Package: bHIVE (via r-universe)

November 8, 2026

Title B-cell Hybrid Immune Variant Engine

Version 0.99.8

Description bHIVE implements an artificial immune network (AI-Net) for clustering and classification of high-dimensional data such as mass cytometry and single-cell profiles. A population of prototypes, called antibodies, evolves by clonal selection, somatic hypermutation and network suppression, and the learned prototypes summarize cell populations in the original marker units. The same fitted model clusters a dataset and labels new cells. bHIVE accepts SummarizedExperiment and SingleCellExperiment objects and integrates with bluster through bHIVEParam. A C++ backend computes affinities, mutations and suppression, and optional R6 modules add mechanisms such as germinal center selection, microenvironment-dependent mutation and isotype switching. The package also provides multilayer clustering, grid search and caret model wrappers.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

biocViews Software, Clustering, Classification, SingleCell, FlowCytometry, CellBasedAssays, Network

Depends R (>= 4.5.0)

Imports BiocParallel, bluster, cluster, clusterCrit, ggplot2 (>= 3.4.0), methods, R6, Rcpp, S4Vectors, SingleCellExperiment, stats, SummarizedExperiment

LinkingTo Rcpp, RcppArmadillo

Suggests BiocStyle, caret, FlowSOM, HDCytoData, knitr, rmarkdown, Rtsne, spelling, testthat (>= 3.0.0), umap

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 libglpk-dev libxml2-dev zlib1g-dev

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

Date/Publication 2026-10-16 18:55:33 UTC

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

RemoteRef HEAD

RemoteSha bcab3c85a0ef081a6c9831d84fd73659fe4eae8d

Contents

ActivationGate . . . 2
AINet . . . 4
bHIVE . . . 8
bHIVEmodel . . . 12
bHIVEParam . . . 13
ClassSwitcher . . . 15
ConvergentSelector . . . 17
GerminalCenter . . . 19
honeycombHIVE . . . 21
honeycombHIVEmodel . . . 24
IdiotypicNetwork . . . 25
ImmuneAlgorithm . . . 28
ImmuneRepertoire . . . 30
MemoryPool . . . 33
Microenvironment . . . 36
refineB . . . 38
SHMEngine . . . 40
swarmbHIVE . . . 43
VDJLibrary . . . 45
visualizeHIVE . . . 47

Index **50**

`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)

Value

An `R6Class` generator. `ActivationGate$new()` returns an `ActivationGate` object.

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$format()`
- `ActivationGate$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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
ActivationGate$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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.

Value

An R6Class generator. AINet$new() returns an AINet object. Call $fit() on it to train, then $predict() to label new data.

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",
  consolidate = TRUE,
  consolidationSteps = 10L,
  scale = c("none", "zscore", "robust", "arcsinh"),
  scaleCofactor = 5,
  targetK = NULL,
  epsilonQuantile = NULL,
  coverageBoost = FALSE,
  coverageQuantile = 0.05,
  minPerClass = 5,
  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.

`consolidate` Logical. For clustering, run Lloyd-style consolidation (an M-step) after affinity maturation so antibodies are pulled onto the data manifold and become true data-space prototypes. Has no effect on classification. Default TRUE.

`consolidationSteps` Integer. Maximum consolidation iterations.

`scale` Character. Per-feature input scaling applied at `fit()` and re-applied to new data at `predict()`. One of `"none"` (default, no transform), `"zscore"` (center/SD), `"robust"` (median / IQR, outlier-tolerant), or `"arcsinh"` (inverse hyperbolic sine with cofactor `scaleCofactor`, the standard mass-cytometry transform). Because `epsilon`, mutation scale, and all distances live in feature units, scaling makes the same defaults behave consistently across datasets of different magnitude.

`scaleCofactor` Numeric. Cofactor for `scale = "arcsinh"` (`asinh(x / cofactor)`). Default 5 (CyTOF convention; use ~150 for fluorescence flow).

