Package: bHIVE (via r-universe)

June 9, 2026

Title B-cell Hybrid Immune Variant Engine

Version 0.99.4

Description The bHIVE package implements an Artificial Immune Network (AI-Net) algorithm for clustering and classification tasks. Inspired by biological immune systems, it employs clonal selection, mutation, and network suppression to analyze and model datasets. This package provides flexible functionality, including affinity metrics, mutation strategies, and hyperparameter tuning.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

biocViews Software, Clustering, Classification, Network

Depends R (>= 4.5.0)

Imports BiocParallel, cluster, clusterCrit, ggplot2, R6, Rcpp, Rtsne, stats, umap, viridis

LinkingTo Rcpp, RcppArmadillo

Suggests BiocStyle, caret, devtools, knitr, pkgdown, rmarkdown, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

URL https://www.borch.dev/uploads/bhive/

BugReports https://github.com/BorchLab/bHIVE/issues

Config/pak/sysreqs libpng-dev libssl-dev python3

Repository https://biocstaging.r-universe.dev

Date/Publication 2026-06-09 18:26:47 UTC

RemoteUrl https://github.com/BiocStaging/bHIVE

RemoteRef HEAD

RemoteSha 183b0ab49202e02ebb7a1ed7bb98ca2dee6c10ab

Contents

`ActivationGate` *ActivationGate*

Description

Two-signal activation gate implementing the immunological principle that immune cell activation requires both antigen-specific recognition (Signal 1) AND costimulatory context (Signal 2).

Details

Prevents spurious activation on isolated outliers. An antibody is only allowed to clone if both signals exceed their thresholds. This is biologically-principled regularization.

Signal 2 options:

- `"density"`: Local data density around the antibody
- `"danger"`: User-provided danger signal vector
- `"entropy"`: Local label entropy (classification only)

Public fields

`signal2_type` Type of costimulatory signal.

`threshold1` Minimum affinity for Signal 1 (antigen recognition).

`threshold2` Minimum costimulatory signal for Signal 2.

`danger_signals` User-provided danger signal vector (for "danger" type).

Methods

Public methods:

- ActivationGate$new()
- ActivationGate$evaluate()
- ActivationGate$print()
- ActivationGate$clone()

Method `new()`**:** Create a new ActivationGate.

Usage:

```
ActivationGate$new(
  signal2_type = "density",
  threshold1 = 0.1,
  threshold2 = 0.3,
  danger_signals = NULL
)
```

Arguments:

`signal2_type` Character. "density", "danger", or "entropy".

`threshold1` Numeric. Minimum affinity threshold.

`threshold2` Numeric. Minimum Signal 2 threshold.

`danger_signals` Numeric vector. Per-data-point danger scores.

Method `evaluate()`**:** Evaluate which antibody-data interactions pass the two-signal gate.

Usage:

```
ActivationGate$evaluate(affinity_matrix, X, A, y = NULL, task = "clustering")
```

Arguments:

`affinity_matrix` Numeric matrix (n x m) of affinities.

`X` Numeric matrix of data (n x d).

`A` Numeric matrix of antibodies (m x d).

`y` Target vector or NULL.

`task` Character. Task type.

Returns: Logical matrix (n x m) where TRUE means the interaction is activated.

Method `print()`**:** Print summary.

Usage:

```
ActivationGate$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ActivationGate$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Two-signal activation gate
data(iris)
X <- as.matrix(iris[, 1:4])
A <- X[sample(150, 10), ]
rep <- ImmuneRepertoire$new(A)
gate <- ActivationGate$new(signal2_type = "density", threshold2 = 0.3)
aff <- rep$affinity_matrix(X, "gaussian")
activated <- gate$evaluate(aff, X, A)
sum(activated)  # number of activated interactions
```

AINet *AINet*

Description

R6 implementation of the Artificial Immune Network algorithm. This is the core bHIVE algorithm using C++ backends for performance-critical operations. Supports composable modules for somatic hypermutation, idiotypic network regulation, germinal center selection, and more.

Super class

bHIVE::ImmuneAlgorithm -> AINet

Methods

Public methods:

- AINet$new()
- AINet$fit()
- AINet$clone()

Method `new()`**:** Create a new AINet algorithm instance.

Usage:

```
AINet$new(
  nAntibodies = 20,
  beta = 5,
  epsilon = 0.01,
  maxIter = 50,
  k = 3,
  affinityFunc = "gaussian",
  distFunc = "euclidean",
  affinityParams = list(alpha = 1, c = 1, p = 2, Sigma = NULL),
  mutationDecay = 1,
  mutationMin = 0.01,
  maxClones = Inf,
  stopTolerance = 0,
```

```
  noImprovementLimit = Inf,
  initMethod = "sample",
  shm = NULL,
  init = NULL,
  activation = NULL,
  idiotypic = NULL,
  germinalCenter = NULL,
  microenvironment = NULL,
  memory = NULL,
  classSwitcher = NULL,
  verbose = TRUE
)
```

Arguments:

`nAntibodies` Integer. Initial antibody population size.

`beta` Numeric. Clone multiplier.

`epsilon` Numeric. Suppression distance threshold.

`maxIter` Integer. Maximum iterations.

`k` Integer. Top-k antibodies to clone per data point.

`affinityFunc` Character. Affinity function name.

`distFunc` Character. Distance function name.

`affinityParams` List. Parameters for affinity/distance functions.

`mutationDecay` Numeric. Per-iteration mutation rate decay.

`mutationMin` Numeric. Minimum mutation rate.

`maxClones` Numeric. Maximum clones per antibody.

`stopTolerance` Numeric. Early stopping tolerance.

`noImprovementLimit` Integer. Early stopping patience.

`initMethod` Character. Initialization method.

`shm` An SHMEngine instance or NULL for default uniform mutation.

`init` A VDJLibrary instance or NULL for default initialization.

`activation` An ActivationGate instance or NULL.

`idiotypic` An IdiotypicNetwork instance or NULL.

`germinalCenter` A GerminalCenter instance or NULL.

`microenvironment` A Microenvironment instance or NULL.

`memory` A MemoryPool instance or NULL.

`classSwitcher` A ClassSwitcher instance or NULL.

`verbose` Logical. Print progress.

Method `fit()`**:** Fit the AINet algorithm to data.

Usage:

```
AINet$fit(X, y = NULL, task = NULL, ...)
```

Arguments:

`X` Numeric matrix or data frame (n x d).

`y` Optional factor target for classification.

`task` Character: "clustering" or "classification". Inferred from y if NULL.

`...` Additional arguments (currently unused).

