Faculty of Mathematics and Information Sciences

WARSAW UNIVERSITY OF TECHNOLOGY

Deliverable 3

Adapting the gips library for classification problem utilizing discriminant analysis – gipsDA

Authors:
Norbert Frydrysiak
Antoni Kingston

Supervisors:
MSc Eng. Adam Chojecki
PhD Bartosz Kołodziejek, Assoc. Prof.

Version 1.0

November 26, 2025

Contents

1 Abstract

This work details the implementation of the `gipsDA` project, transitioning a previously proposed theoretical framework into a functional R package. The software architecture is presented, focusing on two core components: a user-facing `models` module that provides a familiar interface by extending the functionality of the `MASS` package, and a backend `gipsmult` module containing the novel logic for identifying common symmetries. Furthermore, the final model-building API is specified, including user-configurable parameters that allow for the selection between two covariance estimation strategies: one based on the single most probable permutation (argmax) and an alternative based on a weighted average across the posterior distribution. The purpose is to provide a comprehensive overview of the implemented software, demonstrating its readiness for the experimental validation phase and serving as a definitive record of the project's construction.

2 History of changes

Authors	Description	Version	Date
Norbert Frydrysiak	Initial draft of the document structure and content outline.	**0.1**	**19.11.2025**
Norbert Frydrysiak, Antoni Kingston	Added a detailed description of the 'Testing Environment' Section.	**0.2**	**20.11.2025**
Norbert Frydrysiak	Added a reference to deliverable 2.	**0.4**	**21.11.2025**
Norbert Frydrysiak	Created sections: 'Modules implementation' and 'Build Model'.	**0.6**	**22.11.2025**
Norbert Frydrysiak, Antoni Kingston	Modified sections: 'Modules implementation', 'Build Model' and 'References to deliverable 1, 2'.	**1.0**	**25.11.2025**

3 References to deliverable 1, 2

A key extension to the methodology proposed in the previous project stage is the introduction of an alternative covariance matrix estimator. The original proposal focused primarily on an *argmax*-based estimator, which selects the covariance structure corresponding to the single most probable permutation group:

$$\hat{\Sigma} = \Sigma_{c^*}, \tag{1}$$

where

$$c^* = \underset{c}{\operatorname{argmax}}\, \mathbb{P}(\Gamma = c \mid X = x, Y = y). \tag{2}$$

In addition to this approach, we also consider a second estimator based on a weighted average over all possible permutation groups. This estimator is defined as:

$$\hat{\Sigma} = \sum_{c \in \sigma_P} \mathbb{P}(\Gamma = c \mid X = x, Y = y)\Sigma_c \tag{3}$$

This alternative model considers the entire posterior distribution of the permutation groups, potentially offering a more robust estimation by incorporating uncertainty about the true underlying symmetry. Both of these estimation strategies are readily implemented using the functionalities provided by the `gips` package.

It should be noted that these formulas are exact only when the entire space of permutations can be exhaustively searched, which is feasible in our implementation for $p \leq 9$, where p denotes the number of features (columns) in the data. For a larger number of features, the Metropolis-Hastings algorithm, as described in [1], is used to approximate these estimators, as the permutation space becomes too large to explore completely.

4 Modules implementation

The solution is implemented as an R package named `gipsDA`. Standard R package development practices are followed, including the use of the `roxygen2` package for generating documentation directly from comments in the source code. The core design philosophy was to extend the well-established and widely-used

functions of the MASS package, ensuring a familiar interface for users. Specifically, our functions heavily leverage the internal code of MASS::lda and MASS::qda, injecting the gips projection methodology at critical points of the covariance matrix estimation process. The implementation is organized into two primary components: the user-facing models module and the backend gipsmult module.