`targetK` Integer or NULL. If set, force the clustering solution to exactly `targetK` clusters. Affinity maturation still discovers where prototypes belong, but the final consolidation seeds a K-means (Lloyd) refinement at exactly `targetK` centroids: surviving antibodies are agglomerated (if more than K) or split with k-means++ (if fewer than K) before refinement. This decouples the reported cluster count from the emergent suppression dynamics. NULL (default) keeps the emergent, self-selected K. Ignored for classification.

`epsilonQuantile` Numeric in (0, 1) or NULL. If set, the suppression threshold is recomputed each iteration as this quantile of the pairwise distances among the current antibodies, making suppression scale-free and adaptive instead of using the fixed `epsilon`. NULL (default) uses the fixed `epsilon`.

`coverageBoost` Logical. Clustering only. After maturation, find data points that no surviving antibody covers well (max affinity in the bottom `coverageQuantile` tail) and seed fresh antibodies there with k-means++. Counters the clonal-selection bias toward dense regions, which otherwise leaves rare populations unrepresented. Pairs naturally with `targetK`: the extra seeds give the forced-K refinement candidate prototypes for sparse populations. Default FALSE.

`coverageQuantile` Numeric in (0, 1). Affinity-coverage tail that defines "poorly covered" points for `coverageBoost`. Default 0.05.

`minPerClass` Integer. Classification only. Antibodies are seeded per class: each class gets `max(minPerClass, round(nAntibodies * class share))` antibodies drawn from its own observations with `initMethod`, and starts with that class label. Without this, a rare class (for example a population of a few cells in a cytometry sample) often receives no antibody at all and can never be predicted. The total can slightly exceed `nAntibodies`. Set to 0 to sample all observations without regard to class. Ignored when an `init` module is supplied. Default 5. The guarantee holds at seeding only: suppression and idiotypic regulation remove antibodies afterwards and can take every one a class had. `fit()` reports any class that ends with no antibody or that recovers less than half of its own training observations, so raise `nAntibodies` or lower `epsilon` when that happens.

`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. Report progress with `message()`, so it can be silenced with `suppressMessages()`.

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, ...)

## S4 method for signature 'ANY'
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,
  scale = c("none", "zscore", "robust", "arcsinh"),
  targetK = NULL,
  epsilonQuantile = NULL,
  verbose = TRUE,
  ...
```

```
)

