Package: DICErClust (via r-universe)

June 2, 2026

Title Deep Significance Clustering for Clinical Risk Stratification

Version 0.1.2

Description We provide an R implementation of Deep Significance Clustering (DICE), a self-supervised learning framework designed to identify clinically meaningful and risk-stratified patient subgroups from electronic health record (EHR) data. DICE jointly optimizes deep representation learning, clustering, and outcome prediction while enforcing statistical significance between predicted outcomes and cluster membership. This integrated optimization produces subgroups that are both clinically coherent and predictive, addressing a gap where traditional unsupervised clustering methods and supervised risk prediction models alone may fail to generate actionable clinical groupings. See Huang et al. (2021) <doi:10.1093/jamia/ocab203>.

License MIT + file LICENSE

Encoding UTF-8

Imports argparser, ggplot2, torch, pROC

Suggests knitr, rmarkdown, scales

VignetteBuilder knitr

NeedsCompilation no

Author Sarah Ayton [aut, cre] (ORCID: <https://orcid.org/0000-0001-5247-9912>), Yiye Zhang [aut] (ORCID: <https://orcid.org/0000-0003-3494-2699>)

Maintainer Sarah Ayton <saa7050@med.cornell.edu>

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-05-28 12:50:09 UTC

RemoteUrl https://github.com/cran/DICErClust

RemoteRef HEAD

RemoteSha 5c66e4d5d622c2a2a9018a5fa2129653e66f5e1e

Contents

`analysis_p_value_related`

Analyze P-Value and Outcome Ratios for Clusters

Description

This function performs an analysis of p-values and outcome ratios related to clusters in a dataset. It calculates the ratio of positive outcomes within each cluster and computes p-values for the association between clusters and the outcome variable.

Usage

```
analysis_p_value_related(data_train, num_clusters, if_check = FALSE)
```

Arguments

`data_train`	A list representing the training data. This list must include the following components: • `C`: An integer vector of cluster assignments for each sample. • `data_v`: A matrix or data frame of additional covariates. • `data_y`: A binary response vector with values 0 or 1, representing the outcome variable.
`num_clusters`	An integer specifying the number of clusters in the data.
`if_check`	A logical value indicating whether to print intermediate steps for debugging purposes. Default is `FALSE`.

Details

The analysis_p_value_related function performs the following steps:

1. Converts the cluster assignments into a one-hot encoded matrix and counts the occurrences of each cluster.
2. Calculates the ratio of positive outcomes within each cluster.
3. Combines the cluster one-hot encodings with additional covariates to create a matrix of predictor variables.
4. Iteratively removes one or two clusters from the predictor matrix and calculates the p-value for the remaining clusters using the p_value_calculate function.

Value

A list containing the following components:

- dict_outcome_ratio: A numeric vector representing the ratio of positive outcomes (where data_y equals 1) within each cluster.
- dict_p_value: A list of p-values, with each element corresponding to a pair of clusters. The p-values are calculated using a likelihood ratio test for the association between the clusters and the outcome variable.

Examples

```
set.seed(1)
n <- 80L
data_train <- list(
  C      = sample(0:1, n, replace = TRUE),
  data_v = matrix(rbinom(n * 3L, 1, 0.5), n, 3L),
  data_y = rbinom(n, 1, 0.3)
)
result <- analysis_p_value_related(data_train, num_clusters = 2L)
result$dict_outcome_ratio   # outcome rate per cluster
result$dict_p_value          # LRT p-value for the cluster pair
```

---

analysis_p_value_related_onlyc

Analyze P-Value and Outcome Ratios for Clusters Using Only Cluster Information

---

Description

This function performs an analysis of p-values and outcome ratios related to clusters in a dataset, using only the cluster assignments and not additional covariates. It calculates the ratio of positive outcomes within each cluster and computes p-values for the association between clusters and the outcome variable.