4.1 The models Module

This module contains the primary, user-facing functions: gipsLDA, gipsQDA, and gipsMultQDA. Our implementation strategy was to adopt the complete structure of the original MASS functions to ensure a consistent user experience. The gipsLDA function is a direct modification of lda.default(). The other two models, gipsQDA and its variant gipsMultQDA, share a common foundation, as both are implemented by modifying the qda.default() method. For all three functions, our changes are precisely targeted to replace the standard covariance estimation with our gips-based projection.

4.1.1 gipsLDA

This function is designed as a direct, enhanced replacement for its MASS::lda counterpart. The implementation adopts the S3 method dispatch system from MASS to provide a familiar and flexible user interface.

S3 Methods and User Interface To ensure consistency with standard R practices, gipslda is an S3 generic function with several methods that handle different input types. The gipslda.formula(), gipslda.data.frame(), and gipslda.matrix() methods are almost identical to their equivalents in the MASS package. Their primary role is to process the input data—handling formulas, subsets, and missing values—before dispatching to the core computational engine, gipslda.default(). This design ensures that users familiar with MASS::lda can use gipslda with no change to their workflow.

The Core Engine: gipslda.default() This function contains the main algorithm and is where our modifications to the MASS code are concentrated. It accepts the following arguments:

x, grouping The input matrix of predictors and the vector of class labels.

prior A vector of prior probabilities for the classes, defaulting to the class proportions in the training data.

tol A tolerance threshold to detect zero-variance columns, defaulting to 1.0e-4.

weighted_avg A logical parameter, defaulting to FALSE, which controls the covariance pooling strategy.

MAP A logical parameter, defaulting to TRUE, which selects the estimation method (argmax or weighted average).

optimizer A character string, defaulting to NULL. If NULL, it is automatically set to "BF" for $p < 10$ and "MH" for $p \geq 10$.

max_iter An integer, defaulting to NULL. If the optimizer is "MH" and this is NULL, it is set to 100 and a warning is issued.

The function's workflow begins with input validation (checking for finite values, consistent dimensions, empty groups), which is identical to the procedure in MASS::lda.default(). The key modifications occur in two stages:

1. **Covariance Pooling:** The first modification intercepts the calculation of the pooled covariance matrix. The logic proceeds based on the weighted_avg parameter:
   - If FALSE (default), the standard unbiased pooled covariance matrix is computed, as in MASS.
   - If TRUE, the code first calculates the individual covariance matrix S_g for each class and then combines them using the formula $S = \frac{1}{n} \sum_g n_g S_g$.
2. **Gips Projection:** The resulting pooled covariance matrix is then passed as a single-element list to the project_covs() helper function. This is the primary injection point of our methodology. The helper function applies the gips optimization and returns the final, projected covariance matrix.

After this step, the remainder of the function proceeds exactly as in `MASS::lda.default()`, performing Singular Value Decomposition (SVD) and calculating the discriminant function coefficients, but now using the *projected* covariance matrix.

Return Value and Post-Processing The function returns an object of class `gipslda`. This is a `list` containing the standard components from a `MASS::lda` object (e.g., `prior`, `counts`, `means`, `scaling`), plus an additional element, `optimization_info`, which stores the output from the `gips` optimization. The package provides `predict()`, `print()`, and `plot()` methods for `gipslda` objects, which are also direct adaptations of their `MASS` counterparts. This ensures that a fitted `gipslda` object can be used for prediction and visualization without any side effects beyond the standard console or plot output. The `print.gipslda()` method has been extended to display the contents of the `optimization_info` element.

4.1.2 gipsQDA

This function is designed as an enhanced replacement for `MASS::qda`, allowing for class-specific covariance matrices, each with its own unique symmetry structure. The implementation follows the S3 method dispatch system of `MASS` to maintain a familiar user interface.

S3 Methods and User Interface Consistent with standard R practices, `gipsqda` is an S3 generic function. The `gipsqda.formula()`, `gipsqda.data.frame()`, and `gipsqda.matrix()` methods are nearly identical to their counterparts in the `MASS` package. Their purpose is to preprocess the input data by handling formulas, subsets, and missing values before passing a clean data matrix and grouping factor to the core computational engine, `gipsqda.default()`.

