.

2.3 ℓ_1-Constrained Fisher Discriminant

This method adds sparsity to the discriminant direction β in the classical LDA setting. By applying Fisher's criterion, they target to find the direction that separates the classes by maximizing the ratio of between-class to within-class variance. The ℓ_1 constraint forces most entries in β to be zero, so only a few features are used for classification.

$$\min_{\beta} \ \beta^T \hat{\Sigma} \beta \quad \text{subject to } (\hat{\mu}_2 - \hat{\mu}_1)^T \beta = 1, \ \|\beta\|_1 \leq \tau$$

Pros: This method makes use of the correlation between features. It is similar to Fisher's LDA idea but

with a sparse setup.

Cons: It doesn't work well when the number of features p is much larger than n. Also, the performance depends heavily on the choice of τ.

2.4 ℓ_1-Penalized LDA

This method tries to build a sparse discriminant direction, similar to Wu's method, but it adds a penalty term to the Fisher criterion.

$$\max_{\beta} \ \beta^T B\beta - \lambda \sum_j |s_j \beta_j| \quad \text{subject to } \beta^T \hat{\Sigma} \beta \leq 1$$

Pros: This method tries to balance two goals: separating the classes and keeping the model simple. It uses a penalty to avoid overfitting.

Cons: This optimization is non-convex, thus the solution might not be the best one possible. It can pick too many noisy features when the features were correlated.

2.5 Motivation for SDA

Considering the Cons of existing methods, we want a classifier that can take into account feature correlations, keeps things sparse and works well even when the number of features is extremely large. Mai et al. designed SDA to meet all these goals.

3 Our Method

3.1 Overview

The SDA method from Mai et al. (2012) tries to solve classification by turning it into a regression problem. It adds a LASSO penalty to make the solution sparse, which only keeps the most important features. The method finds a direction to separate the two classes, which is similar to the idea of Bayes' rule. The paper also shows that SDA still works well when there are way more features than samples, and even when many features are correlated. It can find the key features and the right separating direction with different simulation settings.

3.2 Least-Squares Formulation of LDA

The main idea behind SDA is that in two-class classification, we can think of LDA as just solving a least-squares regression, by attaching the class labels with particular values:

$$y_i = \begin{cases} -\frac{n}{n_1} & \text{if } G_i = 1, \\ \\ \frac{n}{n_2} & \text{if } G_i = 2, \end{cases}$$

where n_1, n_2 are the sample sizes in each class and $n = n_1 + n_2$. The unpenalized least-squares solution for regressing y on x is thus related to the LDA direction.

Mai et al. show that if $(\beta_0^{\text{ols}}, \beta^{\text{ols}})$ minimizes the residual sum of squares:

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \mathbf{x}_i^T\beta\right)^2,$$

which is basically the same as the LDA direction, except scaled by some constant c.

$$\beta^{\text{ols}} = c\,\hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$$

The regression is trying to find a weight vector aligned with different class means. By solving an ordinary least squares (OLS) regression of this y on X, we can get the same discriminant direction as classical LDA.

3.3 LASSO-Penalized Discriminant Function

SDA solves the following arguments to estimate the sparse discriminant vector $\hat{\beta}$:

$$(\hat{\beta}_0, \hat{\beta}) = \underset{\beta_0,\, \beta}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_0 - \mathbf{x}_i^T \beta)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

where $\lambda > 0$ is a regularization parameter controlling sparsity. The first term is the mean squared error, and the second term is the LASSO penalty $\lambda \|\beta\|_1 = \lambda \sum_j |\beta_j|$.

This formulation is a standard LASSO regression problem, which can be solved via coordinate descent in `glmnet` package.

With $\lambda > 0$, we obtain a **lassoed discriminant**. That is a subset of features with nonzero weights, chosen to best predict class labels in a least-squares sense. The LASSO penalty shrinks small regression coefficients to zero.

After solving the penalized regression, we obtain $\hat{\beta}_0$ (intercept) and $\hat{\beta}$. These define the **linear discriminant function**:

$$\delta(x) = \hat{\beta}_0 + x^T \hat{\beta} \ ,$$

and the **classification rule** becomes:

$$\text{Classify } x \text{ as } G = 2 \iff x^T \hat{\beta} + \hat{\beta}_0 > 0 \ . \qquad \text{(DecisionRule)}$$

$\hat{\beta}_0$ must account for class priors. From the original paper, Mai et al. derive a closed-form formula for the optimal intercept:

$$\hat{\beta}_0^{\text{opt}} = -\frac{1}{2}(\hat{\mu}_1 + \hat{\mu}_2)^T \hat{\beta} + \frac{\hat{\beta}^T \hat{\Sigma} \hat{\beta}}{(\hat{\mu}_2 - \hat{\mu}_1)^T \hat{\beta}} \ln \frac{n_2}{n_1} \ ,$$

which reduces to the familiar:

$$-\frac{1}{2}(\hat{\mu}_1 + \hat{\mu}_2)^T \hat{\beta}$$

when $n_1 = n_2$.

