

Package: PhysioML (via r-universe)

October 3, 2026

Title Leakage-Aware Machine Learning for Physiological Time Series

Version 0.4.2

Description Provides leakage-aware, reproducible feature transforms and reference deep-learning workflows for physiological time series stored in PhysioExperiment objects. Canonical catch22 characteristics are computed through the official R backend, while ROCKET and MiniRocket transforms use guarded optional Python backends. Guarded optional torch and luz backends provide case-aware datasets, compact neural architectures, training, and prediction. Explicit contracts preserve case, channel, feature, normalization, backend, and model provenance without claiming clinical validation. Named transfer-learning policies, leakage-safe CORAL alignment, and immutable hash-verified foundation-model caches provide bounded reuse across domains.

Depends R (>= 4.2), PhysioExperiment (>= 2.0.0)

Imports digest, jsonlite, methods, S4Vectors, SummarizedExperiment, stats

Suggests PhysioCore, Rcatch22, reticulate, torch, luz, testthat (>= 3.2.0), knitr, rmarkdown

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/testthat/edition 3

Collate 'PhysioML-package.R' 'feature-input.R' 'features-catch22.R' 'features-rocket.R' 'reduced-features.R' 'torch-input.R' 'torch-dataset.R' 'torch-models.R' 'torch-train.R' 'torch-predict.R' 'onnx-contract.R' 'onnx-export.R' 'onnx-import.R' 'onnx-predict.R' 'transfer.R' 'domain-adapt.R' 'foundation-cache.R' 'foundation.R' 'zzz.R'

URL <https://github.com/x-biosignal/PhysioML>,
<https://x-biosignal.r-universe.dev/PhysioML>,
<https://x-biosignal.github.io/PhysioML/>

BugReports <https://github.com/x-biosignal/PhysioML/issues>

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

Config/pak/sysreqs zlib1g-dev

Repository <https://x-biosignal.r-universe.dev>

Date/Publication 2026-10-03 13:11:43 UTC

RemoteUrl <https://github.com/x-biosignal/PhysioML>

RemoteRef HEAD

RemoteSha 9d10f45581685a9592b0846fe7d16ff93741ad30

Contents

cacheFoundationModel	3
catch22	4
domainAdapt	5
exportONNX	6
fineTune	7
foundationCacheInfo	9
foundationEmbed	9
freezeLayers	11
importONNX	12
minirocket	13
onnxPredict	14
peDataset	15
peModel	16
peReducedFeatures	17
predictModel	18
rocket	19
trainModel	21
validateSubjectSplit	22
withCaseData	23

cacheFoundationModel *Cache an immutable physiological foundation model*

Description

A first network download requires explicit consent, a fixed 40-character revision, a predeclared canonical manifest hash, and license acceptance. Cache hits revalidate the complete inventory and perform no download.

Usage

```
cacheFoundationModel(  
  model = c("moment", "chronos2", "labram", "bendr"),  
  model_id = NULL,  
  revision,  
  manifest_sha256,  
  cache_dir = tools::R_user_dir("PhysioML", "cache"),  
  allow_download = FALSE,  
  offline = FALSE,  
  license_accepted = FALSE,  
  overwrite = FALSE  
)
```

Arguments

model	Exact adapter name.
model_id	Optional exact upstream model identifier.
revision	Immutable 40-character commit hash.
manifest_sha256	Expected canonical manifest SHA-256.
cache_dir	Caller-owned cache root.
allow_download	Explicit first-download consent.
offline	Whether all network access is forbidden.
license_accepted	Whether the upstream license was accepted.
overwrite	Whether an existing valid entry may be replaced.

Value

Invisibly, path-free cache provenance.

Examples

```
# A first download needs explicit consent, the exact pinned revision, and the
# expected canonical manifest hash; it also needs a caller-managed Python env.
cacheFoundationModel(
  model = "moment",
  revision = "411e288267f82cce86296dbe4d6c8bc533cc162f",
  manifest_sha256 = strrep("0", 64), # replace with the real manifest hash
  cache_dir = tempfile("physioml-cache"),
  allow_download = TRUE,
  license_accepted = TRUE
)
```

catch22

Calculate canonical catch22 features

Description

Calls the official C-backed `Rcatch22::catch22_all()` implementation for every selected case-channel series. Input samples are passed unchanged.

Usage

```
catch22(x, assay_name = NULL, channels = NULL, cases = NULL, catch24 = FALSE)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> .
<code>assay_name</code>	NULL for <code>PhysioExperiment::defaultAssay()</code> or an exact assay name.
<code>channels</code>	NULL or exact unique channel labels in requested order.
<code>cases</code>	NULL, exact unique case labels, or exact unique positive case indices in requested order.
<code>catch24</code>	One non-missing logical. When TRUE, append upstream mean and standard-deviation features after the canonical 22.

Details

A 2-D assay is interpreted as time x channel for one case. A 3-D assay is interpreted as time x channel x case. Series must be finite, equal length within an assay, and contain at least 10 samples. `PhysioML` does not impute, resample, normalize, detrend, or collapse channels before calling `Rcatch22`.

Value

An `S4Vectors::DataFrame()` with one row per case-channel pair, identity columns followed by 22 or 24 numeric feature columns. Stable transform provenance and typed diagnostics are stored as attributes.

References

Lubba CH, Sethi SS, Knaute P, Schultz SR, Fulcher BD, Jones NS (2019). catch22: CAnonical Time-series CHaracteristics. *Data Mining and Knowledge Discovery*, 33, 1821-1852. doi:10.1007/s1061801900647x

Henderson T (2026). Rcatch22: Calculation of 22 Canonical Time-Series Characteristics. R package.

Examples