Usage

```
analysis_p_value_related_onlyc(data_train, num_clusters, if_check = FALSE)
```

Arguments

`data_train`	A list representing the training data. This list must include the following components: • `C`: An integer vector of cluster assignments for each sample. • `data_v`: A matrix or data frame of additional covariates (not used in this function). • `data_y`: A binary response vector with values 0 or 1, representing the outcome variable.
`num_clusters`	An integer specifying the number of clusters in the data.
`if_check`	A logical value indicating whether to print intermediate steps for debugging purposes. Default is `FALSE`.

Details

The `analysis_p_value_related_onlyc` function performs the following steps:

1. Converts the cluster assignments into a one-hot encoded matrix and counts the occurrences of each cluster.
2. Calculates the ratio of positive outcomes within each cluster.
3. Iteratively removes one or two clusters from the predictor matrix and calculates the p-value for the remaining clusters using the `p_value_calculate` function.

Value

A list containing the following components:

- `dict_outcome_ratio`: A numeric vector representing the ratio of positive outcomes (where `data_y` equals 1) within each cluster.
- `dict_p_value`: A list of p-values, with each element corresponding to a pair of clusters. The p-values are calculated using a likelihood ratio test for the association between the clusters and the outcome variable.

Examples

```
set.seed(1)
n <- 80L
data_train <- list(
  C      = sample(0:1, n, replace = TRUE),
  data_v = matrix(rbinom(n * 3L, 1, 0.5), n, 3L),
  data_y = rbinom(n, 1, 0.3)
)
result <- analysis_p_value_related_onlyc(data_train, num_clusters = 2L)
result$dict_outcome_ratio
result$dict_p_value
```

`change_label_from_highratio_to_lowratio` *Reassign Cluster Labels Based on Outcome Ratios*

Description

This function reassigns cluster labels such that clusters with higher ratios of positive outcomes are assigned lower labels, and those with lower ratios are assigned higher labels. This relabeling can be useful for aligning cluster labels with the significance of outcomes.

Usage

```
change_label_from_highratio_to_lowratio(args, oldlabel, data_train)
```

Arguments

`args`	A list of arguments, which should include the following component: • `K_clusters`: An integer specifying the number of clusters.
`oldlabel`	An integer vector of the original cluster labels assigned to each sample.
`data_train`	A list representing the training data. This list must include the following components: • `data_v`: A matrix or data frame of additional covariates. • `data_y`: A binary response vector with values 0 or 1, representing the outcome variable.

Details

The `change_label_from_highratio_to_lowratio` function performs the following steps:

1. Calculates the count of samples and the count of positive outcomes (where `data_y` equals 1) for each cluster.
2. Computes the ratio of positive outcomes within each cluster.
3. Orders the clusters by their outcome ratios in descending order.
4. Reassigns cluster labels such that clusters with higher outcome ratios receive lower labels.
5. Returns the new cluster labels and the mapping from old labels to new labels.

Value

A list containing:

- `new_list_c`: A tensor of the new cluster labels, with clusters relabeled according to the outcome ratios.
- `order_c_map`: A named vector that maps the old cluster labels to the new ones, where the names represent the original labels and the values represent the new labels.

Examples

```
## Called internally by DICEr() after each k-means step.
## Requires a trained model object; see ?DICEr for a full usage example.
```

DecoderRNN *DecoderRNN Neural Network Module*

Description

A neural network module implementing a decoder using an LSTM architecture, based on the torch package. This module is typically used in sequence-to-sequence tasks during the decoding phase.

Usage

```
DecoderRNN(input_size, nhidden, nlayers, dropout)
```

Arguments

input_size	A numeric value representing the number of features in the input data.
nhidden	A numeric value representing the number of hidden units in each LSTM layer.
nlayers	A numeric value indicating the number of LSTM layers.
dropout	A numeric value between 0 and 1 indicating the dropout probability to be applied between LSTM layers.

Value

A tensor representing the output sequence, flipped along the sequence dimension.