## S4 method for signature 'SummarizedExperiment'
bHIVE(X, y = NULL, ..., assay.type = "logcounts", name = "bHIVE")

## S4 method for signature 'SingleCellExperiment'
bHIVE(
  X,
  y = NULL,
  ...,
  assay.type = "logcounts",
  use.dimred = NULL,
  name = "bHIVE"
)
```

Arguments

X	A numeric matrix or data frame of input features, with rows as observations and columns as features. Or a SummarizedExperiment or SingleCellExperiment with features in rows and cells in columns.
...	Additional arguments forwarded to AINet, including immunology modules (shm, idiotypic, germinalCenter, microenvironment, activation, memory, classSwitcher, init), consolidate / consolidationSteps, coverageBoost and, for classification, minPerClass (antibodies seeded per class, default 5). For the SummarizedExperiment methods, all arguments of the matrix method.
y	Optional. A factor target vector for classification. If NULL, clustering will be performed. For SummarizedExperiment input, y may also be the name of a colData() column holding the labels.
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). "auto" sets the bandwidth by the median heuristic, which is recommended for high-dimensional data such as mass cytometry.

- 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.

scale — Character. Per-feature input scaling: "none" (default), "zscore", "robust" (median/IQR), or "arcsinh" (CyTOF). Passed to AINet; makes epsilon and distances behave consistently across datasets of different magnitude.

targetK — Integer or NULL. If set, force the clustering result to exactly targetK clusters via a seeded K-means refinement (the immune network supplies the seeds). NULL (default) keeps the emergent cluster count.

epsilonQuantile — Numeric in (0, 1) or NULL. If set, the suppression threshold adapts each iteration to this quantile of pairwise antibody distances instead of the fixed epsilon.

verbose — Logical. If TRUE, reports progress each iteration with message().

assay.type — String or integer. The assay holding the features to cluster, for SummarizedExperiment input. Ignored when use.dimred is set.

name — String. The colData() column (and metadata() entry) the results are stored under.

use.dimred — String or integer. The reducedDim() embedding to cluster, for SingleCellExperiment input. Takes precedence over assay.type. Recommended for sparse single-cell RNA-seq, since an assay is converted to a dense matrix.

Details

bHIVE() accepts a plain matrix or data frame (observations in rows) or a SummarizedExperiment / SingleCellExperiment (features in rows, cells in columns, the Bioconductor convention). For the latter, the cells are clustered or classified using an assay or, for a SingleCellExperiment, a reducedDim() embedding, and the labels are stored in colData(). To use bHIVE through **bluster**, see bHIVEParam.

Value

For matrix or data frame input, a list:

- `antibodies`: Final prototype matrix (prototypes x features), in the scaled space the model was trained in.
- `assignments`: For clustering, integer cluster IDs, where cluster `i` is row `i` of `antibodies`. For classification, the predicted class labels.
- `antibody_classes`: Classification only. The class label of each antibody.
- `antibody_index`: Classification only. The row of `antibodies` that labeled each observation.
- `antibodies_unscaled`: The prototypes back-transformed to the original feature units.
- `task`: The chosen task.
- `model`: The fitted `AINet` object. Use `res$model$predict(newdata)` to label new observations.

For SummarizedExperiment or SingleCellExperiment input, `X` with the labels added as a factor column `colData(X)[[name]]` and the full result list above in `metadata(X)[[name]]`.

See Also

`bHIVEParam` for **bluster** integration, `AINet` for the underlying R6 engine.

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)

# Example 3: SingleCellExperiment input (features x cells)
```

```
library(SingleCellExperiment)
sce <- SingleCellExperiment(assays = list(logcounts = t(X)),
                            colData = DataFrame(species = iris$Species))
sce <- bHIVE(sce, targetK = 3, maxIter = 10, verbose = FALSE)
table(sce$bHIVE, sce$species)

# Cluster on an embedding, or classify with a colData column as labels
reducedDim(sce, "PCA") <- prcomp(X)$x[, 1:2]
sce <- bHIVE(sce, use.dimred = "PCA", name = "bHIVE_pca", verbose = FALSE)
sce <- bHIVE(sce, y = "species", name = "bHIVE_class", verbose = FALSE)
mean(sce$bHIVE_class == sce$species)
```

`bHIVEmodel` *bHIVE classification model for caret*

Description

A custom model definition that lets `caret::train()` tune, resample and evaluate `bHIVE` classifiers.

Usage

```
bHIVEmodel
```

Format

A list with the components **caret** expects of a custom model.

Details

Pass `bHIVEmodel` as the `method` of `caret::train()`. Training fits `bHIVE` with the tuned `nAntibodies`, `beta` and `epsilon`. Further arguments given to `caret::train()` (for example `maxIter`, `scale` or immunology modules) are passed on to `bHIVE()`. Progress output is off unless `verbose = TRUE` is passed. Prediction labels each new observation with the class of its best-matching antibody, using the fitted `AINet` model, so new data gets the same scaling and kernel as the training data.

caret requires column names on `x`.

Value

A list defining a custom **caret** model, for use as `caret::train(method = bHIVEmodel)`.

Components

`label` "Artificial Immune Network (bHIVE)".

`library` "bHIVE".

`type` "Classification".

`parameters` The tunable parameters: `nAntibodies` (initial antibody count), `beta` (clone multiplier) and `epsilon` (suppression threshold).

`grid function(x, y, len, search)`. A default tuning grid, regular (`search = "grid"`) or random.

`fit function(x, y, wts, param, lev, last, classProbs, ...)`. Fits `bHIVE()` and returns its result.

`predict function(modelFit, newdata, submodels)`. Returns a factor of predicted classes.

`prob NULL`. Class probabilities are not implemented, so `caret::train()` warns and continues if `classProbs = TRUE`.