Returns: Invisible self, with `result` populated.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AINet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Clustering with Iris data
data(iris)
X <- as.matrix(iris[, 1:4])
model <- AINet$new(nAntibodies = 15, maxIter = 10, verbose = FALSE)
model$fit(X, task = "clustering")
table(model$result$assignments)

# Classification
model2 <- AINet$new(nAntibodies = 20, maxIter = 10, verbose = FALSE)
model2$fit(X, iris$Species, task = "classification")
mean(model2$result$assignments == as.character(iris$Species))

# Predict on new data
preds <- model2$predict(X[1:10, ])
```

`bHIVE` *bHIVE: B-cell Hybrid Immune Variant Engine*

Description

Implements an artificial immune network algorithm for clustering and classification tasks. The algorithm evolves a population of "antibodies" via clonal selection and mutation, applies network suppression to maintain diversity, and assigns data points based on affinity or distance metrics.

Usage

```
bHIVE(
  X,
  y = NULL,
  task = NULL,
  nAntibodies = 20,
  beta = 5,
  epsilon = 0.01,
  maxIter = 50,
```

```
  affinityFunc = "gaussian",
  distFunc = "euclidean",
  affinityParams = list(alpha = 1, c = 1, p = 2, Sigma = NULL),
  mutationDecay = 1,
  mutationMin = 0.01,
  maxClones = Inf,
  stopTolerance = 0,
  noImprovementLimit = Inf,
  initMethod = c("sample", "random", "random_uniform", "kmeans++"),
  k = 3,
  verbose = TRUE
)
```

Arguments

X	A numeric matrix or data frame of input features, with rows as observations and columns as features.
y	Optional. A factor target vector for classification. If NULL, clustering will be performed.
task	Character. Specifies the task to perform: "clustering" or "classification". If NULL, it is inferred based on y.
nAntibodies	Integer. The initial population size of antibodies.
beta	Numeric. Clone multiplier (controls how many clones are generated for top-matching antibodies).
epsilon	Numeric. Similarity threshold used in network suppression; antibodies closer than epsilon are considered redundant.
maxIter	Integer. Maximum number of iterations to run the AI-Net algorithm.
affinityFunc	Character. Specifies the affinity (similarity) function to use for antibody-data matching. One of "gaussian", "laplace", "polynomial", "cosine", or "hamming".
distFunc	Character. Specifies the distance function for clustering and suppression. One of "euclidean", "manhattan", "minkowski", "cosine", "mahalanobis", or "hamming".
affinityParams	A list of optional parameters for the chosen affinity or distance function. • alpha (for RBF or Laplace kernel), • c, p (for polynomial kernel or Minkowski distance), • Sigma (for Mahalanobis distance).
mutationDecay	Numeric. Factor by which the mutation rate decays each iteration (should be ≤ 1.0). Default is 1.0 (no decay).
mutationMin	Numeric. Minimum mutation rate, preventing the mutation scale from shrinking to zero.
maxClones	Numeric. Maximum number of clones per top-matching antibody; defaults to Inf.
stopTolerance	Numeric. If the change in the number of antibodies (repertoire size) is $\leq stopTolerance$ for consecutive iterations, this may trigger the noImprovementLimit.

noImprovementLimit
: Integer. Stops the algorithm early if there is no further improvement in antibody count (beyond stopTolerance) for this many consecutive iterations. Default is Inf, meaning no early stop based on improvement.

initMethod
: Character. Method for initializing antibodies. Can be:
  - "sample" - randomly selects rows from X as initial antibodies.
  - "random" - samples Gaussian noise using X's column means/sds.
  - "random_uniform" - samples uniformly in [min, max] of each column.
  - "kmeans++" - tries a kmeans++-like initialization for coverage.

k
: Integer. Number of top-matching antibodies (by affinity) to consider cloning for each data point.

verbose
: Logical. If TRUE, prints progress messages each iteration.

Value

A list:

- antibodies: Final antibody vectors (nAntibodies x nFeatures).
- assignments: - For clustering: integer cluster IDs in [1..#Antibodies]. - For classification: predicted labels.
- task: The chosen task.

Examples

```
# Example 1: Clustering with the Iris dataset
data(iris)
X <- as.matrix(iris[, 1:4])  # Numeric features only
res <- bHIVE(X = X,
             task = "clustering",
             nAntibodies = 30,
             beta = 5,
             epsilon = 0.01,
             maxIter = 20,
             k = 3,
             verbose = FALSE)
table(res$assignments)

# Example 2: Classification with Iris species
y <- iris$Species
res <- bHIVE(X = X,
              y = y,
              task = "classification",
              nAntibodies = 30,
              beta = 5,
              epsilon = 0.01,
              maxIter = 20,
              k = 3,
              verbose = FALSE)
table(res$assignments, y)
```

bHIVEmodel *B-cell-based Hybrid Immune Virtual Evolution (bHIVE) for caret*

Description

A wrapper for integrating the B-cell Hybrid Immune Variant Engine (bHIVE) algorithm with the `caret` package. Supports classification tasks, providing compatibility with `caret::train()` for model training and validation.

Usage

```
bHIVEmodel
```

Format

A list containing the components required for integration with the `caret` package.

Details

The `bHIVEmodel` wrapper facilitates the use of bHIVE for classification. It defines the model label, parameter grid, fitting function, and prediction methods to conform to the `caret` model specification.

Components

`label` Character string. Identifies the model as "B-cell-based Hybrid Immune Virtual Evolution".

`library` Character string. Specifies the R package containing the bHIVE implementation. Default is "customPackage".

`type` Character vector. Specifies the supported tasks: "Classification".

`parameters` A `data.frame` describing the tunable parameters:

- `parameter`: Name of the parameter.
- `class`: Data type of the parameter ("numeric").
- `label`: Short description of the parameter.

`grid` Function. Generates a grid of tuning parameters for hyperparameter optimization.

`fit` Function. Trains the bHIVE model using specified hyperparameters and task type.

`predict` Function. Generates predictions for new data (classification labels).

`prob` Function. Calculates class probabilities for classification tasks.

Parameters

`nAntibodies` Number of initial antibodies in the bHIVE algorithm.

`beta` Clone multiplier. Controls the number of clones generated for top-matching antibodies.

`epsilon` Similarity threshold for antibody suppression. Smaller values encourage more diversity in the repertoire.

Functions

- `grid(x, y, len)`: Generates a grid of tuning parameters. Accepts:
  - `x`: Feature matrix or data frame.
  - `y`: Factor target vector for classification.
  - `len`: Number of grid points for each parameter.
- `fit(x, y, wts, param, lev, last, classProbs, ...)`: Trains the bHIVE model. Key arguments:
  - `x`: Feature matrix or data frame.
  - `y`: Target vector.
  - `param`: List of hyperparameters (`nAntibodies`, `beta`, `epsilon`).
  - `...`: Additional arguments passed to the bHIVE function.
- `predict(modelFit, newdata, submodels)`: Generates predictions for new data.
  - `modelFit`: Trained bHIVE model.
  - `newdata`: New feature data for prediction.
- `prob(modelFit, newdata, submodels)`: Calculates class probabilities (classification only).

Example Usage

```
library(caret)

# Simulated classification dataset
set.seed(123)
X <- matrix(rnorm(100 * 5), ncol = 5)
y <- factor(sample(c("Class1", "Class2"), 100, replace = TRUE))

# Train bHIVE model using caret
trainControl <- trainControl(method = "cv", number = 5, classProbs = TRUE)
tunedModel <- train(
  x = X,
  y = y,
  method = bHIVEmodel,
  trControl = trainControl,
  tuneLength = 3
)

# Predictions
predictions <- predict(tunedModel, newdata = X)
probabilities <- predict(tunedModel, newdata = X, type = "prob")
```

See Also

`train`, `trainControl`

Examples

```
# View model structure
bHIVEmodel$label
bHIVEmodel$parameters
```

`ClassSwitcher` *ClassSwitcher*

Description

Implements antibody isotype/class switching, allowing antibodies to change their matching breadth. Inspired by real B cell class switching from IgM (broad, pentameric) to IgG (specific, monomeric) to IgA (mucosal).

Details

In bHIVE, class switching modifies the effective affinity kernel width:

- `IgM`: Broad matching (large kernel width) – good for initial exploration and capturing general patterns
- `IgG`: Specific matching (small kernel width) – good for fine-grained discrimination after patterns are identified
- `IgA`: Boundary patrol (medium kernel width) – good for maintaining coverage at decision boundaries

Public fields

`alpha_IgM` Kernel width for IgM mode (broad).

`alpha_IgG` Kernel width for IgG mode (specific).

`alpha_IgA` Kernel width for IgA mode (boundary).

Methods

Public methods:

- `ClassSwitcher$new()`
- `ClassSwitcher$switch_isotypes()`
- `ClassSwitcher$get_alpha()`
- `ClassSwitcher$print()`
- `ClassSwitcher$clone()`

Method `new()`**:** Create a new ClassSwitcher.

Usage:

```
ClassSwitcher$new(alpha_IgM = 0.1, alpha_IgG = 5, alpha_IgA = 1)
```

Arguments:

`alpha_IgM` Numeric. Kernel width for broad matching.

`alpha_IgG` Numeric. Kernel width for specific matching.

`alpha_IgA` Numeric. Kernel width for boundary matching.

Method `switch_isotypes()`**:** Determine appropriate isotype for each antibody based on its microenvironment zone.

Usage:

```
ClassSwitcher$switch_isotypes(repertoire, zones)
```

Arguments:

`repertoire` An `ImmuneRepertoire`.

`zones` Character vector from Microenvironment assessment.

Returns: Named numeric vector of alpha values per antibody.

Method `get_alpha()`**:** Get alpha value for a given isotype.

Usage:

```
ClassSwitcher$get_alpha(isotype)
```

Arguments:

`isotype` Character. "IgM", "IgG", or "IgA".

Returns: Numeric.

Method `print()`**:** Print summary.

Usage:

```
ClassSwitcher$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ClassSwitcher$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Switch antibody isotypes based on microenvironment zones
A <- matrix(rnorm(50), nrow = 10, ncol = 5)
rep <- ImmuneRepertoire$new(A)
cs <- ClassSwitcher$new(alpha_IgM = 0.1, alpha_IgG = 5.0)
zones <- sample(c("stable", "explore", "boundary"), 10, replace = TRUE)
alphas <- cs$switch_isotypes(rep, zones)
table(rep$metadata$isotype)  # IgM, IgG, IgA distribution
```

ConvergentSelector *ConvergentSelector*

Description

Identifies "public antibodies" shared across independent repertoires, implementing the concept of convergent selection from TCR/BCR immunology as a biologically-motivated ensemble method.

Details

In real immunity, certain immune receptor sequences appear across multiple individuals (public clones), suggesting they are driven by common selection pressures. Similarly, antibodies that appear across multiple independent bHIVE runs represent the most robust patterns in the data.

Public fields

`tolerance` Distance tolerance for matching antibodies across repertoires.

`min_appearances` Minimum number of repertoires an antibody must appear in to be considered "public".

`public_antibodies` The identified public antibodies.

Methods

Public methods:

- `ConvergentSelector$new()`
- `ConvergentSelector$find_public()`
- `ConvergentSelector$from_results()`
- `ConvergentSelector$print()`
- `ConvergentSelector$clone()`

Method `new()`**:** Create a new ConvergentSelector.

Usage:

```
ConvergentSelector$new(tolerance = 0.5, min_appearances = 2)
```

Arguments:

`tolerance` Numeric. Maximum distance for two antibodies to be considered the same across repertoires.

`min_appearances` Integer. Minimum repertoires for an antibody to be public.

Method `find_public()`**:** Find public antibodies shared across multiple repertoires.

Usage:

```
ConvergentSelector$find_public(repertoires, distFunc = "euclidean")
```

Arguments:

`repertoires` List of ImmuneRepertoire objects or list of antibody matrices.

`distFunc` Character. Distance function for matching.

Returns: Numeric matrix of public (consensus) antibodies.

Method `from_results()`**:** Run convergent selection from multiple bHIVE results.

Usage:

```
ConvergentSelector$from_results(results, distFunc = "euclidean")
```

Arguments:

`results` List of bHIVE result objects (each with $antibodies).

`distFunc` Character. Distance function.

Returns: Numeric matrix of consensus antibodies.

Method `print()`**:** Print summary.

Usage:

```
ConvergentSelector$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ConvergentSelector$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Find public antibodies across multiple runs
data(iris)
X <- as.matrix(iris[, 1:4])
results <- lapply(1:3, function(i) {
  m <- AINet$new(nAntibodies = 15, maxIter = 5, verbose = FALSE)
  m$fit(X, task = "clustering")
  m$result
})
conv <- ConvergentSelector$new(tolerance = 1.0, min_appearances = 2)
public <- conv$from_results(results)
nrow(public)  # consensus antibodies
```

GerminalCenter *GerminalCenter*

Description

Models T follicular helper (Tfh) cell selection pressure on B cells within a germinal center reaction. Implements resource competition where antibodies compete for Tfh help, and only helped antibodies survive.

Details

The germinal center is where B cells undergo affinity maturation through iterative cycles of mutation and selection. Tfh cells act as quality-control selectors:

- Each Tfh evaluates B cell (antibody) quality using a task-aware metric
- B cells compete for Tfh help (resource competition)
- Only helped B cells survive to the next round
- Selection pressure controls the stringency of the process

Public fields

`nTfh` Number of Tfh selectors (determines how many antibodies survive).

`selectionPressure` Numeric [0,1]. 0 = no selection (all survive), 1 = only the very best survive.

`rounds` Number of selection rounds per call.

`last_survivors` Integer vector of survivor indices (relative to the input repertoire) from the most recent call to `select()`.

Methods

Public methods:

- `GerminalCenter$new()`
- `GerminalCenter$select()`
- `GerminalCenter$print()`
- `GerminalCenter$clone()`

Method `new()`**:** Create a new GerminalCenter.

Usage:

```
GerminalCenter$new(nTfh = 10, selectionPressure = 0.5, rounds = 1)
```

Arguments:

`nTfh` Integer. Number of Tfh helper cells. Each helps one B cell.

`selectionPressure` Numeric [0,1]. Stringency of selection.

`rounds` Integer. Number of competition rounds.

Method `select()`**:** Run germinal center selection on a repertoire.

Usage:

```
GerminalCenter$select(
  repertoire,
  X,
  y = NULL,
  task = "clustering",
  affinityFunc = "gaussian",
  affinityParams = list(alpha = 1, c = 1, p = 2)
)
```

Arguments:

`repertoire` An `ImmuneRepertoire` object.

`X` Numeric matrix of training data.

`y` Factor target vector or NULL for clustering.

`task` Character: "clustering" or "classification".

`affinityFunc` Character. Affinity function for evaluation.

`affinityParams` List. Parameters for affinity function.

Returns: Integer vector of survivor indices relative to the input repertoire (composed across all selection rounds). Also stored on `self$last_survivors` for inspection. Repertoire is modified in place.

Method `print()`**:** Print summary.

Usage:

```
GerminalCenter$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GerminalCenter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Germinal center selection on Iris
data(iris)
X <- as.matrix(iris[, 1:4])
gc <- GerminalCenter$new(nTfh = 5, selectionPressure = 0.5)
rep <- ImmuneRepertoire$new(X[sample(150, 20), ])
gc$select(rep, X, iris$Species, "classification")
rep$size()  # fewer antibodies after selection
```

honeycombHIVE *honeycombHIVE: Multilayer AIS with optional gradient-based fine-tuning*

Description

The honeycombHIVE function implements a multilayer artificial immune system that iteratively refines a set of prototypes - referred to as antibodies - to model the structure of the input data. In each layer, the function first uses the bHIVE algorithm to generate or update antibodies based on the current data representation and task (clustering or classification). Optionally, it applies gradient-based fine-tuning (via refineB) to these antibodies, allowing for advanced refinement through various optimizers (e.g., SGD, Adam, RMSProp) and customizable loss functions. The final output is a hierarchical set of layers that encapsulate both the refined prototypes and the corresponding cluster assignments or predictions for the original observations, making honeycombHIVE a versatile tool for adaptive learning and pattern recognition.

Usage