Usage

```
DecoderRNN <- nn_module(
  "DecoderRNN",

  initialize = function(input_size, nhidden, nlayers, dropout) {
    ...
  },

  forward = function(x, h) {
    ...
  }
)
```

Methods

initialize(input_size, nhidden, nlayers, dropout) Initializes the DecoderRNN module.

\itemize{
\item{\code{input_size}: A numeric value representing the number of features in the input data.}
\item{\code{nhidden}: A numeric value representing the number of hidden units in each LSTM layer.}
\item{\code{nlayers}: A numeric value indicating the number of LSTM layers.}
\item{\code{dropout}: A numeric value between 0 and 1 indicating the dropout probability to be app
}

forward(x, h) Executes the forward pass of the DecoderRNN module.

\itemize{
\item{\code{x}: A tensor of shape \code{(batch_size, sequence_length, input_size)} representing t
\item{\code{h}: A list containing the hidden state (\code{hn}) and cell state (\code{cn}) from the
}

\value{
A tensor representing the output sequence, flipped along the sequence dimension.
}

Examples

```
dec <- DecoderRNN(input_size = 4L, nhidden = 8L, nlayers = 1L, dropout = 0.0)
enc <- EncoderRNN(input_size = 4L, nhidden = 8L, nlayers = 1L, dropout = 0.0)
x   <- torch::torch_randn(2L, 1L, 4L)
enc_out <- enc(x)
dec_out <- dec(enc_out[[3]], enc_out[[2]])
```

DICEr *DICEr Function for Training and Evaluating a Deep Learning Model*

Description

This function orchestrates the training and evaluation of a deep learning model, specifically for autoencoder-based representation learning followed by clustering and classification. It handles data loading, model training, testing, and saving the best model based on performance criteria.

Usage

DICEr(args)

Arguments

`args` A list of arguments that configure the model training and evaluation process. The list should include:

- `seed`: An integer value for setting the random seed for reproducibility.
- `input_path`: A character string specifying the path to the input data directory.
- `filename_train`: A character string specifying the filename of the training dataset.
- `filename_test`: A character string specifying the filename of the test dataset.
- `n_input_fea`: An integer specifying the number of input features.
- `n_hidden_fea`: An integer specifying the number of hidden features in the model.
- `lstm_layer`: An integer specifying the number of LSTM layers in the model.
- `lstm_dropout`: A numeric value specifying the dropout rate for the LSTM layers.
- `K_clusters`: An integer specifying the number of clusters for the k-means clustering.
- `n_dummy_demov_fea`: An integer specifying the number of dummy demographic features.
- `cuda`: A logical value indicating whether to use CUDA (GPU acceleration) for model training.
- `lr`: A numeric value specifying the learning rate for the optimizer.
- `init_AE_epoch`: An integer specifying the number of epochs for training the autoencoder.
- `iter`: An integer specifying the number of iterations for the main optimization process.
- `epoch_in_iter`: An integer specifying the number of epochs in each iteration of the main optimization process.
- `lambda_AE`: A numeric value specifying the weight of the autoencoder loss in the overall loss function.
- `lambda_classifier`: A numeric value specifying the weight of the classification loss in the overall loss function.
- `lambda_outcome`: A numeric value specifying the weight of the outcome prediction loss in the overall loss function.
- `lambda_p_value`: A numeric value specifying the weight of the p-value loss in the overall loss function.

Details

The `DICEr` function executes the following steps:

1. Sets the random seed for reproducibility.
2. Loads and preprocesses the training and test datasets.
3. Initializes the model, optimizer, and loss functions.

4. Trains an autoencoder for representation learning, saving the model and plotting loss curves.
5. Performs k-means clustering on the learned representations and reassigns cluster labels based on outcome ratios.
6. Iteratively trains the model for clustering, classification, and outcome prediction, optimizing the combined loss function.
7. Saves the best model based on the outcome prediction likelihood and checks for p-value significance.
8. Outputs and saves relevant training metrics, including loss curves and model checkpoints.