Overall, SDA finds a sparse direction $\hat{\beta}$ approximating the Bayes discriminant, which sets the threshold $\hat{\beta}_0$ to mimic the Bayes classifier.

3.4 Connection to Bayes Optimality

Under the LDA model, the Bayes-optimal classifier assigns x to class 2 if $(x - \frac{\mu_1+\mu_2}{2})^T\Sigma^{-1}(\mu_2 - \mu_1) + \ln(\pi_2/\pi_1) > 0$. The vector $\beta^{\text{Bayes}} = \Sigma^{-1}(\mu_2 - \mu_1)$ is the optimal discriminant direction. SDA's objective is designed to estimate this vector in a high-dimensional context. Mai et al. prove that if the Bayes rule depends on a subset A of features (i.e. $\beta_j^{\text{Bayes}} = 0$ for $j \notin A$), their estimator $\hat{\beta}$ will correctly identify A with high probability and $\hat{\beta}$ will point in the same direction as β^{Bayes}. This holds even when p grows much faster than n. Thus, SDA effectively approximates the Bayes optimal direction and improves the accuracy of that direction.

4 Simulation

To test our proposed Lassoed Sparse Discriminant Analysis (SDA), we ran simulations on six synthetic datasets. These datasets were designed to represent different types of settings, especially with varying levels of sparsity and high feature correlations. Models 1 to 4 were designed to be sparse, only a few features were actually useful for classification. Models 5 and 6 depends more on all features, but a sparse model can still work well.

Each dataset has between 100 to 400 samples and 400 to 800 features. We randomly generated class labels and used sparse linear transformations to create their means. The covariance structures we followed the exact settings of the author.

We compared six classification methods:

- **Nearest Shrunken Centroids (Tibshirani)** – shrinks class centroids and selects features based on thresholds.
- **FAIR (Fan and Fan)** – uses simple independence rules and picks features based on marginal screening.
- **Penalized LDA (Witten)** – applies an L1 penalty to find a sparse discriminant direction.
- **L1-constrained Fisher (Wu)** – uses shrinkage discriminant analysis with a penalty to force sparsity.
- **Bonferroni t-test + LDA** – selects features using p-values and then applies LDA.
- **Our Method (Lassoed SDA)** – converts the classification problem into a regression, uses LASSO via `glmnet` package. We use 5-fold cross-validation for tuning, and classify based on the sign of predictions.

With respect to six models:

- **Model 1**: $n = 100$, $p = 400$, AR(1) correlation structure with $\Sigma_{ij} = 0.5^{|i-j|}$, and 3 strong signal features.
- **Model 2**: Same Σ as Model 1, but only two features have strong signals, while the third signal is weak.
- **Model 3**: $n = 400$, $p = 800$, with correlated covariance ($\Sigma_{ij} = 0.5$ for $i \neq j$), and 5 informative features.
- **Model 4**: $n = 300$, $p = 800$, block correlation structure, where features within blocks of size 160 are correlated at 0.6.

- **Model 5**: All 800 features contribute weakly to the signal, but only the first 5 have relatively dominant effects.
- **Model 6**: Same as Model 5, but with the reduced signal-to-noise ratio.

We used bootstrap resampling 100 times for each dataset to increase the accuracy. In every run, we trained the model on the resampled training set and tested it on the remaining samples. For each method and dataset, we reported the median classification error across the 100 iterations.

5 Results

Simulation Results of Error Rates

	Tibshirani	Fan	Witten	Wu	t-test classifier	Our method
Data_1	0.0286	0.0580	0.2957	0.0667	0.0278	0.0278
Data_2	0.0256	0.1622	0.2839	0.1282	0.0263	0.0286
Data_3	0.1187	0.1157	0.2687	0.0858	0.1241	0.0207
Data_4	0.0551	0.1223	0.2651	0.1448	0.0631	0.0826
Data_5	0.0637	0.0637	0.2316	0.0450	0.0596	0.0211
Data_6	0.0828	0.0830	0.2674	0.0697	0.0784	0.0195

The simulation results show that our method performs very well among all six datasets. In sparse settings of Data 1 and Data 2 with moderate correlations, it gives slightly better accuracy compared to the t-test classifier and Tibshirani's NSC. In harder settings, such as Data 3 and Data 4, with much larger number of features, as well as greater correlation, our method clearly outperforms the others. For example, the error rates are as 0.0207 and 0.0826, while methods like Witten's and Wu's are approximately 0.26 and 0.14. Even in dense datasets like Data 5 and Data 6, our method still gives the best results, with errors of 0.0211 and 0.0195. This suggests that its overall great performance in both sparse settings and highly spread-out signal.