See Also

bHIVE, honeycombHIVEmodel, train

Examples

```
bHIVEmodel$label
bHIVEmodel$parameters

if (requireNamespace("caret", quietly = TRUE)) {
    X <- as.matrix(iris[, 1:4])
    y <- iris$Species
    set.seed(42)
    fit <- caret::train(
        x = X, y = y, method = bHIVEmodel,
        trControl = caret::trainControl(method = "cv", number = 3),
        tuneGrid = data.frame(nAntibodies = 40, beta = 5, epsilon = 0.05),
        maxIter = 10
    )
    fit
    table(Predicted = predict(fit, X), Actual = y)
}
```

`bHIVEParam` *bHIVE clustering through bluster*

Description

Run bHIVE clustering through clusterRows, the **bluster** interface that `bluster::clusterSweep()` builds on. This makes bHIVE a drop-in alternative to the k-means, SOM and graph-based clustering that **bluster** already wraps.

Usage

```
bHIVEParam(
  nAntibodies = NULL,
  beta = NULL,
  epsilon = NULL,
  maxIter = NULL,
```

```
  k = NULL,
  affinityFunc = NULL,
  distFunc = NULL,
  initMethod = NULL,
  scale = NULL,
  targetK = NULL,
  epsilonQuantile = NULL,
  affinityParams = list(),
  ...
)

## S4 method for signature 'bHIVEParam'
.defaultScalarArguments(x)

## S4 method for signature 'bHIVEParam'
show(object)

## S4 method for signature 'ANY,bHIVEParam'
clusterRows(x, BLUSPARAM, full = FALSE)
```

Arguments

nAntibodies, beta, epsilon, maxIter, k, affinityFunc, distFunc, initMethod, scale, targetK, epsilonQuantile	Scalar arguments passed to bHIVE. NULL (the default) uses the bHIVE() default. These can be varied with bluster::clusterSweep().
affinityParams	Named list passed to bHIVE(), for example list(alpha = "auto"). An empty list uses the bHIVE() default.
...	Further named arguments passed to bHIVE() and AINet, such as coverageBoost, consolidate, verbose or immunology modules (shm, idiotypic, ...). Note that R6 modules carry state, so a MemoryPool passed here is shared across every run that uses this parameter object.
x	A numeric matrix-like object with observations in rows, or a bHIVEParam for show().
object	A bHIVEParam object.
BLUSPARAM	A bHIVEParam object.
full	Logical. Return the full bHIVE() result as well as the clusters.

Details

bHIVE learns a set of prototypes (antibodies) by clonal selection, mutation and network suppression, then assigns each observation to its nearest prototype. By default the number of clusters emerges from suppression. Set targetK to force exactly that many clusters. For mass cytometry, scale = "arcsinh" and affinityParams = list(alpha = "auto") are good starting points. Progress output is off unless verbose = TRUE is passed through ....

Value

`bHIVEParam()` returns a `bHIVEParam` object, a subclass of BlusterParam.

`clusterRows()` returns a factor of cluster labels, one per row of `x`. With `full = TRUE` it returns a list with `clusters` and `objects`, where `objects$bHIVE` is the `bHIVE()` result, including the fitted `AINet` model in `objects$bHIVE$model` for labeling new cells.

See Also

`bHIVE`, `clusterRows`, `clusterSweep`.

Examples

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

p <- bHIVEParam(targetK = 3, maxIter = 10)
p
set.seed(1)
clusters <- clusterRows(X, p)
table(clusters, iris$Species)

# Full output keeps the fitted model for labeling new cells
out <- clusterRows(X, p, full = TRUE)
out$objects$bHIVE$model$predict(X[1:5, ])