Value

Called for its side effects: trains the model and saves results to disk. Returns `NULL` invisibly.

Examples

```
## Prepare data in DICErClust format (length-3 RDS list)
data_dir <- tempdir()
n <- 100L; p <- 4L; q <- 2L
set.seed(1)
saveRDS(list(matrix(runif(n * p), n, p),                        # data_x
             matrix(as.double(rbinom(n * q, 1, 0.5)), n, q),  # data_v (float)
             rbinom(n, 1, 0.3)),                                # data_y
        file.path(data_dir, "train.rds"))
saveRDS(list(matrix(runif(30L * p), 30L, p),
             matrix(as.double(rbinom(30L * q, 1, 0.5)), 30L, q),
             rbinom(30L, 1, 0.3)),
        file.path(data_dir, "test.rds"))

args <- list(
  seed = 1L, input_path = paste0(data_dir, "/"),
  filename_train = "train.rds", filename_test = "test.rds",
  n_input_fea = p, n_hidden_fea = 2L,
  lstm_layer = 1L, lstm_dropout = 0.0, K_clusters = 2L,
  n_dummy_demov_fea = q, cuda = FALSE, lr = 1e-4,
  init_AE_epoch = 2L, iter = 5L, epoch_in_iter = 1L,
  lambda_AE = 1.0, lambda_classifier = 1.0,
  lambda_outcome = 1.0, lambda_p_value = 1.0
)
old_wd <- setwd(tempdir())
DICEr(args)
setwd(old_wd)
```

`EncoderRNN` *EncoderRNN Neural Network Module*

Description

An RNN-based encoder module using Long Short-Term Memory (LSTM) architecture, implemented with the `torch` package. This module is designed to process sequential data and is commonly used in tasks like sequence-to-sequence modeling, language modeling, and time series forecasting.

Usage

```
EncoderRNN(input_size, nhidden, nlayers, dropout)
```

Arguments

`input_size`	A numeric value representing the number of features in the input data.
`nhidden`	A numeric value representing the number of hidden units in each LSTM layer.
`nlayers`	A numeric value indicating the number of LSTM layers.
`dropout`	A numeric value between 0 and 1 indicating the dropout probability to be applied between LSTM layers.

Value

An `nn_module` object (torch LSTM-based encoder). When called with a tensor of shape (`batch_size`, `seq_len`, `input_size`), returns a list of: (1) processed output tensor, (2) list of hidden and cell states (`hn`, `cn`), and (3) the input with a leading zero tensor prepended.

Usage

```
EncoderRNN <- nn_module(
  "EncoderRNN",

  initialize = function(input_size, nhidden, nlayers, dropout) {
    ...
  },

  forward = function(x) {
    ...
  }
)
```

Methods

`initialize(input_size, nhidden, nlayers, dropout)` Initializes the EncoderRNN module.

```
    \itemize{
     \item{\code{input_size}: A numeric value representing the number of features in the input data.}
     \item{\code{nhidden}: A numeric value representing the number of hidden units in each LSTM layer.}
     \item{\code{nlayers}: A numeric value indicating the number of LSTM layers.}
     \item{\code{dropout}: A numeric value between 0 and 1 indicating the dropout probability to be app
    }
```

`forward(x)` Executes the forward pass of the EncoderRNN module.

```
\itemize{
 \item{\code{x}: A tensor of shape (batch_size, sequence_length, input_size) representing the inpu
}

\value{
A list containing:
  \itemize{
   \item{\code{output}: The output tensor from the LSTM, processed and flipped.}
   \item{\code{list(hn, cn)}: A list containing the hidden state (\code{hn}) and cell state (\code{
   \item{\code{newinput}: A tensor representing the input data with an additional zero tensor at the
  }
}
```

Examples

```
enc <- EncoderRNN(input_size = 4L, nhidden = 8L, nlayers = 1L, dropout = 0.0)
x   <- torch::torch_randn(2L, 1L, 4L)   # batch=2, seq_len=1, features=4
out <- enc(x)
dim(out[[1]])  # encoded output
```

`func_analysis_test_error_D0406`

Analyze Test Error and Performance Metrics for a Model

Description

This function evaluates a trained model on a test dataset, calculating various performance metrics including autoencoder loss, outcome likelihood, classification accuracy, and AUC score for outcome prediction.