6 Conclusions

In this paper, we went through the Sparse Discriminant Analysis (SDA) method introduced by Mai, Zou, and Yuan (2012), and compared its simulated performance with other high-dimensional classifiers. SDA used a LASSO-penalized least squares formulation to link the classical LDA, and significantly overcomed the challenges in $p \gg n$ settings.

Compared to methods like NSC and FAIR with independence assumption, and ℓ_1-based LDA methods, SDA apparently handles correlated features much better and gives a good performance in sparse problems.

Pros: SDA works pretty well in ultra-high dimensions. It can deal with correlated features and select a small number of useful ones for interpretation. It also has strong theory to support finding the correct signals, as well as estimating the Bayes-optimal classifier when p is very large.

Cons: The method highly depends on the choice of regularization parameter λ. If λ is not chosen well, the accuracy might go down. Also, if the correlation structure is very complex, SDA might still struggle like other linear methods.

Overall, Sparse Discriminant Analysis is a reliable method for high-dimensional classification. The simulations show that it has great accuracy and feature selection in different sparse settings, typically in applications like gene expression data.

7 References

Mai, Q., Zou, H., & Yuan, M. (2012). *A direct approach to sparse discriminant analysis in ultra-high dimensions. Biometrika*, *99*(1), 29–42. https://doi.org/10.1093/biomet/asr066

Tibshirani, R., Hastie, T., Narasimhan, B., & Chu, G. (2002). *Diagnosis of multiple cancer types by shrunken centroids of gene expression. Proceedings of the National Academy of Sciences*, *99*(10), 6567–6572. https://doi.org/10.1073/pnas.082099299

Fan, J., & Fan, Y. (2008). *High-dimensional classification using features annealed independence rules. Annals of Statistics*, *36*(6), 2605–2637. https://doi.org/10.1214/07-AOS504

Wu, Y., & Lange, K. (2008). *Coordinate descent algorithms for lasso penalized regression. Annals of Applied Statistics*, *2*(1), 224–244. https://doi.org/10.1214/07-AOAS147

Witten, D. M., & Tibshirani, R. (2011). *Penalized classification using Fisher's linear discriminant. Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *73*(5), 753–772. https://doi.org/10.1111/j.1467-9868.2011.00783.x

Friedman, J., Hastie, T., & Tibshirani, R. (2010). *Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software*, *33*(1), 1–22. https://www.jstatsoft.org/v33/i01/

Fan, J., & Li, R. (2001). *Variable selection via nonconcave penalized likelihood and its oracle properties. Journal of the American Statistical Association*, *96*(456), 1348–1360. https://doi.org/10.1198/016214501753382273