# Sweep over parameters
sweep <- clusterSweep(X, bHIVEParam(maxIter = 5), nAntibodies = c(10, 30))
sweep$parameters

# Cluster a SingleCellExperiment embedding and store the labels
library(SingleCellExperiment)
sce <- SingleCellExperiment(assays = list(logcounts = t(X)))
reducedDim(sce, "PCA") <- prcomp(X)$x[, 1:3]
colLabels(sce) <- clusterRows(reducedDim(sce, "PCA"), p)
table(colLabels(sce), iris$Species)
```

`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

Value

An `R6Class` generator. `ClassSwitcher$new()` returns a `ClassSwitcher` object.

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$get_alpha()`
- `ClassSwitcher$format()`
- `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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
ClassSwitcher$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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.

Value

An R6Class generator. ConvergentSelector$new() returns a ConvergentSelector object.

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$from_results()
- ConvergentSelector$format()
- 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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
ConvergentSelector$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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

Value

An R6Class generator. GerminalCenter$new() returns a GerminalCenter object.

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$format()
- GerminalCenter$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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
GerminalCenter$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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,
  smallClusterAction = c("merge", "keep", "drop"),
  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. Clusters with fewer members are handled per smallClusterAction. NULL (default) keeps every cluster regardless of size.

`smallClusterAction`
: One of `"merge"` (default), `"keep"`, or `"drop"`, controlling clusters below `minClusterSize`. `"merge"` reassigns their members to the nearest surviving prototype so no observation is lost (the rare cluster is absorbed, not deleted). `"keep"` retains the small cluster as its own prototype (use this to *preserve* rare populations). `"drop"` is the legacy behavior that discards the cluster and its members from subsequent layers. Ignored when `minClusterSize` is `NULL`.

`distance`
: Distance metric for medoid calculation, e.g. "euclidean".

`verbose`
: Logical, if TRUE reports progress at each layer with `message()`.

`refine`
: Logical, if TRUE apply gradient-based refinement via `refineB()` to each layer's antibodies, in the scaled space the layer was trained in. Refinement polishes the reported antibodies and the final layer's model. The collapse into the next layer's prototypes uses the bHIVE assignments.

`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) is the `bHIVE` result for that layer plus:

- `antibodies`: The antibodies learned in that layer.
- `assignments`: For each row of the layer's input (the original rows for layer 1, the previous layer's prototypes after that), the cluster ID or, for classification, the class label.
- `prototypes`: The collapsed prototypes that feed the next layer, one row per cluster. Row `k` summarizes the original observations with `membership == k`.
- `membership`: For each **original** row in `X`, the row of `prototypes` it belongs to (`NA` if its cluster was dropped).

- `predictions`: For classification, the predicted label for each original row: the majority label of its cluster.
- `model`: Final layer only. The fitted `AINet` model, for labeling new observations with `$predict()`.
- `task`: The specified task.

For classification, each layer collapses observations by the antibody that labeled them, so a layer keeps several prototypes per class.

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` *Multilayered honeycombHIVE classification model for caret*

Description

A custom model definition that lets `caret::train()` tune, resample and evaluate `honeycombHIVE` classifiers with gradient-based refinement.

Usage

```
honeycombHIVEmodel
```

Format

A list with the components **caret** expects of a custom model.

Details

Training runs `honeycombHIVE(refine = TRUE)` with the tuned parameters. Further arguments given to `caret::train()` are passed on to `honeycombHIVE()`. Progress output is off unless `verbose = TRUE` is passed. Prediction uses the fitted `AINet` model of the final layer. caret requires column names on `x`. Class probabilities are not implemented (`prob = NULL`).

Value

A list defining a custom **caret** model, for use as `caret::train(method = honeycombHIVEmodel)`.

Parameters

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

See Also

`honeycombHIVE`, `bHIVEmodel`, `train`

Examples

```
if (requireNamespace("caret", quietly = TRUE)) {
    X <- as.matrix(iris[, 1:4])
    y <- iris$Species
    set.seed(42)
    fit <- caret::train(
        x = X, y = y, method = honeycombHIVEmodel,
        trControl = caret::trainControl(method = "cv", number = 3),
        tuneGrid = data.frame(nAntibodies = 30, beta = 5, epsilon = 0.05,
                              layers = 2, refineOptimizer = "adam",
                              refineSteps = 3, refineLR = 0.01),
        maxIter = 10
    )
    fit
    table(Predicted = predict(fit, X), Actual = y)
}
```

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.

Value

An R6Class generator. `IdiotypicNetwork$new()` returns an `IdiotypicNetwork` object.

Experimental

The activation rule is applied to each antibody pair and the responses are summed: every neighbor whose affinity falls outside `[theta_low, theta_high]` counts as one unit of suppression. In a small, tight repertoire this culls crowded and isolated antibodies as intended (see the example). In a large or spread-out repertoire most pairs are far apart, their affinity is near zero, and every antibody is suppressed. `regulate()` then warns and leaves the repertoire unchanged, and AINet keeps the top 10% by population and reports it with a message. Treat the module as experimental and check that antibodies survive on your data. A field-based form, in which the activation acts on the total stimulation each antibody receives, is planned.

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$format()`
- `IdiotypicNetwork$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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
IdiotypicNetwork$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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