Usage

```
func_analysis_test_error_D0406(args, model, data_test, dataloader_test)
```

Arguments

`args`	A list of arguments, typically containing configurations such as whether to use CUDA for computation.
`model`	A trained model that will be evaluated. The model should have a `forward` method that supports an "outcome_logistic_regression" mode.
`data_test`	A list representing the test dataset. This list will be updated with predicted cluster assignments.
`dataloader_test`	An object representing the test data loader, which provides batches of test data to the model.

Details

The `func_analysis_test_error_D0406` function performs the following steps:

1. Sets the model to evaluation mode to ensure proper handling of layers like dropout.
2. Iterates over the test data batches provided by `dataloader_test`.
3. For each batch, it performs a forward pass through the model using the "outcome_logistic_regression" mode, obtaining the encoded representations, decoded outputs, raw cluster predictions, and outcome predictions.
4. Computes the mean squared error (MSE) for the autoencoder loss, binary cross-entropy (BCE) for the outcome prediction loss, and classification accuracy for cluster prediction.
5. Accumulates the true outcomes and predicted probabilities for calculating the AUC score.
6. Updates the `data_test` list with the predicted cluster assignments.
7. Returns the average values of the computed metrics across all test batches.

Value

A list containing the following performance metrics:

- `test_AE_loss`: The mean autoencoder loss (MSE) across all batches in the test dataset.
- `test_classifier_c_accuracy`: The classification accuracy of the model in predicting cluster assignments.
- `test_outcome_likelihood`: The mean binary cross-entropy loss for the outcome prediction.
- `outcome_auc_score`: The AUC (Area Under the Curve) score for the outcome prediction, which measures the model's ability to distinguish between positive and negative classes.

Examples

```
## Called internally by DICEr() to evaluate the held-out test set.
## Requires a trained DICEr model object; see ?DICEr for a full example.
```

---

main *Main Function for Training and Evaluating a Deep Learning Model*

Description

This function orchestrates the training and evaluation of a deep learning model, specifically for autoencoder-based representation learning followed by clustering and classification. It handles data loading, model training, testing, and saving the best model based on performance criteria.

Usage

```
main(args)
```

Arguments

`args`	A list of arguments that configure the model training and evaluation process. The list should include:

- `seed`: An integer value for setting the random seed for reproducibility.
- `input_path`: A character string specifying the path to the input data directory.
- `filename_train`: A character string specifying the filename of the training dataset.
- `filename_test`: A character string specifying the filename of the test dataset.
- `n_input_fea`: An integer specifying the number of input features.
- `n_hidden_fea`: An integer specifying the number of hidden features in the model.
- `lstm_layer`: An integer specifying the number of LSTM layers in the model.
- `lstm_dropout`: A numeric value specifying the dropout rate for the LSTM layers.
- `K_clusters`: An integer specifying the number of clusters for the k-means clustering.
- `n_dummy_demov_fea`: An integer specifying the number of dummy demographic features.
- `cuda`: A logical value indicating whether to use CUDA (GPU acceleration) for model training.
- `lr`: A numeric value specifying the learning rate for the optimizer.
- `init_AE_epoch`: An integer specifying the number of epochs for training the autoencoder.
- `iter`: An integer specifying the number of iterations for the main optimization process.
- `epoch_in_iter`: An integer specifying the number of epochs in each iteration of the main optimization process.
- `lambda_AE`: A numeric value specifying the weight of the autoencoder loss in the overall loss function.
- `lambda_classifier`: A numeric value specifying the weight of the classification loss in the overall loss function.
- `lambda_outcome`: A numeric value specifying the weight of the outcome prediction loss in the overall loss function.
- `lambda_p_value`: A numeric value specifying the weight of the p-value loss in the overall loss function.