The Core Engine: `gipsqda.default()` This function contains the main algorithm and is where our modifications to the `MASS` code are implemented. It accepts the following arguments:

`x, grouping` The input matrix of predictors and the vector of class labels.

`prior` A vector of prior probabilities for the classes, defaulting to the class proportions in the training data.

`MAP` A logical parameter, defaulting to `TRUE`, which selects the estimation method (argmax or weighted average).

`optimizer` A character string, defaulting to `NULL`. If `NULL`, it is automatically set to `"BF"` for $p < 10$ and `"MH"` for $p \geq 10$.

`max_iter` An integer, defaulting to `NULL`. If the optimizer is `"MH"` and this is `NULL`, it is set to `100` and a warning is issued.

The function's workflow begins with input validation inherited from `MASS::qda.default()`, such as checking for finite values and ensuring that each group has enough observations to estimate a covariance matrix ($n_g > p$). The key modification is introduced within the main `for` loop that iterates through each class:

1. For the current class, an empirical covariance matrix is estimated using `MASS::cov.mve()`.
2. **Gips Projection:** This single covariance matrix is then passed as a single-element list to the `project_covs()` helper function. The helper applies the `gips` optimization to find the optimal symmetry structure for this specific class and returns the projected covariance matrix.
3. The remainder of the loop proceeds with the standard `qda` logic, performing Singular Value Decomposition (SVD) on the *projected* matrix to calculate the scaling components and log-determinant for that class.

This process is repeated for every class, resulting in a model where each class has its own individually optimized covariance structure.

Return Value and Post-Processing The function returns an object of class `gipsqda`. This is a `list` containing the standard components from a `MASS::qda` object (e.g., `prior`, `counts`, `means`, `scaling`, `ldet`), plus an additional element, `optimization_info`. It should be noted that since the optimization is performed independently for each class, the final `optimization_info` element stored in the returned object corresponds to the results from the *last* class processed in the loop. The package provides `predict()` and `print()` methods for `gipsqda` objects, which are direct adaptations of their `MASS` counterparts, ensuring standard functionality for prediction and inspection. The `print.gipsqda()` method has been extended to display the contents of the `optimization_info` element.

4.1.3 gipsMultQDA

This function implements the intermediate model, which allows for class-specific covariance matrices but constrains them to share a single, common permutation symmetry. The implementation is a structural modification of `MASS::qda` and, like the other functions, uses the S3 method dispatch system.

S3 Methods and User Interface To maintain a consistent API, `gipsmultqda` is an S3 generic function. The `gipsmultqda.formula()`, `gipsmultqda.data.frame()`, and `gipsmultqda.matrix()` methods are nearly identical to their counterparts in the `MASS` package. They handle the initial data processing before dispatching to the core computational engine, `gipsmultqda.default()`.

The Core Engine: `gipsmultqda.default()` This function contains the main algorithm and is where our modifications to the `MASS` code are implemented. It accepts the following arguments:

`x, grouping` The input matrix of predictors and the vector of class labels.

`prior` A vector of prior probabilities for the classes, defaulting to the class proportions in the training data.

`MAP` A logical parameter, defaulting to `TRUE`, which selects the estimation method (argmax or weighted average).

`optimizer` A character string, defaulting to `NULL`. If `NULL`, it is automatically set to `"BF"` for $p < 10$ and `"MH"` for $p \geq 10$.

`max_iter` An integer, defaulting to `NULL`. If the optimizer is `"MH"` and this is `NULL`, it is set to `100` and a warning is issued.