```
honeycombHIVE(
  X,
  y = NULL,
  task = c("clustering", "classification"),
  layers = 3,
  nAntibodies = 20,
  minAntibodies = 5,
  epsilon = 0.05,
  beta = 5,
  maxIter = 10,
  collapseMethod = c("centroid", "medoid", "median", "mode"),
  minClusterSize = NULL,
  distance = "euclidean",
  verbose = TRUE,
  refine = FALSE,
  refineLoss = "categorical_crossentropy",
  refineSteps = 5,
  refineLR = 0.01,
  refinePushAway = TRUE,
  refineOptimizer = "sgd",
  refineMomentumCoef = 0.9,
  refineBeta1 = 0.9,
  refineBeta2 = 0.999,
  refineRmspropDecay = 0.9,
  refineEpsilon = 1e-08,
  ...
)
```

Arguments

X	A numeric matrix or data frame of input features (rows = observations, columns = features).
y	Optional target factor vector for classification.
task	Character, one of "clustering" or "classification".
layers	Integer, how many layers (AIS iterations) to run.
nAntibodies	Integer, how many antibodies (prototypes) to generate initially in each layer.
minAntibodies	Integer, minimal number of antibodies to keep in each layer.
epsilon	Numeric, threshold param for bHIVE suppression.
beta	Numeric, selection pressure param for bHIVE.
maxIter	Integer, maximum iterations for bHIVE each layer.
collapseMethod	One of "centroid","medoid","median","mode".
minClusterSize	Minimum cluster size. Smaller clusters can be merged/discarded if not NULL.
distance	Distance metric for medoid calculation, e.g. "euclidean".
verbose	Logical, if TRUE prints progress at each layer.
refine	Logical, if TRUE apply gradient-based refinement via refineB() to each layer's prototypes.
refineLoss	Character specifying the loss for refineB() (e.g. "categorical_crossentropy", "mae").
refineSteps	Integer, number of gradient steps in refineB().
refineLR	Numeric, learning rate for gradient updates.
refinePushAway	Logical, if TRUE and classification, push prototypes away from differently labeled points.
refineOptimizer	Character, one of "sgd", "momentum", "adagrad","adam", "rmsprop" to be passed to refineB().
refineMomentumCoef	Numeric, momentum coefficient (if using momentum).
refineBeta1	Numeric, first moment decay rate (if using Adam).
refineBeta2	Numeric, second moment decay rate (if using Adam).
refineRmspropDecay	Numeric, decay rate for the moving average of squared gradients (if using RMSProp).
refineEpsilon	Numeric, a small constant for numerical stability (used in adaptive optimizers).
...	Additional arguments passed to bHIVE.

Value

A list of length `layers`. Each element (layer) includes:

- `antibodies`: The prototypes in that layer.
- `assignments`: Antibody index (in that layer) for each row of `current_X`.
- `membership`: For each **original** row in `X`, which cluster/antibody it belongs to in this layer.
- `predictions`: If classification, predicted label for each original row in `X`.
- `task`: The specified task.

Examples

```
# Clustering
data(iris)
X_iris <- iris[, 1:4]
resC <- honeycombHIVE(
  X = X_iris,
  task = "clustering",
  layers = 3,
  nAntibodies = 15,
  beta = 5,
  maxIter = 10
)
```

---

honeycombHIVEmodel *Mulilayered honeycombHIVE for caret*

Description

A `caret` wrapper for the `honeycombHIVE` function, enabling seamless integration with the `caret` package for hyperparameter tuning, cross-validation, and performance evaluation.

Usage

```
honeycombHIVEmodel
```

Format

An object of class `list` of length 8.

Value

A `caret` model definition list. Pass it to `train` for model training and evaluation.

Parameters

- `nAntibodies`: Number of initial antibodies in the network.
- `beta`: Clone multiplier controlling the number of clones per antibody.
- `epsilon`: Threshold for network suppression to remove redundant antibodies.
- `layers`: Number of hierarchical layers for iterative refinement.
- `refineOptimizer`: Optimizer for gradient-based refinement (e.g. "sgd", "momentum", "adagrad", "adam", "rmsprop").
- `refineSteps`: Number of gradient update steps in refinement.
- `refineLR`: Learning rate for refinement.

Supported Tasks

- `"Classification"`: Assigns class labels to input observations.
- `"Clustering"`: Groups data points based on similarity (though typically `caret` is used for supervised tasks).

See Also

`train`, `trainControl`

Examples

```
## Not run:
  library(caret)
  # Example: Classification with Iris
  data(iris)
  X <- as.matrix(iris[, 1:4])
  y <- iris$Species

  train_control <- trainControl(method = "cv", number = 5)
  set.seed(42)
  model <- train(
    x = X,
    y = y,
    method = honeycombHIVEmodel,
    trControl = train_control,
    tuneGrid = expand.grid(
      nAntibodies = c(10, 20),
      beta = c(3, 5),
      epsilon = c(0.01, 0.05),
      layers = c(1, 2),
      refineOptimizer = "adam",
      refineSteps = 5,
      refineLR = 0.01
    )
  )
  print(model)

## End(Not run)
```

IdiotypicNetwork *IdiotypicNetwork*

Description

Implements Jerne's idiotypic network theory for antibody repertoire regulation. Replaces crude epsilon-threshold suppression with principled network dynamics based on Varela & Coutinho's (1991) second- generation immune network model.

Details

The idiotypic network models antibody-antibody interactions where each antibody's variable region can be recognized by other antibodies. This creates a regulatory network with emergent properties:

- A bell-shaped (double-threshold) activation function: too little stimulation leads to cell death, moderate stimulation to activation, and excessive stimulation to suppression. - Population dynamics with source, decay, activation, and suppression terms. - Self-organized repertoire structure with memory and tolerance properties.

This is the single most novel contribution of the overhauled bHIVE package. No existing AIS implementation uses idiotypic network dynamics for repertoire regulation.

Public fields

`theta_low` Lower activation threshold. Below this, cells die.

`theta_high` Upper activation threshold. Above this, cells are suppressed.

`source_rate` Rate of new cell generation (basal production).

`decay_rate` Natural cell death rate.

`dt` Time step for Euler integration.

`timeSteps` Number of simulation time steps.

`survival_threshold` Minimum population level to survive.

`last_dynamics` Result from the last regulation step.

Methods

Public methods:

- `IdiotypicNetwork$new()`
- `IdiotypicNetwork$regulate()`
- `IdiotypicNetwork$get_network()`
- `IdiotypicNetwork$get_population()`
- `IdiotypicNetwork$print()`
- `IdiotypicNetwork$clone()`

Method `new()`**:** Create a new IdiotypicNetwork regulator.

Usage:

```
IdiotypicNetwork$new(
  theta_low = 0.01,
  theta_high = 0.5,
  source_rate = 0.5,
  decay_rate = 0.1,
  dt = 0.1,
  timeSteps = 20,
  survival_threshold = 0.5
)
```

Arguments:

`theta_low` Lower activation threshold.

`theta_high` Upper activation threshold.

`source_rate` Basal cell production rate.

`decay_rate` Natural decay rate.

`dt` Euler integration time step.

`timeSteps` Number of dynamics simulation steps.

`survival_threshold` Minimum population to survive.

Method `regulate()`**:** Run idiotypic network dynamics on an antibody repertoire.

Usage:

```
IdiotypicNetwork$regulate(
  repertoire,
  affinityFunc = "gaussian",
  affinityParams = list(alpha = 1, c = 1, p = 2)
)
```

Arguments:

`repertoire` An `ImmuneRepertoire` object.

`affinityFunc` Character. Affinity function for Ab-Ab interactions.

`affinityParams` List. Parameters for the affinity function.

Returns: Invisible self. The repertoire is modified in place (dead antibodies removed). Access `$last_dynamics` for full results.

Method `get_network()`**:** Get the Ab-Ab affinity matrix from the last regulation.

Usage:

```
IdiotypicNetwork$get_network()
```

Returns: Numeric matrix (m x m) or NULL if not yet run.

Method `get_population()`**:** Get population levels from the last regulation.

Usage:

```
IdiotypicNetwork$get_population()
```

Returns: Numeric vector or NULL if not yet run.

Method `print()`**:** Print summary.

Usage:

```
IdiotypicNetwork$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
IdiotypicNetwork$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create and run idiotypic regulation
idi <- IdiotypicNetwork$new(theta_low = 0.01, theta_high = 0.5)
A <- matrix(rnorm(50), nrow = 10, ncol = 5)
rep <- ImmuneRepertoire$new(A)
idi$regulate(rep, "gaussian", list(alpha = 0.5))
print(idi)
```

`ImmuneAlgorithm` *ImmuneAlgorithm*

Description

Abstract R6 base class for all immune-inspired algorithms. Subclasses must implement the `fit` method.

Public fields

`repertoire` An `ImmuneRepertoire` object.

`config` Named list of algorithm hyperparameters.

`modules` Named list of injected module instances (SHMEngine, IdiotypicNetwork, GerminalCenter, etc.).

`history` List of per-iteration metrics.

`result` The result from the last call to `fit()`.

Methods

Public methods:

- `ImmuneAlgorithm$new()`
- `ImmuneAlgorithm$fit()`
- `ImmuneAlgorithm$predict()`
- `ImmuneAlgorithm$print()`

- `ImmuneAlgorithm$summary()`
- `ImmuneAlgorithm$clone()`

Method `new()`**:** Create a new ImmuneAlgorithm.

Usage:

```
ImmuneAlgorithm$new(config = list(), modules = list())
```

Arguments:

`config` Named list of hyperparameters.

`modules` Named list of module instances.

Method `fit()`**:** Fit the algorithm to data. Must be overridden by subclasses.

Usage:

```
ImmuneAlgorithm$fit(X, y = NULL, task = NULL, ...)
```

Arguments:

`X` Numeric matrix (n x d).

`y` Optional factor target for classification.

`task` Character: "clustering" or "classification".

`...` Additional arguments.

Returns: The algorithm object (invisibly), with `result` populated.

Method `predict()`**:** Predict on new data using the trained repertoire.

Usage:

```
ImmuneAlgorithm$predict(newdata)
```

Arguments:

`newdata` Numeric matrix (n_new x d).

Returns: Predictions (class labels, numeric values, or cluster IDs).

Method `print()`**:** Print summary of the algorithm.

Usage:

```
ImmuneAlgorithm$print(...)
```

Arguments:

`...` Not used.

Method `summary()`**:** Get a summary of the fitting history.

Usage:

```
ImmuneAlgorithm$summary()
```

Returns: Data frame of per-iteration metrics.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ImmuneAlgorithm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# ImmuneAlgorithm is abstract; use AINet for concrete instances
algo <- ImmuneAlgorithm$new()
print(algo)
```

`ImmuneRepertoire` *ImmuneRepertoire*

Description

R6 class representing a collection of antibodies (immune cells) with associated metadata. Core data structure for bHIVE algorithms.

Details

An ImmuneRepertoire holds a matrix of antibody vectors (each row is one antibody in feature space) plus per-antibody metadata (isotype, state, age, lineage). All heavy computation is dispatched to C++ via RcppArmadillo.

Public fields

`cells` Numeric matrix (nAntibodies x nFeatures).

`metadata` Data frame with per-antibody attributes.

Methods

Public methods:

- `ImmuneRepertoire$new()`
- `ImmuneRepertoire$affinity_matrix()`
- `ImmuneRepertoire$distance_matrix()`
- `ImmuneRepertoire$suppress()`
- `ImmuneRepertoire$size()`
- `ImmuneRepertoire$n_features()`
- `ImmuneRepertoire$subset()`
- `ImmuneRepertoire$add()`
- `ImmuneRepertoire$age_all()`
- `ImmuneRepertoire$as_matrix()`
- `ImmuneRepertoire$print()`
- `ImmuneRepertoire$clone()`

Method `new()`**:** Create a new ImmuneRepertoire.

Usage:

```
ImmuneRepertoire$new(cells, metadata = NULL)
```

Arguments:

`cells` Numeric matrix (nAntibodies x nFeatures).

`metadata` Optional data frame with columns: isotype, state, age, lineage.

Method `affinity_matrix()`**:** Compute affinity matrix between data X and antibodies.

Usage:

```
ImmuneRepertoire$affinity_matrix(
  X,
  method = "gaussian",
  params = list(alpha = 1, c = 1, p = 2)
)
```

Arguments:

`X` Numeric matrix (n x d) of data points.

`method` Affinity function: "gaussian", "laplace", "polynomial", "cosine", "hamming".

`params` List with alpha, c, p parameters.

Returns: Numeric matrix (n x m) of affinity values.

Method `distance_matrix()`**:** Compute distance matrix between data X and antibodies.

Usage:

```
ImmuneRepertoire$distance_matrix(
  X,
  method = "euclidean",
  params = list(p = 2, Sigma = NULL)
)
```

Arguments:

`X` Numeric matrix (n x d).

`method` Distance function: "euclidean", "manhattan", "minkowski", "cosine", "mahalanobis", "hamming".

`params` List with p, Sigma parameters.

Returns: Numeric matrix (n x m) of distances.

Method `suppress()`**:** Network suppression: remove redundant antibodies.

Usage:

```
ImmuneRepertoire$suppress(
  epsilon,
  method = "euclidean",
  params = list(p = 2, Sigma = NULL)
)
```

Arguments:

`epsilon` Distance threshold for suppression.

`method` Distance function for suppression.

`params` List with p, Sigma parameters.

Returns: Invisible self (modified in place).

Method `size()`: Get number of antibodies.

Usage:

```
ImmuneRepertoire$size()
```

Returns: Integer.

Method `n_features()`: Get number of features.

Usage:

```
ImmuneRepertoire$n_features()
```

Returns: Integer.

Method `subset()`: Subset the repertoire.

Usage:

```
ImmuneRepertoire$subset(idx)
```

Arguments:

`idx` Integer vector of row indices to keep.

Returns: Invisible self (modified in place).

Method `add()`: Add antibodies to the repertoire.

Usage:

```
ImmuneRepertoire$add(new_cells, new_metadata = NULL)
```

Arguments:

`new_cells` Numeric matrix (k x d) of new antibodies.

`new_metadata` Optional data frame of metadata for new antibodies.

Returns: Invisible self (modified in place).

Method `age_all()`: Increment age of all antibodies.

Usage:

```
ImmuneRepertoire$age_all()
```

Returns: Invisible self (modified in place).

Method `as_matrix()`: Convert to plain matrix.