8 Appendix

8.1 Codes

```r
install.packages("/Users/yunzhaoli/Downloads/penalizedLDA_1.1.tar.gz", repos = NULL, type="source")


library(flsa)
library(pamr)           # Model 1
library(sda)            # Model 2, 4, 6
library(MASS)           # LDA
library(penalizedLDA)   # Model 3
library(glmnet)
library(kableExtra)
library(knitr)



# Simulation 1
set.seed(101)
n1 <- 100
p1 <- 400
Sigma1 <- outer(1:p1, 1:p1, function(i, j) 0.5^abs(i - j))
beta1 <- c(3, 1.5, 0, 0, 2, rep(0, p1 - 5))
y1 <- sample(1:2, n1, replace = TRUE)
mu1_class1 <- rep(0, p1)
mu1_class2 <- as.vector(Sigma1 %*% beta1)
X1 <- t(sapply(y1, function(g) {
  mu <- if (g == 1) mu1_class1 else mu1_class2
  mvrnorm(1, mu, Sigma1)
}))


# Simulation 2
set.seed(102)
n2 <- 100
```

```r
p2 <- 400

Sigma2 <- Sigma1
beta2 <- c(3, 2.5, -2.8, rep(0, p2 - 3))
y2 <- sample(1:2, n2, replace = TRUE)
mu2_class1 <- rep(0, p2)
mu2_class2 <- as.vector(Sigma2 %*% beta2)
X2 <- t(sapply(y2, function(g) {
  mu <- if (g == 1) mu2_class1 else mu2_class2
  mvrnorm(1, mu, Sigma2)
}))

# Simulation 3
set.seed(103)
n3 <- 400
p3 <- 800

Sigma3 <- matrix(0.5, p3, p3)
diag(Sigma3) <- 1

beta3 <- c(3, 1.7, -2.2, -2.1, 2.55, rep(0, p3 - 5))
y3 <- sample(1:2, n3, replace = TRUE)

mu3_class1 <- rep(0, p3)
mu3_class2 <- as.vector(Sigma3 %*% beta3)

X3 <- t(sapply(y3, function(g) {
  mu <- if (g == 1) mu3_class1 else mu3_class2
  mvrnorm(1, mu, Sigma3)
}))

# Simulation 4
```

```r
set.seed(104)
n4 <- 300
p4 <- 800

Sigma_tilde <- outer(1:160, 1:160, function(i, j) 0.6^abs(i - j))
Sigma4 <- kronecker(diag(5), Sigma_tilde)

beta4 <- c(1.2, -1.4, 1.15, -1.64, 1.5, 1, 2, rep(0, p4 - 7))
y4 <- sample(1:2, n4, replace = TRUE)

mu4_class1 <- rep(0, p4)
mu4_class2 <- as.vector(Sigma4 %*% beta4)

X4 <- t(sapply(y4, function(g) {
  mu <- if (g == 1) mu4_class1 else mu4_class2
  mvrnorm(1, mu, Sigma4)
}))

# Simulation 5
set.seed(105)
n5 <- 400
p5 <- 800

Sigma5 <- Sigma3

beta5 <- c(3, 1.7, -2.2, -2.1, 2.55, rep(1/(p5 - 5), p5 - 5))
y5 <- sample(1:2, n5, replace = TRUE)

mu5_class1 <- rep(0, p5)
mu5_class2 <- as.vector(Sigma5 %*% beta5)

X5 <- t(sapply(y5, function(g) {
```

```r
  mu <- if (g == 1) mu5_class1 else mu5_class2
  mvrnorm(1, mu, Sigma5)
}))

# Simulation 6
set.seed(106)
n6 <- 400
p6 <- 800

Sigma6 <- Sigma3
beta6 <- beta5
y6 <- sample(1:2, n6, replace = TRUE)

mu6_class1 <- rep(0, p6)
mu6_class2 <- as.vector(Sigma6 %*% beta6)

X6 <- t(sapply(y6, function(g) {
  mu <- if (g == 1) mu6_class1 else mu6_class2
  mvrnorm(1, mu, Sigma6)
}))



# Predict Function for Penalized LDA
predict.penlda <- function(object, newdata) {
  if (!inherits(object, "penlda")) stop("Not a penalizedLDA object")
  if (!is.null(attr(object$x, "scaled:center"))) {
    meanX <- attr(object$x, "scaled:center")
  } else if (!is.null(object$meanX)) {
    meanX <- object$meanX
  } else {
    stop("Cannot find training means for centering new data.")
  }
```

```r
  # Ensure matrix input
  newdata <- as.matrix(newdata)
  newdata_centered <- scale(newdata, center = meanX, scale = FALSE)

  # Project new data
  proj <- newdata_centered %*% object$discrim

  # Class centroids in projected space
  class_means_proj <- rowsum(object$xproj, object$y) / as.vector(table(object$y))

  # Handle number of test samples and classes
  num_classes <- length(unique(object$y))
  num_samples <- nrow(newdata)

  # Compute squared distances between each projected sample and each class centroid
  dists <- matrix(NA, nrow = num_samples, ncol = num_classes)
  for (k in 1:num_classes) {
    dists[, k] <- (proj - class_means_proj[k])^2
  }

  # Predict class: assign to the closest centroid
  pred_class <- sort(unique(object$y))[max.col(-dists, ties.method = "first")]
}



# Fit and Predict
set.seed(346)