The function's workflow begins with the same input validation as `MASS::qda.default()`. The core logic is then executed in three distinct stages:

1. **Covariance Collection:** A preliminary `for` loop iterates through all classes. In each iteration, it calculates the empirical covariance matrix for that class using `MASS::cov.mve()` and collects it into a list.
2. **Joint Gips Projection:** After the loop, the entire list of covariance matrices is passed to the `project_covs()` helper function. This is the primary injection point of the `gipsmult` methodology. The helper function finds a single, common symmetry structure that is jointly optimal for all classes and returns a list of the projected covariance matrices and the optimization results.
3. **SVD and Scaling:** A second `for` loop then iterates through the classes again. For each class, it takes the corresponding projected matrix from the list returned by the helper function and performs the standard SVD and scaling calculations, as in `qda.default()`.

Return Value and Post-Processing The function returns an object of class `gipsmultqda`. This is a `list` containing the standard components from a `MASS::qda` object (e.g., `prior`, `counts`, `means`, `scaling`, `ldet`), plus an additional element, `optimization_info`. This element stores the results of the joint optimization, such as the single common MAP permutation or the posterior probabilities. The package provides `predict()` and `print()` methods for `gipsmultqda` objects, which are direct adaptations of their `MASS` counterparts. The `print.gipsmultqda()` method has been extended to display the contents of the `optimization_info` element.

4.1.4 Helper Functions

A utility file (`models_utils.R`) contains the core bridge functions that connect the modified `MASS` code to the `gips` logic. The implementation was streamlined to use a single, versatile function that handles both estimation strategies.

- `project_covs(emp_covs, ns_obs, MAP, optimizer, max_iter, tol)`: This is the primary helper function that acts as a unified interface for both the argmax and weighted-average estimation methods. It takes several arguments to control the process:

  `emp_covs` A `list` of numeric `matrix` objects, where each matrix is an empirical covariance matrix for a class.

  `ns_obs` A numeric `vector` containing the number of observations for each corresponding class.

  `MAP` A logical scalar. If `TRUE`, the function finds the single most probable permutation. If `FALSE`, it calculates the weighted-average projection.

  `optimizer` A character string specifying the search algorithm, either `'BF'` or `'MH'`. The resolution of the `'auto'` option is handled by the parent model function.

  `max_iter` An integer specifying the number of iterations for the Metropolis-Hastings optimizer.

  `tol` A numeric tolerance threshold used when `MAP = FALSE`. When calculating the weighted average estimator from Equation (3), permutations with a posterior probability below this threshold are excluded from the summation to improve computational efficiency.

  The function returns a `list` containing two named elements:

  - `covs`: A `list` of numeric `matrix` objects, containing the final projected covariance matrices. This list has the same length and structure as the input `emp_covs`.
  - `opt_info`: Contains information from the optimization process. If `MAP = TRUE`, this is the optimal permutation object. If `MAP = FALSE`, this is a named numeric `vector` of the posterior probabilities used for weighting.

- `project_matrix_multiperm(emp_cov, probs)`: This is a lower-level utility that implements the weighted-average projection for a single matrix. It takes two arguments:

  `emp_cov` A single numeric `matrix`.

  `probs` A named numeric `vector` where the names are permutations and the values are their posterior probabilities.

  The function returns a single numeric `matrix` of the same dimensions as the input `emp_cov`, representing the final weighted-average covariance matrix.

4.2 The `gipsmult` Module

This module provides the novel backend logic for finding a common permutation symmetry across multiple groups of data. It is designed as a self-contained system, heavily inspired by the architecture of the original `gips` package, and is centered around a new S3 class, `gipsmult`. The module's primary purpose is to serve the `gipsMultQDA` model, but it is a fully functional component on its own.

4.2.1 Core Object and Constructor: `gipsmult()`

The central component is the `gipsmult` S3 object, which acts as a container for all necessary data. It is created by the constructor function `gipsmult()`.

Arguments and Defaults

- `Ss`: A `list` of numeric `matrix` objects, where each matrix is a covariance matrix for a class.
- `numbers_of_observations`: A numeric `vector` of integers, containing the number of observations for each corresponding class.
- `delta`: A numeric scalar, defaulting to `3`.