```
# Four antibodies with pairwise affinity 0.25 (inside the default
# activation window) stimulate each other and survive. A fifth, far from
# the rest, is under-stimulated and dies.
a <- sqrt(log(4) / 2)
A <- rbind(cbind(diag(4) * a, 0), c(0, 0, 0, 0, 10))
rep <- ImmuneRepertoire$new(A)
idi <- IdiotypicNetwork$new(theta_low = 0.01, theta_high = 0.5)
idi$regulate(rep, "gaussian", list(alpha = 1))
print(idi)
rep$size()
```

`ImmuneAlgorithm` *ImmuneAlgorithm*

Description

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

Value

An `R6Class` generator. `ImmuneAlgorithm$new()` returns an `ImmuneAlgorithm` object. It is an abstract base class: use `AINet` for a fitted model.

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$predict()
- ImmuneAlgorithm$format()
- 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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
ImmuneAlgorithm$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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.

Value

An `R6Class` generator. `ImmuneRepertoire$new()` returns an `ImmuneRepertoire` object.

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$format()
- ImmuneRepertoire$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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
ImmuneRepertoire$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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.

Value

An `R6Class` generator. `MemoryPool$new()` returns a `MemoryPool` object.

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$format()`
- `MemoryPool$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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
MemoryPool$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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

Value

An `R6Class` generator. `Microenvironment$new()` returns a `Microenvironment` object.

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$format()`
- `Microenvironment$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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
Microenvironment$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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	Vector (length = nSamples) giving the antibody each sample belongs to. Numeric values are row indices of A. Character or factor values are matched to rownames(A) when every value is a row name, and otherwise mapped to rows in order of first appearance.
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, reports progress with message() each step.
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$Species
res <- bHIVE(X, y, task = "classification", nAntibodies = 10,
             maxIter = 5, verbose = FALSE)
# antibody_index gives the antibody (row of res$antibodies) that labeled
# each sample
A_refined <- refineB(res$antibodies, X, y = y,
                     assignments = res$antibody_index,
                     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.

Value

An R6Class generator. SHMEngine$new() returns an SHMEngine object.

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$reset_state()`
- `SHMEngine$format()`
- `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.

Returns: The engine, invisibly, with its moment state set.

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.

Returns: The engine, invisibly, with its moment state reset or subset.

Method `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
SHMEngine$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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, one row per combination. Any column named after a bHIVE or AINet argument is tuned (for example nAntibodies, beta, epsilon, targetK, scale, affinityFunc or coverageBoost). Other columns are ignored. Factor columns, as produced by expand.grid(), are converted to character.
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). A maxIter column in grid takes precedence.
BPPARAM	Character. A BiocParallel::bpparam() object that can be used for parallelization. The function supports SerialParam, MulticoreParam, BatchtoolsParam, and SnowParam.
verbose	Logical. If TRUE, reports progress with message().

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

Value

An `R6Class` generator. `VDJLibrary$new()` returns a `VDJLibrary` object.

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$generate()`
- `VDJLibrary$format()`
- `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 `format()`**:** Format a short summary, one element per line. Used by `print()`.

Usage:

```
VDJLibrary$format(...)
```

Arguments:

`...` Not used.

Returns: A character vector.

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

Plots the data together with the prototypes learned by bHIVE or honeycombHIVE. One or more layers can be shown, faceted by layer:

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 = NULL,
  ...
)
```

Arguments

result	A list produced by bHIVE or honeycombHIVE. For multilayer results, each element represents one layer.
X	A numeric matrix or data frame of the original input features, with the observations in the same order as the input to bHIVE or honeycombHIVE. Required.
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 X has more than two columns, the dimensionality reduction in transformation_method is applied for scatterplots. Otherwise the first two columns are plotted.
transformation_method	Character. The method used for dimensionality reduction. Options are "PCA", "UMAP" (needs the **umap** package), "tSNE" (needs the **Rtsne** package), 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, "clustering" or "classification", or NULL (default) to use the task stored in result. Decides whether observations are grouped by cluster or by predicted class.
...	Additional arguments passed to the dimensionality reduction: stats::prcomp(), umap::umap() or Rtsne::Rtsne().

