Package ‘eimodel’

February 15, 2025

Title A fast alternative for the R x C ecological inference problem.

Version 0.0.0.1

Author Charles Thraves, Pablo Ubilla, Daniel Hermosilla

Maintainer Daniel Hermosilla <daniel.hermosilla.r@ug.uchile.cl>

Description Calculates the probability matrix of a R x C ecological inference problem with an Expected Maximization algorithm

Encoding UTF-8

Imports Rcpp,
R6,
jsonlite

LinkingTo Rcpp

NeedsCompilation yes

BugReports
https://github.com/DanielHermosilla/ecological-inference-elections/issues

Contents

`eim` *An R6 Class for estimating an probability matrix from an R x C ecological inference problem.*

Description

This class implements an EM algorithm using different methods to approximate the E-step, such as `"Multinomial"`, `"Hit and Run"`, `"MVN CDF"`, `"MVN PDF"`, and `"Exact"`.

Usage

```
## S3 method for class 'eim'
summary(object)

## S3 method for class 'eim'
predict(object, ...)

## S3 method for class 'eim'
as.matrix(object)

std.eim(object, ...)
```

Public fields

X *(matrix)* An (b x c) matrix with the observed results of the candidate votes (c) for each ballot box (b). Provided manually or loaded from JSON.

W *(matrix)* An (b x g) matrix with the observed results of the demographic group votes (g) for each ballot box (b). Provided manually or loaded from JSON.

method *(character)* A string indicating the EM method. One of: "Multinomial", "Hit and Run", "MVN CDF", "MVN PDF", or "Exact".

probability *(matrix)* An (g x c) matrix that will store the final estimated probabilities of a given group (g) voting for a candidate (c).

logLikelihood *(numeric)* A numeric vector that will store the log-likelihood values over all iterations of the Expectation-Maximization algorithm.

total_iterations *(integer(1))* An integer indicating the total number of iterations performed by the Expectation-Maximization algorithm.

total_time *(numeric(1))* The time taken by the EM algorithm.

finish_state *(character)* The reason for the algorithm's termination. It can be either "Maximum iterations", "Log-likelihood decrease", "Convergence", or "Early exit".

Active bindings

samples *(integer(1))* Active variable to show the Hit and Run samples if and only if the method is "Hit and Run".

step_size *(integer(1))* Active variable to show the Hit and Run step size if and only if the method is "Hit and Run".

multivariate_method *(character)* Active variable to show the method used to estimate the Multivariate Normal CDF, if and only if self$method is "MVN CDF".

`multivariate_error` *(numeric(1))* Active variable to show the error threshold for the Monte Carlo simulation of the Multivariate Normal CDF.

`multivariate_iterations` *(numeric(1))* Active variable to show the number of iterations for the Monte Carlo simulation of the Multivariate Normal CDF.

`std` *(matrix)* Active variable to show an estimate of the probability standard deviation if the bootstrapping method has been called.

Methods

Public methods:

- `eim$new()`
- `eim$precompute()`
- `eim$compute()`
- `eim$bootstrap()`
- `eim$print()`
- `eim$summary()`
- `eim$save_results()`
- `eim$clone()`

Method `new()`**:** Creates the object by defining the X and W matrix attributes.

Usage:

```
eim$new(X = NULL, W = NULL, json_path = NULL)
```

Arguments:

`X` *(matrix)* A matrix (c x b) of observed candidate votes per ballot box (b) (optional; required if `json_path` is `NULL`).

`W` *(matrix)* A matrix (b x g) of observed demographic group votes per ballot box (b) (optional; required if `json_path` is `NULL`).

`json_path` *(character)* A string containing a path to a JSON file with "X" and "W" matrices. (optional; required if `X` or `W` are `NULL`)

Returns: An initialized eim object.

Examples:

```
# Example 1: Create a eim object from a matrix
model <- eim$new(X = matrix(1:9, 3, 3), W = matrix(1:9, 3, 3))

# Example 2: Create a eim object from a JSON file
\dontrun{
model2 <- eim$new(json_path = "a/file/path/to/a/file.json")
}
```

Method `precompute()`**:** Calculates the EM-independent variables for the `Hit` and `Run` and `Exact` methods.