Xs <- list(X1, X2, X3, X4, X5, X6)
ys <- list(y1, y2, y3, y4, y5, y6)

B <- 100
lambda_seq <- c(1e-4, 1e-3, 5e-3, 1e-2, 5e-2, 0.1)
```

```
error_matrix <- matrix(NA, nrow = 6, ncol = 6)
colnames(error_matrix) <- c("Tibshirani", "Fan", "Witten", "Wu", "t-test classifier", "Our method")
rownames(error_matrix) <- paste0("Data_", 1:6)

for (data_i in 1:6) {
  X <- Xs[[data_i]]
  y <- ys[[data_i]]
  n <- nrow(X)

  errors <- list(
    model1 = numeric(B),
    model2 = numeric(B),
    model3 = numeric(B),
    model4 = numeric(B),
    model5 = numeric(B),
    model6 = numeric(B)
  )

  cat("\nProcessing dataset", data_i, "\n")

  for (b in 1:B) {
    idx_train <- sample(1:n, n, replace = TRUE)
    idx_test <- setdiff(1:n, unique(idx_train))

    if (length(idx_test) == 0) {
      for (k in names(errors)) errors[[k]][b] <- NA
      next
    }

    Xtr <- X[idx_train, ]
    ytr <- y[idx_train]
    Xte <- X[idx_test, ]
```

```r
yte <- y[idx_test]

# Model 1: Nearest Shrunken Centroids (Tibshirani)
dat <- list(x = t(Xtr), y = ytr)
m1 <- pamr.train(dat)
pred1 <- pamr.predict(m1, t(Xte), threshold = 1)
errors$model1[b] <- mean(pred1 != yte)

# Model 2: FAIR (Fan)
m2 <- sda(Xtr, ytr, diagonal = TRUE)
pred2 <- predict(m2, Xte)$class
errors$model2[b] <- mean(pred2 != yte)

# Model 3: Penalized LDA (Witten)
meanXtr <- colMeans(Xtr)
Xtr_centered <- scale(Xtr, center = meanXtr, scale = FALSE)
m3 <- PenalizedLDA(Xtr_centered, ytr, lambda = 0.01, K = 1)
m3$meanXtr <- meanXtr
Xte_centered <- scale(Xte, center = meanXtr, scale = FALSE)
pred3 <- predict(m3, Xte_centered)
errors$model3[b] <- mean(pred3 != yte)

# Model 4: L1-constrained Fisher (Wu)
m4 <- sda(Xtr, ytr, lambda = 1e-3)
pred4 <- predict(m4, Xte)$class
errors$model4[b] <- mean(pred4 != yte)

# Model 5: Bonferroni t-test + LDA (t-test classifier)
pvals <- apply(Xtr, 2, function(f) t.test(f ~ ytr)$p.value)
padj <- p.adjust(pvals, method = "bonferroni")
selected <- which(padj < 0.05)
```

```r
  if (length(selected) >= 2) {
    lda_model <- lda(Xtr[, selected], ytr)
    pred5 <- predict(lda_model, Xte[, selected])$class
    errors$model5[b] <- mean(pred5 != yte)
  } else {
    errors$model5[b] <- NA
  }

  # Model 6: Lassoed SDA
  lambda_seq <- 10^seq(-4, -1, length.out = 6)
  folds <- sample(rep(1:5, length.out = length(ytr)))  # 5-fold CV assignment

  cv_errors <- numeric(length(lambda_seq))

  for (i in seq_along(lambda_seq)) {
    lam <- lambda_seq[i]
    fold_errs <- c()

    for (fold in 1:5) {
      idx_train <- which(folds != fold)
      idx_valid <- which(folds == fold)

      X_train <- Xtr[idx_train, ]
      y_train <- ifelse(ytr[idx_train] == 1, 1, -1)  # convert to -1/+1
      X_valid <- Xtr[idx_valid, ]
      y_valid <- ytr[idx_valid]

      fit <- glmnet(X_train, y_train, alpha = 1, lambda = lam, family = "gaussian")
      y_pred <- predict(fit, newx = X_valid)
      pred_label <- ifelse(y_pred >= 0, 1, 2)

      fold_errs <- c(fold_errs, mean(pred_label != y_valid))
```

```
      }

      cv_errors[i] <- mean(fold_errs)
    }

    best_lambda <- lambda_seq[which.min(cv_errors)]

    # Fit final model on full training set
    fit_final <- glmnet(Xtr, ifelse(ytr == 1, 1, -1), alpha = 1, lambda = best_lambda, family = "gaussi
    pred_vals <- predict(fit_final, newx = Xte)
    pred_labels <- ifelse(pred_vals >= 0, 1, 2)

    errors$model6[b] <- mean(pred_labels != yte)
  }
  error_matrix[data_i, ] <- sapply(errors, function(x) median(x, na.rm = TRUE))
}

print(round(error_matrix, 4))
kable(round(error_matrix, 4), align = "c", caption = "Simulation Results of Error Rates") %>%
  kable_styling(full_width = FALSE, position = "center", font_size = 14) %>%
  row_spec(0, bold = TRUE, align = "c")  # center and bold headers
```