Details

The `main` function executes the following steps:

1. Sets the random seed for reproducibility.
2. Loads and preprocesses the training and test datasets.
3. Initializes the model, optimizer, and loss functions.

4. Trains an autoencoder for representation learning, saving the model and plotting loss curves.
5. Performs k-means clustering on the learned representations and reassigns cluster labels based on outcome ratios.
6. Iteratively trains the model for clustering, classification, and outcome prediction, optimizing the combined loss function.
7. Saves the best model based on the outcome prediction likelihood and checks for p-value significance.
8. Outputs and saves relevant training metrics, including loss curves and model checkpoints.

Value

Called for its side effects: trains the model and saves results to disk. Returns `NULL` invisibly.

Examples

```
## See ?DICEr for a complete usage example.
## main() is an alias for DICEr() kept for backwards compatibility.
```

`model_2` *model_2 Neural Network Module*

Description

A neural network module implementing a multi-functional model with encoder-decoder architecture using the `torch` package. The module is constructed using `nn_module()` and configured via its `initialize()` method.

Usage

```
model_2(input_size, nhidden, nlayers, dropout, n_clusters, n_dummy_demov_fea, para_cuda)
```

Arguments

`input_size`	Integer. Number of features in the input data.
`nhidden`	Integer. Number of hidden units in each LSTM layer.
`nlayers`	Integer. Number of LSTM layers.
`dropout`	Numeric. Dropout probability between LSTM layers (0–1).
`n_clusters`	Integer. Number of clusters/classes.
`n_dummy_demov_fea`	Integer. Number of demographic dummy features.
`para_cuda`	Logical. Whether to use CUDA (GPU acceleration).
	The `forward()` method arguments: `"get_representation"`, `"classifier"`, `"outcome_logistic_regression"`.

Details

After initialization, the module can be used via `model$forward(x, function_name, demov = NULL, mask_BoolTensor = NULL)`.

Arguments:

- `x`: Input tensor.
- `function_name`: One of "autoencoder", "get_representation", "classifier", "outcome_logistic_regression".
- `demov`: Optional demographic tensor.
- `mask_BoolTensor`: Optional boolean mask tensor.

Value

An `nn_module` object.

Examples

```
if (requireNamespace("torch", quietly = TRUE)) {
  mod <- model_2(
    input_size = 10L, nhidden = 64L, nlayers = 2L, dropout = 0.1,
    n_clusters = 3L, n_dummy_demov_fea = 2L, para_cuda = FALSE
  )
}
```

---

p_value_calculate *Calculate P-Value for Logistic Regression Model*

Description

This function calculates the p-value for a logistic regression model by comparing the likelihoods of a full model and a null model.

Usage

```
p_value_calculate(X, y, is_intercept, X_null = NULL)
```

Arguments

`X`	A matrix of predictor variables for the full model.
`y`	A binary response vector, with values of 0 or 1.
`is_intercept`	A logical value indicating whether the null model should include only an intercept (i.e., no predictors). If `TRUE`, the null model includes only the intercept; if `FALSE`, the null model is specified by `X_null`.
`X_null`	(Optional) A matrix of predictor variables for the null model. This argument is required if `is_intercept` is `FALSE`.

Details

The p_value_calculate function performs the following steps:

1. Fits a full logistic regression model using X as the predictor matrix.
2. Depending on the value of is_intercept, it either:
   - Calculates the null model's log-likelihood using only an intercept (if is_intercept = TRUE).
   - Fits a null logistic regression model using X_null as the predictor matrix (if is_intercept = FALSE).
3. Computes the likelihood ratio statistic G, which is twice the difference in log-likelihoods between the full and null models.
4. Calculates the p-value using the chi-squared distribution with degrees of freedom equal to the difference in the number of parameters between the full and null models.