Usage:

```
ImmuneRepertoire$as_matrix()
```

Returns: Numeric matrix (nAntibodies x nFeatures).

Method `print()`: Print summary.

Usage:

```
ImmuneRepertoire$print(...)
```

Arguments:

`...` Not used.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ImmuneRepertoire$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a repertoire from random antibodies
A <- matrix(rnorm(50), nrow = 10, ncol = 5)
rep <- ImmuneRepertoire$new(A)
print(rep)

# Compute affinity to data
X <- matrix(rnorm(100), nrow = 20, ncol = 5)
aff <- rep$affinity_matrix(X, "gaussian", list(alpha = 1))
dim(aff)  # 20 x 10

# Network suppression
rep$suppress(epsilon = 1.5, method = "euclidean")
rep$size()  # fewer antibodies after suppression
```

`MemoryPool` *MemoryPool*

Description

Manages long-lived memory cells that can be recalled when distribution shifts are detected. Implements the immunological distinction between short-lived effector cells and long-lived memory cells.

Public fields

`memory_cells` Numeric matrix of archived memory antibodies.

`memory_metadata` Data frame of metadata for memory cells.

`archive_threshold` Affinity threshold for archiving (only high-quality antibodies become memory).

`max_memory` Maximum number of memory cells to store.

`recall_threshold` Threshold for triggering memory recall.

Methods

Public methods:

- `MemoryPool$new()`
- `MemoryPool$archive()`
- `MemoryPool$recall()`
- `MemoryPool$size()`
- `MemoryPool$print()`
- `MemoryPool$clone()`

Method `new()`**:** Create a new MemoryPool.

Usage:

```
MemoryPool$new(
  archive_threshold = 0.5,
  max_memory = 100,
  recall_threshold = 0.3
)
```

Arguments:

`archive_threshold` Numeric. Minimum average affinity to archive.

`max_memory` Integer. Maximum memory cells.

`recall_threshold` Numeric. Minimum affinity to recall a memory.

Method `archive()`**:** Archive high-performing antibodies as memory cells.

Usage:

```
MemoryPool$archive(
  repertoire,
  X,
  affinityFunc = "gaussian",
  affinityParams = list(alpha = 1, c = 1, p = 2)
)
```

Arguments:

`repertoire` An `ImmuneRepertoire`.

`X` Training data matrix.

`affinityFunc` Character. Affinity function.

`affinityParams` List. Affinity parameters.

Returns: Integer. Number of new memory cells archived.

Method `recall()`**:** Recall memory cells relevant to current data.

Usage:

```
MemoryPool$recall(
  X,
  affinityFunc = "gaussian",
  affinityParams = list(alpha = 1, c = 1, p = 2)
)
```

Arguments:

`X` Data matrix to match against memory.

`affinityFunc` Character. Affinity function.

`affinityParams` List. Affinity parameters.

Returns: Numeric matrix of recalled memory cells (may be empty).

Method `size()`**:** Get current memory pool size.

Usage:

```
MemoryPool$size()
```

Returns: Integer.

Method `print()`**:** Print summary.

Usage:

```
MemoryPool$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MemoryPool$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Archive and recall memory cells
data(iris)
X <- as.matrix(iris[, 1:4])
A <- X[sample(150, 10), ]
mp <- MemoryPool$new(archive_threshold = 0.01)
rep <- ImmuneRepertoire$new(A)
mp$archive(rep, X)
mp$size()  # number of archived memories
recalled <- mp$recall(X[1:5, ])
nrow(recalled)  # memories relevant to query
```

`Microenvironment` *Microenvironment*

Description

Models local microenvironment cues that influence antibody behavior based on the density and structure of nearby data points.

Details

In real immunity, B cell fate is strongly influenced by local signals: chemokines, cytokines, and interactions with stromal cells in specific tissue microenvironments. This module translates that concept into density-dependent adaptation:

- **High density zones**: Promote memory formation (stabilize antibodies, reduce mutation rate)
- **Low density zones**: Promote exploration (increase mutation rate for broader search)
- **Boundary zones**: Trigger class switching (change matching breadth between IgM-like broad and IgG-like specific modes)
- **Chemokine-like gradients**: Bias mutation direction toward higher-density regions

Public fields

`density_bandwidth` Bandwidth for kernel density estimation.

`high_density_threshold` Density percentile above which antibodies stabilize.

`low_density_threshold` Density percentile below which antibodies explore.

`stabilization_factor` Mutation rate multiplier for high-density zones.

`exploration_factor` Mutation rate multiplier for low-density zones.

`last_densities` Per-antibody local density from last assessment.

`last_zones` Per-antibody zone classification from last assessment.

Methods

Public methods:

- `Microenvironment$new()`
- `Microenvironment$assess()`
- `Microenvironment$print()`
- `Microenvironment$clone()`

Method `new()`**:** Create a new Microenvironment module.

Usage:

```
Microenvironment$new(
  density_bandwidth = NULL,
  high_density_threshold = 0.75,
  low_density_threshold = 0.25,
  stabilization_factor = 0.3,
  exploration_factor = 2
)
```

Arguments:

`density_bandwidth` Numeric. KDE bandwidth (NULL for auto).

`high_density_threshold` Numeric [0,1]. Percentile threshold for stabilization.

`low_density_threshold` Numeric [0,1]. Percentile threshold for exploration.

`stabilization_factor` Numeric. Mutation rate multiplier for stable zones.

`exploration_factor` Numeric. Mutation rate multiplier for exploration zones.

Method `assess()`**:** Assess the microenvironment around each antibody.

Usage:

```
Microenvironment$assess(
  repertoire,
  X,
  affinityFunc = "gaussian",
  affinityParams = list(alpha = 1, c = 1, p = 2)
)
```

Arguments:

`repertoire` An `ImmuneRepertoire` object.

`X` Numeric matrix of training data (n x d).

`affinityFunc` Character. Affinity function.

`affinityParams` List. Affinity parameters.

Returns: Named list with densities, zones, and mutation_modifiers per antibody.

Method `print()`**:** Print summary.

Usage:

```
Microenvironment$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Microenvironment$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Assess microenvironment around antibodies
data(iris)
X <- as.matrix(iris[, 1:4])
me <- Microenvironment$new()
rep <- ImmuneRepertoire$new(X[sample(150, 15), ])
env <- me$assess(rep, X)
table(env$zones)  # stable, explore, boundary
env$mutation_modifiers  # per-antibody rate scaling
```

`refineB` *refineB: Gradient-based fine-tuning for bHIVE antibodies with multiple loss functions and optimizers*

Description

After running `bHIVE` (or within `honeycombHIVE`), you have a set of final antibody positions (A) in feature space. This function refines those prototypes by iterating over the data points assigned to each antibody and applying gradient-based updates using a user-chosen loss function and optimizer.

Usage