- `D_matrix`: A numeric `matrix` for the prior distribution, defaulting to `NULL`. If `NULL`, a default matrix is constructed based on the input data, as described in [1].
- `was_mean_estimated`: A logical scalar, defaulting to `TRUE`.

Assumptions The function assumes that all matrices in the `Ss` list are square, symmetric, positive semi-definite, and of the same dimension. It also expects that the length of the `Ss` list is equal to the length of the `numbers_of_observations` vector.

Return Value The function returns an S3 object of class `gipsmult`. This object is a `list` containing an initial permutation, with all the input data (the list of covariance matrices, observation counts, etc.) stored as attributes.

4.2.2 Optimization Engine: `find_MAP()`

The main user-facing function for running the optimization is `find_MAP()`. It takes a `gipsmult` object and searches for the permutation that maximizes the joint posterior probability.

Arguments and Defaults

- `g`: A `gipsmult` object, as created by the constructor.
- `optimizer`: A character string, defaulting to `NA`, which resolves to `"BF"` or `"MH"` based on the problem size.
- `max_iter`: An integer, defaulting to `NA`.
- `return_probabilities`: A logical scalar, defaulting to `FALSE`. If `TRUE`, the posterior probabilities of visited permutations are estimated.

Implementation Logic The function acts as a high-level dispatcher. Based on the `optimizer` argument, it calls one of the backend algorithms (`Metropolis_Hastings_optimizer`, `hill_climbing_optimizer`, or `brute_force_optimizer`) to perform the search over the permutation space.

Return Value The function returns an updated `gipsmult` object. The main list element is replaced with the best permutation found, and a new attribute, `optimization_info`, is added. This attribute is a `list` containing detailed results of the search, such as the convergence history, the algorithm used, and optionally, the posterior probabilities.

4.2.3 Objective Function: `log_posteriori_of_gipsmult()`

This function calculates the value that the `find_MAP()` optimizers aim to maximize.

Implementation Logic Technically, this function does not compute the true log-posterior probability but rather a value proportional to it (the true log-posterior shifted by a constant). This is sufficient for optimization, as the location of the maximum is unaffected. The function takes a `gipsmult` object, extracts the list of covariance matrices and observation counts, and then uses `mapply()` to call the underlying `gips:::log_posteriori_of_perm()` function for each class. The final value is the sum of these individual log-posterior terms.

Return Value The function returns a single numeric value representing the joint log-posterior score for the permutation currently stored in the `gipsmult` object.

4.2.4 User Interface and Post-Processing

To ensure a user-friendly experience, the module provides standard S3 methods for the `gipsmult` object. The `print.gipsmult()` method summarizes the object and its optimization status, while the `plot.gipsmult()` method can be used to generate convergence plots or heatmaps of the projected matrices. These functions have the side effect of printing to the console or creating a plot, respectively. Additionally, the `get_probabilities_from_gipsmult()` function can be used to extract a named vector of posterior probabilities from an optimized object, if they were calculated.

5 Build Model

The models are constructed as extensions of the familiar functions from the `MASS` package. The implementation provides three main functions, `gipsLDA`, `gipsQDA`, and `gipsMultQDA`. These are designed to be intuitive for users of `MASS::lda` and `MASS::qda`, enhancing classical methods by integrating the `gips` methodology for covariance matrix estimation. The standard R user interface, including formula-based model specification and S3 methods like `predict`, is preserved.

5.1 Model Descriptions

The package offers three primary model-building functions from a user's perspective, each representing a distinct modeling assumption.

gipsLDA This function serves as the analog to `MASS::lda`, building a classifier under the assumption of a common covariance matrix for all classes. It includes a specific parameter to select the pooling strategy, allowing the user to choose between the classic, unbiased estimator or a simple weighted average.

gipsQDA This function is the analog to `MASS::qda` and represents the most flexible model. The `gips` algorithm is applied independently to the data of each class. This results in a model where each class has its own distinct covariance matrix and its own unique permutation symmetry.

gipsMultQDA This function implements the intermediate model. It identifies a single, shared permutation symmetry that is most probable across all classes simultaneously. It then estimates a separate covariance matrix for each class, with each matrix being projected onto this common symmetry structure, providing a balance between the flexibility of QDA and the parsimony of a shared structure.