Details

- "scatter": The observations, colored by cluster or class, with the prototypes drawn on top in the same coordinates. When X has more than two columns and transform is TRUE, a dimensionality reduction is fitted to X and the prototypes are projected into it.
- "boxplot", "violin": A selected feature by group, with the prototypes' values overlaid.
- "density": Density of a selected feature by group, with the prototypes' values marked.

For scatterplots the reduction is "PCA", "UMAP", "tSNE" or "none". PCA and UMAP are fitted to X only and the prototypes are placed with the fitted model's predict() method, so both share one coordinate system. UMAP's projection of new points is an approximation. tSNE has no such projection, so the data and prototypes are embedded together and the prototypes take a small part in shaping the layout.

The prototypes drawn are, for a bHIVE() result, the antibodies in original feature units (antibodies_unscaled), and for a honeycombHIVE() layer, the collapsed prototypes that summarize each cluster.

Value

A ggplot object representing the visualization.

Examples

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

# A single bHIVE clustering, with the prototypes projected onto the PCA
set.seed(1)
res <- bHIVE(X, targetK = 3, maxIter = 10, verbose = FALSE)
visualizeHIVE(res, X = X, plot_type = "scatter", title = "bHIVE prototypes")

# Layers 1 and 2 of a honeycombHIVE hierarchy
res_h <- honeycombHIVE(X = X,
                       task = "clustering",
                       epsilon = 0.05,
                       layers = 3,
                       nAntibodies = 30,
                       beta = 5,
                       maxIter = 10,
                       verbose = FALSE)
```

```
visualizeHIVE(res_h, X = X, plot_type = "scatter", layer = 1:2,
              title = "honeycombHIVE layers")

# A feature by class for a classifier, with each antibody's value overlaid
res_c <- bHIVE(X, y = iris$Species, maxIter = 10, verbose = FALSE)
visualizeHIVE(res_c, X = X, plot_type = "violin",
              feature = "Petal.Length", title = "Classifier")
```

Index

∗ **datasets**
    bHIVEmodel, 12
    honeycombHIVEmodel, 24
.defaultScalarArguments,bHIVEParam-method *(*bHIVEParam*)*, 13

ActivationGate, 2
AINet, 4, *9–12*, *14, 15*, *24*, *26*, *28*, *43*

bHIVE, 8, *12–15*, *21*, *23*, *43*, *47*
bHIVE,ANY-method *(*bHIVE*)*, 8
bHIVE,SingleCellExperiment-method *(*bHIVE*)*, 8
bHIVE,SummarizedExperiment-method *(*bHIVE*)*, 8
bHIVE::ImmuneAlgorithm, *5*
bHIVEmodel, 12, *25*
bHIVEParam, *10, 11*, 13
bHIVEParam-class *(*bHIVEParam*)*, 13
BlusterParam, *15*

ClassSwitcher, 15
clusterRows, *13*, *15*
clusterRows,ANY,bHIVEParam-method *(*bHIVEParam*)*, 13
clusterSweep, *15*
ConvergentSelector, 17

GerminalCenter, 19

honeycombHIVE, 21, *24, 25*, *47*
honeycombHIVEmodel, *13*, 24

IdiotypicNetwork, 25
ImmuneAlgorithm, 28
ImmuneRepertoire, *16*, *20*, *27, 28*, 30, *34*, *37*

MemoryPool, 33
Microenvironment, 36

R6Class, *3*, *5*, *16*, *18*, *20*, *26*, *28*, *30*, *33*, *36*, *40*, *45*
refineB, *21*, 38

SHMEngine, 40
show,bHIVEParam-method *(*bHIVEParam*)*, 13
SingleCellExperiment, *10*
SummarizedExperiment, *10*
swarmbHIVE, 43

train, *13*, *25*

VDJLibrary, 45
visualizeHIVE, 47