```
refineB(
  A,
  X,
  y = NULL,
  assignments,
  task = c("clustering", "classification"),
 loss = c("categorical_crossentropy", "binary_crossentropy", "kullback_leibler",
    "cosine", "mae"),
  steps = 5,
  lr = 0.01,
  push_away = TRUE,
  verbose = TRUE,
  optimizer = c("sgd", "momentum", "adagrad", "adam", "rmsprop"),
  momentum_coef = 0.9,
  beta1 = 0.9,
  beta2 = 0.999,
  rmsprop_decay = 0.9,
  epsilon = 1e-08
)
```

Arguments

A	Numeric matrix (nAb x d) of antibody/prototype positions.
X	Matrix or data frame (nSamples x d) of feature data.
y	Optional. Factor (classification) or NULL (clustering).
assignments	Integer or character vector (length = nSamples), specifying the antibody index (or label) each sample belongs to.
task	One of c("clustering", "classification").
loss	One of c("categorical_crossentropy", "binary_crossentropy","kullback_leibler", "cosine", "mae").
steps	Integer. How many gradient steps to run.
lr	Numeric. Learning rate for each update.
push_away	Logical (for classification). Whether to push prototypes away from differently-labeled samples.
verbose	Logical. If TRUE, prints progress messages each iteration.
optimizer	One of c("sgd", "momentum", "adagrad", "adam", "rmsprop"). Specifies the gradient-based optimization approach.
momentum_coef	Numeric. Momentum coefficient (used if optimizer=="momentum").
beta1	Numeric. Exponential decay rate for the first moment estimates (used in Adam).
beta2	Numeric. Exponential decay rate for the second moment estimates (used in Adam).
rmsprop_decay	Numeric. Decay factor for the squared gradient moving average (used in RMSProp).
epsilon	Numeric. A small constant for numerical stability.

Details

The user must provide: - A numeric matrix `A` of antibody/prototype positions (nAb x nFeatures). - A numeric matrix/data frame `X` of data (nSamples x nFeatures). - Optional `y` for classification. If `task="clustering"`, `y` can be NULL or ignored. - An integer or character vector `assignments` (length=nSamples) giving the antibody index (or label) to which each data point is assigned.

Available Losses

Classification (factor y) - **"categorical_crossentropy"**: Pull prototypes toward data points if they share the antibody's dominant label; push away otherwise. - **"binary_crossentropy"**: Similar to categorical CE, but we interpret factor y as binary (two classes). Pull for same label, push for different label. - **"kullback_leibler"**: Very rough approach that treats "dominant label vs. others" as p and q distributions, pushing/pulling prototypes. - **"cosine"**: Interpreted as trying to maximize cosine similarity for same-label points and minimize for different-label points. - **"mae"**: Sign-based pull/push.

Available Optimizers - **"sgd"**: Basic stochastic gradient descent. - **"momentum"**: SGD with momentum. - **"adagrad"**: Adaptive gradient descent. - **"adam"**: Adaptive moment estimation. - **"rmsprop"**: Root Mean Square Propagation.

This function performs gradient-based updates on each antibody using the selected loss function. Depending on the chosen `optimizer`, it may use plain SGD, momentum-based updates, Adagrad, Adam, or RMSProp.

Value

Updated matrix `A` of shape (nAb x d).

Examples

```
data(iris)
X <- as.matrix(iris[, 1:4])
y <- iris$Species
res <- bHIVE(X, y, task = "classification", nAntibodies = 10,
             maxIter = 5, verbose = FALSE)
assignments <- as.integer(factor(res$assignments,
                                 levels = unique(res$assignments)))
A_refined <- refineB(res$antibodies, X, y = y,
                     assignments = assignments,
                     task = "classification",
                     loss = "categorical_crossentropy", optimizer = "adam",
                     steps = 3, lr = 0.01, verbose = FALSE)
dim(A_refined)
```

`SHMEngine` *SHMEngine*

Description

Somatic Hypermutation Engine implementing five biologically- inspired mutation strategies for the bHIVE artificial immune system.

Details

The five strategies are:

uniform Classic random Gaussian noise. Mutation rate = (1-affinity) * decay^(iter-1). This is the original bHIVE behavior.

airs AIRS-style affinity-proportional mutation. Rate = c * exp(-affinity / T). From Watkins & Timmis (AIRS2), achieving 50% better data reduction than uniform.

hotspot Feature-importance-weighted mutation. Features with higher gradient magnitude mutate more, analogous to AID targeting WRCY motifs in real SHM.

energy Energy-budget-constrained mutation. Total mutation magnitude bounded by E = E_0 * (1-affinity)^2, inspired by Kleinstein's E_SHM ~ N_Mut^2 model.

adaptive Per-feature adaptive mutation rate with moment tracking, directly implementing SHM as the Adam optimizer.

Public fields

`method` Character. One of "uniform", "airs", "hotspot", "energy", "adaptive".

`params` Named list of method-specific parameters.

`m1_state` First moment state matrix (for adaptive method).

`m2_state` Second moment state matrix (for adaptive method).

Methods

Public methods:

- `SHMEngine$new()`
- `SHMEngine$init_state()`
- `SHMEngine$reset_state()`
- `SHMEngine$print()`
- `SHMEngine$clone()`

Method `new()`**:** Create a new SHMEngine.

Usage:

```
SHMEngine$new(
  method = "uniform",
  decay = 1,
  mutationMin = 0.01,
  c_rate = 1,
  temperature = 0.5,
  E_0 = 1,
  base_rate = 0.1,
  beta1 = 0.9,
  beta2 = 0.999,
  adam_epsilon = 1e-08
)
```

Arguments:

`method` Character. Mutation strategy.
`decay` Numeric. Per-iteration mutation rate decay (uniform method).
`mutationMin` Numeric. Minimum mutation rate floor.
`c_rate` Numeric. Scaling constant (airs method).
`temperature` Numeric. Temperature parameter (airs method).
`E_0` Numeric. Energy budget base (energy method).
`base_rate` Numeric. Base mutation rate (hotspot, adaptive methods).
`beta1` Numeric. First moment decay (adaptive method, like Adam).
`beta2` Numeric. Second moment decay (adaptive method, like Adam).
`adam_epsilon` Numeric. Numerical stability (adaptive method).

Method `init_state()`**:** Initialize internal state for adaptive method.

Usage:

```
SHMEngine$init_state(nAntibodies, nFeatures)
```

Arguments:

`nAntibodies` Integer. Number of antibodies.
`nFeatures` Integer. Number of features.

Method `reset_state()`**:** Reset moment states (e.g., after suppression changes antibody count).

Usage:

```
SHMEngine$reset_state(nAntibodies, nFeatures, kept_idx = NULL)
```

Arguments:

`nAntibodies` Integer. New number of antibodies.
`nFeatures` Integer. Number of features.
`kept_idx` Integer vector. Indices of antibodies that were kept.

Method `print()`**:** Print summary.

Usage:

```
SHMEngine$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SHMEngine$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create different SHM engines
shm_uniform  <- SHMEngine$new(method = "uniform")
shm_adaptive <- SHMEngine$new(method = "adaptive", base_rate = 0.1)
shm_airs     <- SHMEngine$new(method = "airs", temperature = 0.3)
print(shm_adaptive)
```

swarmbHIVE *Tune Hyperparameters for bHIVE (Swarm/Grid Search)*

Description

Performs hyperparameter tuning for the bHIVE algorithm over a grid of hyperparameter values or an externally provided data frame of parameter combinations. Evaluates each combination using different metrics:

Usage