Usage:

```
eim$precompute(method, ...)
```

Arguments:

`method` *(character)*	The method for precomputing. Options: `"Hit and Run"` or `"Exact"`.
`...`	Additional arguments required by specific methods:

"Hit and Run" Method:

`step_size` *(integer(1))*	The step size (M) for the Hit and Run algorithm. Must be a positive integer.
`samples` *(integer(1))*	The number of samples (S) to generate. Must be an integer.

Returns: The modified eim object (for method chaining). Updates are made on the C internal memory.

Examples:

```
# Example 1: Precompute the Hit and Run method
simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)

model$precompute("Hit and Run",
     step_size = 1000,
     samples = 5000)
# Changes are made to the C internals API

# Example 2: Precompute the Exact method
model$precompute("Exact")
```

Method `compute()`**:** Executes the Expectation-Maximization (EM) algorithm based on the selected method. Additional parameters may be required depending on the method.

Usage:

```
eim$compute(
  method = "Multinomial",
  probability_method = "Group proportional",
  iterations = 1000,
  maximum_time = 1440,
  stopping_threshold = 0.001,
  verbose = FALSE,
  ...
)
```

Arguments:

`method` *(character)*	The method for estimating the Expectation-Maximization (EM) algorithm. Options: `"Multinomial"`, `"MVN CDF"`, `"MVN PDF"`, `"Hit and Run"`, and `"Exact"` (default: `"Multinomial"`).
`probability_method` *(character)*	The method for obtaining the initial probability. Options: `"Group proportional"`, `"Proportional"`, or `"Uniform"`. (default: `"Group proportional"`).
`iterations` *(integer(1))*	The maximum number of iterations to perform on the EM algorithm. (default: `1000`).
`maximum_time` *(integer(1))*	The maximum time (in minutes) that the algorithm will iterate. (default: `1440`).
`stopping_threshold` *(numeric(1))*	The minimum difference between successive probabilities for stopping the iterations. (default: `0.001`).
`verbose` *(boolean(1))*	A boolean indicating whether to print informative messages while iterating. (default: `FALSE`).
`...`	Additional arguments required by specific methods:

"Hit and Run" Method:

`step_size` *(integer(1))*:	The step size (M) for the Hit and Run algorithm.
`samples` *(integer(1))*:	The number of samples (S) to generate.

"MVN CDF" Method:

`multivariate_method` *(character)*:	The integration method. Can be chosen between `"Genz"` and `"Genz2"`. Default is `"Genz2"`.
`multivariate_error` *(numeric(1))*:	The integration error threshold. Default is `1e-6`.
`multivariate_iterations` *(integer(1))*:	The number of Monte Carlo iterations. Default is `5000`.

Returns: The modified eim object (for method chaining).

Examples:

```
# Example 1: Compute the Expectation-Maximization with default values
```

```
simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)
model$compute() # Retrieves the object with updated attributes

# Example 2: Compute the Expectation-Maximization for the Hit and Run method
\donttest{
 model$compute(method = "Hit and Run",
     step_size = 3000,
     samples = 1000)
 }

# Example 3: Omit arguments to the Hit and Run method
\dontrun{
 model$compute(method = "Hit and Run",
     step_size = 3000)
 # Error; must supply the samples parameter as well
 }

# Example 4: Run the MVN CDF with default values
model$compute(method = "MVN CDF")

# Example 5: Run an Exact estimation with user-defined parameters
model$compute(method = "Exact",
    probability_method = "Uniform",
    iterations = 5,
    stopping_threshold = 1e-3)
# Verbose was omitted
```

Method `bootstrap()`**:** Runs a bootstrap algorithm to estimate the **standard deviation** of probabilities.

Usage:

```
eim$bootstrap(
  bootstrap_iterations,
  ballot_boxes,
  method = "Multinomial",
  probability_method = "Group proportional",
  iterations = 1000,
  stopping_threshold = 0.001,
  verbose = FALSE,
  ...
)
```

Arguments:

`bootstrap_iterations` *(integer(1))* The number of EM computations.

`ballot_boxes` *(integer(1))* The number of ballot boxes to use as a sample. It must be strictly less than the total number of ballot boxes.