5.2 Parameter Settings

The following key parameters are available in all three model functions to control the `gips`-based estimation process, in addition to standard arguments like `formula`, `data`, and `prior`.

- **MAP**: A logical parameter, defaulting to `TRUE`. If `TRUE`, the covariance matrix is estimated based on the single most probable permutation group (*argmax*). If `FALSE`, the estimator is calculated as a weighted average of covariance matrices over the entire posterior distribution of permutation groups.
- **optimizer**: A character string that specifies the search algorithm for finding the optimal permutation group. It defaults to `NULL`.
  - If `NULL`, the package automatically selects the optimizer. For datasets with 9 or fewer features ($p \leq 9$), it uses a `"BF"` (Brute-force) search. For 10 or more features ($p > 9$), it switches to a `"MH"` (Metropolis-Hastings) stochastic search.
  - `"BF"`: Forces the brute-force search, which is exhaustive and guarantees finding the optimal solution.
  - `"MH"`: Forces the Metropolis-Hastings stochastic search.
- **max_iter**: An integer specifying the number of iterations for the Metropolis-Hastings optimizer, defaulting to `100`. This parameter is active when `optimizer` is set to `"MH"`, or when `optimizer` is `NULL` for datasets with 10 or more features.

5.3 Usage Example

The functions are designed to be invoked just like their `MASS` counterparts. A typical model-building procedure would look like this:

```
# Build a gipsLDA model using the "weighted average" pooling method.
model_lda <- gipsLDA(Species ~ ., data = iris,
                     weighted_average = TRUE)
```

```
# Build a standard gipsQDA model using the weighted average gips estimator.
model_qda <- gipsQDA(Species ~ ., data = iris,
                     MAP = FALSE)

# Build a gipsMultQDA model, forcing the Metropolis-Hastings optimizer.
model_multqda <- gipsMultQDA(Species ~ ., data = iris,
                             optimizer = "MH", max_iter = 200)

# The returned objects are compatible with standard S3 methods
print(model_lda)
predictions <- predict(model_multqda, newdata = iris)
```

This API provides a powerful and flexible interface that is both familiar to R users and capable of configuring the novel aspects of the `gipsDA` methodology.

6 Testing environment

To ensure reproducibility and robustness across different platforms, the development and testing of the `gipsDA` package were conducted on two distinct hardware and software environments. Both systems were configured with R version 4.5.1.

6.1 Platform 1: Apple Silicon (macOS)

The first testing machine was an ARM-based MacBook Pro with the following configuration:

- **Device:** MacBook Pro
- **CPU:** Apple M4 Max (14-core @ 4.51GHz)
- **GPU:** 32-core @ 1.58GHz
- **RAM:** 36 GB
- **Operating System:** macOS Tahoe 26.1
- **Architecture:** ARM
- **Kernel:** Darwin 25.1.0
- **Filesystem:** APFS

6.2 Platform 2: Linux (x86_64)

The second testing machine was a PC running an Arch Linux-based distribution with the following configuration:

- **Operating System:** CachyOS
- **Architecture:** x86_64
- **Kernel:** Linux 6.17.8-2-cachyos
- **CPU:** AMD Ryzen 7 8840U (16-core with integrated GPU @ 5.13GHz)
- **RAM:** 16 GB
- **Filesystem:** BTRFS

This dual-platform approach ensures the package's compatibility and consistent performance on both ARM-based macOS and x86_64-based Linux systems.

References

[1] Adam Chojecki, Paweł Morgen, and Bartosz Kołodziejek. Learning permutation symmetry of a gaussian vector with gips in R. *Journal of Statistical Software*, 112(7):1–38, 2025.