```
swarmbHIVE(
  X,
  y = NULL,
  task = c("clustering", "classification"),
  grid,
  metric = NULL,
  maxIter = 50,
  BPPARAM = SerialParam(),
  verbose = TRUE
)
```

Arguments

X	A numeric matrix or data frame of input features (rows = observations, columns = features).
y	Optional. A factor target vector for classification. If NULL, clustering is performed.
task	Character. One of "clustering" or "classification".
grid	A data frame specifying the hyperparameter combinations. Should have columns: nAntibodies, beta, epsilon. (Optionally more if you want to pass other arguments to bHIVE().)
metric	Character. Name of the evaluation metric. Options: • **Classification**: "accuracy", "balanced_accuracy", "f1", "kappa" • **Clustering**: "silhouette", "davies_bouldin", "calinski_harabasz"
maxIter	Integer. Maximum iterations for each bHIVE run (default 50).
BPPARAM	Character. A BiocParallel::bpparam() object that can be used for parallelization. The function supports SerialParam, MulticoreParam, BatchtoolsParam, and SnowParam.
verbose	Logical. If TRUE, prints progress messages.

Details

- **Classification**: "accuracy", "balanced_accuracy", "f1", "kappa" - **Clustering**: "silhouette", "davies_bouldin", or "calinski_harabasz"

Note: Some metrics require additional packages or assumptions (e.g., multi-class classification for "f1" is calculated as a macro-average).

Value

A list:

- `best_params`: A list (row) of the best hyperparameters.
- `results`: A data frame with the full grid search results, including the `metric_value` for each combination.

Examples

```
data(iris)
X <- as.matrix(iris[, 1:4])
y <- iris$Species  # classification

# Define hyperparameter grid
grid <- expand.grid(
  nAntibodies = c(10, 20),
  beta        = c(3, 5),
  epsilon     = c(0.01, 0.05)
)

# Perform hyperparameter tuning for classification
tuning_results <- swarmbHIVE(X = X,
                             y = y,
                             task = "classification",
                             grid = grid,
                             metric = "balanced_accuracy",
                             maxIter = 10)

# For clustering with silhouette
set.seed(42)
X_clust <- matrix(rnorm(100 * 5), ncol = 5)
grid_clust <- expand.grid(nAntibodies = c(5, 10),
                          beta = c(3, 5),
                          epsilon = c(0.01, 0.05))
res_clust <- swarmbHIVE(X_clust,
                        task = "clustering",
                        grid = grid_clust,
                        metric = "silhouette")
res_clust$best_params
```

VDJLibrary *VDJLibrary*

Description

V(D)J gene library initialization for antibody populations. Instead of random sampling, antibodies are generated by combinatorial assembly of gene segments, mimicking the V(D)J recombination that generates antibody diversity in real immune systems.

Details

The feature space is decomposed into V, D, and J segments (subsets of dimensions). Each segment is discretized into "alleles" by clustering. New antibodies are generated by combinatorial sampling of one allele from each segment, producing structured coverage of the data manifold.

Methods:

- `"pca"`: PCA components define gene segments
- `"cluster"`: k-means within dimension groups creates alleles
- `"random_partition"`: Random feature grouping

Public fields

`nV` Number of V gene segments.

`nD` Number of D gene segments.

`nJ` Number of J gene segments.

`method` Decomposition method.

`library` The computed gene library (list of allele matrices).

Methods

Public methods:

- `VDJLibrary$new()`
- `VDJLibrary$build()`
- `VDJLibrary$generate()`
- `VDJLibrary$print()`
- `VDJLibrary$clone()`

Method `new()`**:** Create a new VDJLibrary.

Usage:

```
VDJLibrary$new(nV = 10, nD = 5, nJ = 4, method = "pca")
```

Arguments:

`nV` Integer. Number of V gene alleles.

`nD` Integer. Number of D gene alleles.

`nJ` Integer. Number of J gene alleles.

`method` Character. "pca", "cluster", or "random_partition".

Method `build()`**:** Build the gene library from training data.

Usage:

```
VDJLibrary$build(X)
```

Arguments:

`X` Numeric matrix (n x d).

Returns: Invisible self.

Method `generate()`**:** Generate antibodies by combinatorial V(D)J assembly.

Usage:

```
VDJLibrary$generate(nAntibodies, X)
```

Arguments:

`nAntibodies` Integer. Number of antibodies to generate.

`X` Numeric matrix. Training data (used to build library if needed).

Returns: Numeric matrix (nAntibodies x d).

Method `print()`**:** Print summary.

Usage:

```
VDJLibrary$print(...)
```

Arguments:

`...` Not used.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VDJLibrary$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Generate antibodies via V(D)J combinatorial assembly
data(iris)
X <- as.matrix(iris[, 1:4])
vdj <- VDJLibrary$new(nV = 5, nD = 3, nJ = 3, method = "pca")
A <- vdj$generate(20, X)
dim(A)  # 20 x 4
print(vdj)
```

visualizeHIVE *Visualize bHIVE/honeycombHIVE Results*

Description

This function produces publication-quality visualizations for the results generated by the bHIVE or honeycombHIVE functions. Users can specify one or more layers to visualize and choose from several plot types:

Usage

```
visualizeHIVE(
  result,
  X = NULL,
  plot_type = c("scatter", "boxplot", "violin", "density"),
  feature = NULL,
  transform = TRUE,
  transformation_method = c("PCA", "UMAP", "tSNE", "none"),
  title = "HIVE Results",
  layer = 1,
  task = c("clustering", "classification"),
  ...
)
```

Arguments

result	A list object produced by bHIVE or honeycombHIVE. For multilayer models, each element represents one layer.
X	Optional. A numeric matrix or data frame of the original input features. If provided, data points will be plotted along with the prototypes.
plot_type	Character string specifying the type of plot to generate. Options are: • "scatter": A scatterplot of data points and prototypes. • "boxplot": A boxplot of a selected feature by group with prototypes overlaid. • "violin": A violin plot of a selected feature by group with prototypes overlaid. • "density": Density plots of a selected feature by group with prototype markers.
feature	Optional. For "boxplot", "violin", or "density" plots, the name or index of the feature in X to display. If NULL, the first column is used.
transform	Logical. If TRUE and the data (or prototypes) has more than two columns, the specified transformation is applied for scatterplots.
transformation_method	Character. The method used for dimensionality reduction. Options are "PCA", "UMAP", "tSNE", or "none".

title	Character. Title for the plot.
layer	Integer vector indicating which layer(s) of the result to visualize. For bHIVE outputs, default is 1. For multilayer honeycombHIVE outputs, specify one or more layer indices.
task	Character. The prediction task for the result: one of "clustering" or "classification". This is used to determine how grouping is computed.
...	Additional arguments passed to PCA, tSNE, UMAP, or ggplot functions.

Details

- "scatter": A scatterplot of data points and prototypes. When X has more than two columns and transform is TRUE, a dimensionality reduction method is applied.
- "boxplot": A boxplot of a selected feature by group with prototype values overlaid.
- "violin": A violin plot of a selected feature by group with prototype values overlaid.
- "density": Density plots of a selected feature by group with prototype markers.

For scatterplots the transformation can be selected from "PCA", "UMAP", "tSNE", or "none". When multiple layers are visualized, the prototypes and the corresponding grouping information from each layer are combined and faceted by layer.

Value

A ggplot object representing the visualization.

Examples

```
data(iris)
X <- as.matrix(iris[, 1:4])

# Run honeycombHIVE for clustering
res <- honeycombHIVE(X = X,
                     task = "clustering",
                     epsilon = 0.05,
                     layers = 3,
                     nAntibodies = 30,
                     beta = 5,
                     maxIter = 10,
                     verbose = FALSE)

# Visualize layer 2 as a scatterplot (using membership from layer 2).
visualizeHIVE(result = res,
              X = iris[, 1:4],
              plot_type = "scatter",
              title = "Layer 2: Scatterplot",
              layer = 2,
              task = "clustering")

# For classification: assume res[[1]]$predictions holds class labels.
visualizeHIVE(result = res,
              X = iris[, 1:4],
```

```
plot_type = "violin",
feature = "Sepal.Width",
title = "Sepal Width by Group (Layer 1)",
layer = 1,
task = "classification")
```

Index