Arguments for calling the `$compute()` *method*:

`method` *(character)*	The method for estimating the Expectation-Maximization (EM) algorithm. Options: `"Multinomial"`, `"MVN CDF"`, `"MVN PDF"`, `"Hit and Run"`, and `"Exact"` (default: `"Multinomial"`).
`probability_method` *(character)*	The method for obtaining the initial probability. Options: `"Group proportional"`, `"Proportional"`, or `"Uniform"`. (default: `"Group proportional"`).
`iterations` *(integer(1))*	The maximum number of iterations to perform on the EM algorithm. (default: `1000`).
`maximum_time` *(integer(1))*	The maximum time (in minutes) that the algorithm will iterate. (default: `1440`).
`stopping_threshold` *(numeric(1))*	The minimum difference between successive probabilities for stopping the iterations. (default: `0.001`).
`verbose` *(boolean(1))*	A boolean indicating whether to print informative messages while iterating. (default: `FALSE`).
`...`	Additional arguments required by specific methods:

"Hit and Run" Method:

`step_size` *(integer(1))*:	The step size (M) for the Hit and Run algorithm.
`samples` *(integer(1))*:	The number of samples (S) to generate.

"MVN CDF" Method:

`multivariate_method` *(character)*:	The integration method. Can be chosen between `"Genz"` and `"Genz2"`. Default is `"Genz2"`.
`multivariate_error` *(numeric(1))*:	The integration error threshold. Default is `1e-6`.

`multivariate_iterations` *(integer(1))*: The number of Monte Carlo iterations. Default is `5000`.

Returns: The modified eim object, under the field `$std`.

Examples:

```
simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)

model$bootstrap(30, 10)

model$std # An estimate of the standard deviation of the probabilities.
```

Method `print()`**:** Depending on the state of the algorithm (computed or not), it prints a message with its most relevant parameters.

Usage:

```
eim$print()
```

Returns: The object itself (for method chaining).

Examples:

```
simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)
print(model) # Prints the X and W matrices.

model$compute()
print(model) # Prints the X and W matrices along with the EM results.
```

Method `summary()`**:** Shows, in the form of a list, a selection of the most important attributes. It retrieves the method, number of candidates, ballots, and groups, as well as the principal results of the EM algorithm.

Usage:

```
eim$summary()
```

Returns: *(list)* A list with the method, candidates, ballots, groups, probabilities, and log-likelihood.

Examples:

```
simulations <- simulate_elections(num_ballots = 5,
    num_candidates = 3,
    num_groups = 2,
    ballot_voters = rep(100, 5))

model <- eim$new(simulations$X, simulations$W)
a_list <- model$summary()
a_list$method # Not computed yet
```

```
a_list$groups # 2
a_list$ballots # 5
names(a_list)
# "candidates" "groups" "ballots" "method" "probabilities" "logLikelihood"
```

Method `save_results()`**:** Saves the current eim object to a specified file. The results can be saved in:

- **RDS (Binary format):** Preserves object structure for future use in R.
- **JSON:** Saves model data in a human-readable format.
- **CSV:** Saves the probability matrix in a tabular format.

Usage:

```
eim$save_results(filename)
```

Arguments:

`filename` *(character)*	The file name where the results should be saved, including its extension. The file extension determines the format:
**.rds*	Saves as '`RDS`' (default, binary format).
**.json*	Saves as '`JSON`' (readable and shareable).
**.csv*	Saves the *final probability matrix* as '`CSV`'.

Returns: The modified eim object (for method chaining).

Examples:

```
simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)
model$compute()
model$save_results("results.rds")  # Save as RDS
model$save_results("results.json") # Save as JSON
model$save_results("results.csv")  # Save the final probability matrix as CSV
```

Note

Precomputing can eventually accelerate the computation time of the Expectation-Maximization algorithm.

The `$compute`, `$bootstrap`, `$print` and `$summary` methods can also be called as an S3 method with `predict()`, `std()`, `print()` and `summary()`.