Value

A numeric value representing the p-value from the likelihood ratio test between the full and null models.

Examples

```
# Example with only an intercept in the null model
X <- matrix(rnorm(100 * 5), ncol = 5)
y <- rbinom(100, 1, 0.5)
p_value <- p_value_calculate(X, y, is_intercept = TRUE)
print(p_value)

# Example with a specified null model
X_null <- X[, 1:2]  # Using only the first two predictors in the null model
p_value <- p_value_calculate(X, y, is_intercept = FALSE, X_null = X_null)
print(p_value)
```

---

parse_args *Parse Command-Line Arguments for PPD-Aware Clustering*

Description

This function parses command-line arguments for a script that performs PPD-aware clustering, setting up various parameters necessary for training and evaluating a model.

Usage

```
parse_args()
```

Details

The `parse_args` function uses the `argparser` package to define and parse a series of command-line arguments. These arguments are essential for configuring various aspects of the PPD-aware clustering process, including model architecture, training settings, and input/output paths.

The following command-line arguments are supported:

- `--init_AE_epoch`: Integer. Number of epochs for representation initialization.
- `--n_hidden_fea`: Integer. Number of hidden units in the LSTM layers.
- `--output_path`: Character. Location where output results will be saved.
- `--input_path`: Character. Location of the input dataset.
- `--filename_train`: Character. Path to the training data file.
- `--filename_test`: Character. Path to the test data file.
- `--n_input_fea`: Integer. Number of original input features.
- `--n_dummy_demov_fea`: Integer. Number of dummy demographic features.
- `--lstm_layer`: Integer. Number of layers in the LSTM. Default is 1.
- `--lr`: Numeric. Learning rate for training. Default is 1e-4.
- `--lstm_dropout`: Numeric. Dropout rate for the LSTM layers. Default is 0.0.
- `--K_clusters`: Integer. Number of initial clusters.
- `--iter`: Integer. Maximum number of iterations for merging clusters. Default is 20.
- `--epoch_in_iter`: Integer. Number of epochs per iteration when merging clusters. Default is 1.
- `--seed`: Integer. Random seed for reproducibility. Default is 1111.
- `--cuda`: Integer. Flag indicating whether to use CUDA (GPU acceleration). Default is 1.
- `--lambda_AE`: Numeric. Regularization weight for the autoencoder loss in each iteration. Default is 1.0.
- `--lambda_classifier`: Numeric. Regularization weight for the classifier loss in each iteration. Default is 1.0.
- `--lambda_outcome`: Numeric. Regularization weight for the outcome loss in each iteration. Default is 1.0.
- `--lambda_p_value`: Numeric. Regularization weight for the p-value adjustment in each iteration. Default is 1.0.

Value

A list of parsed command-line arguments, each corresponding to a specific parameter required by the clustering model. This list is typically used to configure the model training and evaluation process.

Examples

```
## parse_args() reads from the command line; use the args list interface
## directly when calling DICEr() from within R (see ?DICEr).
args <- parse_args()
```

test_AE *Evaluate Autoencoder Model on Test Data*

Description

This function evaluates the performance of an autoencoder model on a test dataset, calculating the mean squared error (MSE) between the original inputs and the reconstructed outputs.

Usage

```
test_AE(args, model, dataloader_test)
```

Arguments

`args`	A list of arguments, typically containing configurations such as whether to use CUDA for computation.
`model`	A trained autoencoder model that will be evaluated. The model is expected to have a `forward` method that takes input data and a mode (e.g., "autoencoder").
`dataloader_test`	A list representing the test data loader. Each element in the list should be a batch containing test data that the model will evaluate.