Author(s)

Charles Thraves, Pablo Ubilla

References

Thraves, C. and Ubilla, P.: 'Fast Ecological Inference Algorithm for the R×C Case'

Examples

```
## ------------------------------------------------
## Method `eim$new`
## ------------------------------------------------

# Example 1: Create a eim object from a matrix
model <- eim$new(X = matrix(1:9, 3, 3), W = matrix(1:9, 3, 3))

# Example 2: Create a eim object from a JSON file
## Not run:
model2 <- eim$new(json_path = "a/file/path/to/a/file.json")

## End(Not run)

## ------------------------------------------------
## Method `eim$precompute`
## ------------------------------------------------

# Example 1: Precompute the Hit and Run method
simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)

model$precompute("Hit and Run",
     step_size = 1000,
     samples = 5000)
# Changes are made to the C internals API

# Example 2: Precompute the Exact method
model$precompute("Exact")

## ------------------------------------------------
## Method `eim$compute`
## ------------------------------------------------

# Example 1: Compute the Expectation-Maximization with default values

simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)
model$compute() # Retrieves the object with updated attributes

# Example 2: Compute the Expectation-Maximization for the Hit and Run method

 model$compute(method = "Hit and Run",
     step_size = 3000,
```

```
     samples = 1000)


# Example 3: Omit arguments to the Hit and Run method
## Not run:
 model$compute(method = "Hit and Run",
     step_size = 3000)
 # Error; must supply the samples parameter as well

## End(Not run)

# Example 4: Run the MVN CDF with default values
model$compute(method = "MVN CDF")

# Example 5: Run an Exact estimation with user-defined parameters
model$compute(method = "Exact",
    probability_method = "Uniform",
    iterations = 5,
    stopping_threshold = 1e-3)
# Verbose was omitted

## ------------------------------------------------
## Method `eim$bootstrap`
## ------------------------------------------------

simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(X = simulations$X, W = simulations$W)

model$bootstrap(30, 10)

model$std # An estimate of the probabilities' standard deviation.

## ------------------------------------------------
## Method `eim$print`
## ------------------------------------------------

simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)
print(model) # Prints the X and W matrices.

model$compute()
print(model) # Prints the X and W matrices along with the EM results.

## ------------------------------------------------
## Method `eim$summary`
## ------------------------------------------------

simulations <- simulate_elections(num_ballots = 5,
    num_candidates = 3,
```

```
    num_groups = 2,
    ballot_voters = rep(100, 5))

model <- eim$new(simulations$X, simulations$W)
a_list <- model$summary()
a_list$method # Not computed yet
a_list$groups # 2
a_list$ballots # 5
names(a_list)
# "candidates" "groups" "ballots" "method" "probabilities" "logLikelihood"

## ---------------------------------------------
## Method `eim$save_results`
## ---------------------------------------------

simulations <- simulate_elections(num_ballots = 20,
    num_candidates = 5,
    num_groups = 3,
    ballot_voters = rep(100, 20))

model <- eim$new(simulations$X, simulations$W)
model$compute()
model$save_results("results.rds")  # Save as RDS
model$save_results("results.json") # Save as JSON
model$save_results("results.csv")  # Save the final probability matrix as CSV
```

`random_samples` *Randomly create a voting instance by defining an interval*

Description

Given a range of possible **observed** outcomes (such as the number of ballot boxes, candidates, etc.), it creates a completely random voting instance, simulating the unobserved results as well.

Usage

```
random_samples(
  ballots_range,
  candidates_range,
  group_range,
  voting_range,
  seed = NULL
)
```

Arguments

`ballots_range` (integer) A vector of length 2 specifying the lower and upper bounds for the number of ballot boxes.

`candidates_range` (integer) A vector of length 2 specifying the lower and upper bounds for the number of candidates to draw.

`group_range` (integer) A vector of length 2 specifying the lower and upper bounds for the number of demographic groups to draw.

voting_range (integer) A vector of length 2 specifying the lower and upper bounds for the number of votes per ballot box.

seed *(numeric(1))* Optional. If provided, it overrides the current global seed.

Value

A list with components:

X A matrix (b x c) containing candidate votes per ballot box.

W A matrix (b x g) containing demographic votes per ballot box.

real_p A matrix (g x c) containing the estimated **(unobserved)** probabilities that a demographic group votes for a given candidate.

ballots The number of ballot boxes that were drawn.

candidates The number of candidates that were drawn.

groups The number of demographic groups that were drawn.

total_votes A vector containing the total number of votes per ballot box.

See Also

[simulate_elections()]

Examples

```
bal_range <- c(30, 50)
can_range <- c(2, 4)
group_range <- c(2, 6)
voting_range <- c(50, 100)
results <- random_samples(bal_range, can_range, group_range, voting_range)

# X matrix
results$X # A randomly generated matrix of dimension (b x c)
ncol(results$X <= can_range[2]) # Always TRUE
ncol(results$X >= can_range[1]) # Always TRUE
nrow(results$X <= bal_range[2]) # Always TRUE
nrow(results$X >= bal_range[1]) # Always TRUE

# W matrix
results$W # A randomly generated matrix of dimension (b x g)
ncol(results$W <= group_range[2]) # Always TRUE
ncol(results$W >= group_range[1]) # Always TRUE
nrow(results$W <= bal_range[2]) # Always TRUE
nrow(results$W >= bal_range[1]) # Always TRUE

# Probability matrix
results$real_p # A matrix (g x c) that summarizes the unobserved outcomes
ncol(results$real_p) == ncol(results$X) # Always TRUE
nrow(results$real_p) == ncol(results$W) # Always TRUE
```

`simulate_elections` *Simulates an election by creating the candidate and group matrices and their results.*

Description

Given the number of ballots, groups, candidates, and votes per ballot, this function simulates an election. Additionally, it generates a resulting matrix (of dimension g x c) representing the unobserved probabilities that a demographic group votes for a given candidate. These probabilities are drawn from a Dirichlet distribution and adjusted by a `lambda` value, which represents the heterogeneity of the groups.

Usage

```
simulate_elections(
  num_ballots,
  num_candidates,
  num_groups,
  ballot_voters,
  lambda = 0.5,
  seed = NULL
)
```

Arguments

`num_ballots`	*(integer(1))* The number of ballot boxes *("b")*.
`num_candidates`	*(integer(1))* The number of candidates *("c")*.
`num_groups`	*(integer(1))* The number of demographic groups *("g")*.
`ballot_voters`	*(integer(num_ballots))* A vector of length `num_ballots` with the number of votes per ballot box.
`lambda`	*(numeric(1))* A value between 0 and 1 representing the heterogeneity of the groups. Values near 0 yield more heterogeneous results, but may be less realistic; the opposite is true for values closer to 1. (default: 0.5)
`seed`	*(numeric(1))* Optional. If provided, it overrides the current global seed. (default: `NULL`)

Value

A list with components:

`X`	A matrix (b x c) with candidate votes per ballot box.
`W`	A matrix (b x g) with demographic votes per ballot box.
`real_p`	A matrix (g x c) with the estimated (unobserved) probabilities that a demographic group votes for a given candidate.

Note

Note: For the Dirichlet distribution, an alpha value of 1 yields a distribution that is uniform around the mean. Conversely, alpha values less than 1 tend to produce a sparser probability vector, with many values near zero and one or a few larger values. When alpha is greater than 1, the distribution becomes more concentrated around the mean.

See Also

[random_samples()]

Examples

```
result <- simulate_elections(num_ballots = 10,
        num_candidates = 5,
        num_groups = 3,
        ballot_voters = c(100, 10))
result$X # Candidate matrix (b x c)
result$W # Group matrix (b x g)
result$real_p # Probability matrix (g x c)
```

---

sum.eim *Returns the total number of voters in the system.*

Description

Given an initialized object, it returns the total number of voters. It is equivalent to running `sum(object$W)` or `sum(object$X)`.

Usage

```
## S3 method for class 'eim'
sum(object)
```

Arguments

`object`	An eim object.

Value

The total number of voters.

Examples

```
simulations <- simulate_elections(
  num_ballots = 20,
  num_candidates = 5,
  num_groups = 3,
  ballot_voters = rep(100, 20)
)

model <- eim$new(simulations$X, simulations$W)
sum(model) # 2000
```

update.eim — *Update an existing eim model with a new EM algorithm computation*

Description

This function updates an object by performing a new Expectation-Maximization computation with different parameters.

Usage

```
## S3 method for class 'eim'
update(object, ...)
```

Arguments

`...`	New parameters to be passed to `compute()`.
`object`	An eim object.

Value

The updated object with the changes applied.

Examples

```
simulations <- simulate_elections(
  num_ballots = 20,
  num_candidates = 5,
  num_groups = 3,
  ballot_voters = rep(100, 20)
)

model <- eim$new(simulations$X, simulations$W)

predict(model)

update(model, "MVN PDF")

model$method # Returns 'MVN PDF'
```

Index