Details

The `test_AE` function performs the following steps:

1. Initializes the mean squared error (MSE) loss criterion.
2. Sets the model to evaluation mode to ensure layers like dropout are appropriately handled.
3. Iterates over the test data batches provided by `dataloader_test`.
4. For each batch, the function:
   - Extracts the data, ensuring it is in numeric format and reshapes it as necessary.
   - Converts the data into torch tensors, moving them to CUDA if specified.
   - Performs a forward pass through the model in autoencoder mode, obtaining both the encoded representation and the reconstructed output.
   - Computes the MSE loss between the input and the reconstructed output.
   - Accumulates the loss for each batch.
5. Computes the mean of the accumulated losses to return the overall test error.

Value

A numeric value representing the mean test error (MSE) across all batches in the test dataset.

Examples

```
## Called internally by DICEr() after each autoencoder warm-up epoch.
## Requires a trained EncoderRNN/DecoderRNN model; see ?DICEr.
```

update_M *Update Cluster Embeddings in Matrix M*

Description

This function updates the matrix M, which contains cluster embeddings, based on the current cluster assignments and representations in the training data.

Usage

```
update_M(data_train)
```

Arguments

data_train	A list containing the training data. The list must include the following components: • M: A matrix where each column corresponds to a cluster's embedding vector. • n_cat: An integer representing the number of clusters (or categories). • rep: A matrix of representations (embeddings) for the training data. • C: An integer vector indicating the cluster assignment for each training sample.

Details

The update_M function works as follows:

1. It first checks that the number of columns in the M matrix matches the number of clusters (n_cat). If not, an error is thrown.
2. It then initializes a list of lists, dict_c_embedding, where each sublist will hold the embeddings for samples belonging to a specific cluster.
3. The function iterates over all samples, adding their representations to the corresponding cluster sublist in dict_c_embedding.
4. For each cluster, the function computes the mean of the embeddings across all samples assigned to that cluster.
5. The computed mean embedding is then used to update the corresponding column in the M matrix.

Value

The function updates the M matrix in-place within the data_train list, reflecting the new average embeddings for each cluster.

Examples

```
## Called internally by DICEr() to refresh cluster centroids each iteration.
## Requires a data_train list with rep, C, n_cat, and M fields; see ?DICEr.
```

update_testset_R_C_M_K *Update Test Set Representations, Cluster Assignments, and Cluster Centers*

Description

This function updates the representations, cluster assignments, and cluster centers for a test dataset based on a trained model and a reference training dataset.

Usage

```
update_testset_R_C_M_K(args, model, data_test, dataloader_test, data_train)
```

Arguments

args	A list of arguments, typically containing configurations such as the number of hidden features (n_hidden_fea) and whether to use CUDA (cuda).
model	A trained model, typically an autoencoder, that will be used to generate new representations for the test data.
data_test	A list representing the test dataset. This list will be updated with new representations, cluster assignments, and cluster centers.
dataloader_test	An object representing the test data loader, which provides batches of test data to the model.
data_train	A list representing the training dataset, which contains reference cluster centers and other attributes needed for updating the test dataset.

Details

The update_testset_R_C_M_K function performs the following steps:

1. Initializes a tensor, final_embed, to store the embeddings for the test dataset.
2. Sets the model to evaluation mode.
3. Iterates through the test data batches using dataloader_test, processing each batch through the model to obtain the embeddings.
4. Updates the rep attribute of data_test with the new embeddings.
5. Updates data_test$n_cat and data_test$M to match the training dataset.
6. For each test sample, computes the closest cluster center (based on Euclidean distance) and assigns the corresponding cluster to pred_C.

Value

The function updates the following attributes within the `data_test` list:

- `rep`: The new representations (embeddings) for the test dataset.
- `n_cat`: The number of clusters, set to match the training dataset.
- `M`: The cluster centers, copied from the training dataset.
- `pred_C`: The predicted cluster assignments for each test sample, based on the updated representations and cluster centers.

Examples

```
## Called internally by DICEr() to assign test-set cluster labels.
## Requires a trained DICEr model and data objects; see ?DICEr.
```

Index