```
if (requireNamespace("Rcatch22", quietly = TRUE)) {
  values <- matrix(sin(seq(0, 8 * pi, length.out = 100)), 100, 1)
  pe <- PhysioExperiment::PhysioExperiment(
    assays = list(raw = values),
    coldData = S4Vectors::DataFrame(label = "signal"),
    samplingRate = 100
  )
  catch22(pe)
}
```

domainAdapt

Align source embeddings to an unlabeled target domain with CORAL

Description

Fit CORAL only with training-source observations and an explicitly designated unlabeled target adaptation set. Held-out target validation and test observations must not be supplied. CORAL matches first- and second-order statistics; it does not establish causal, class-conditional, fairness, or clinical invariance.

Usage

```
domainAdapt(
  source,
  target,
  method = c("coral"),
  regularization = 1e-06,
  return_model = TRUE
)
```

Arguments

source, target	Finite observation-by-feature matrices, or foundation embedding results, with identical exact feature names and order.
method	Currently exactly "coral".
regularization	Positive covariance regularization.
return_model	Whether to return the fitted transform and diagnostics.

Value

A matrix, or a plain list containing aligned source data and model.

Examples

```
set.seed(1)
feat <- c("f1", "f2", "f3")
source <- matrix(stats::rnorm(30), ncol = 3, dimnames = list(NULL, feat))
target <- matrix(stats::rnorm(30, mean = 1), ncol = 3,
                 dimnames = list(NULL, feat))
fit <- domainAdapt(source, target)
fit$diagnostics$source_mean_residual
```

exportONNX

Export a fitted PhysioML torch model to ONNX

Description

Uses the governed R torch to TorchScript to Python PyTorch bridge. Only the batch axis may be dynamic. The graph and canonical model card are validated before they are atomically landed.

Usage

```
exportONNX(
  model,
  path,
  opset = 17L,
  dynamic_batch = TRUE,
  validate = TRUE,
  metadata_path = paste0(path, ".json"),
  overwrite = FALSE
)
```

Arguments

model	A fitted <code>physio_dl_model</code> .
path	Destination <code>.onnx</code> path.
opset	Positive ONNX opset; the initial default is 17.
dynamic_batch	Whether the batch axis is dynamic.
validate	Whether to enforce the full ONNX checker, graph/card consistency, and logit parity tolerance. Basic graph parsing, I/O contract checks, and CPU inference are always required.
metadata_path	Destination canonical JSON model-card path.
overwrite	Whether an existing graph/card pair may be replaced.

Value

Invisibly, a plain `physio_onnx_export` list.

Examples

```
arr <- array(stats::rnorm(50 * 2 * 6), dim = c(50, 2, 6))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)
path <- tempfile(fileext = ".onnx")

# Export needs torch plus the governed torch->ONNX Python bridge.
train <- peDataset(
  pe, targets = rep(c("a", "b"), 3),
  task = "classification", class_levels = c("a", "b")
)
fit <- trainModel(train, model = "cnn1d", epochs = 1L)
exportONNX(fit, path)
```

fineTune

Fine-tune a fitted torch model

Description

Training is CPU-only and passes only explicitly trainable parameters to Adam. Frozen modules, including their batch-normalization buffers, remain in evaluation mode. Validation and prediction data must reuse frozen normalization statistics.

Usage

```
fineTune(
  model,
  train,
  valid = NULL,
  strategy = c("linear_probe", "partial", "full"),
  unfreeze = NULL,
  epochs = 10L,
  batch_size = 32L,
  learning_rate = 1e-04,
  weight_decay = 0,
  callbacks = NULL,
  seed = 1L,
  device = "cpu",
  num_workers = 0L,
  verbose = FALSE
)
```

Arguments

<code>model</code>	A fitted <code>physio_dl_model</code> or <code>physio_transfer_model</code> .
<code>train, valid</code>	Compatible targeted datasets.
<code>strategy</code>	Exact tuning strategy.
<code>unfreeze</code>	Component prefixes used only by partial tuning.
<code>epochs, batch_size</code>	Positive whole-number settings.
<code>learning_rate</code>	Positive Adam learning rate.
<code>weight_decay</code>	Non-negative Adam weight decay.
<code>callbacks</code>	Currently NULL; callback execution is owned by <code>trainModel()</code> 's luz workflow.
<code>seed</code>	Whole-number CPU seed.
<code>device</code>	Currently exactly "cpu".
<code>num_workers</code>	Currently exactly zero.
<code>verbose</code>	Whether to print one line per epoch.

Value

A fitted `physio_dl_model` with a plain transfer record.

Examples

```
arr <- array(stats::rnorm(50 * 2 * 6), dim = c(50, 2, 6))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)
# Needs the optional torch and luz backends.
if (requireNamespace("torch", quietly = TRUE) &&
    requireNamespace("luz", quietly = TRUE)) {
  train <- peDataset(
    pe, targets = rep(c("a", "b"), 3),
    task = "classification", class_levels = c("a", "b")
  )
  fit <- trainModel(train, model = "cnn1d", epochs = 1L)
  tuned <- fineTune(fit, train, strategy = "linear_probe", epochs = 1L)
}
```

foundationCacheInfo	<i>Inspect immutable foundation-model cache entries</i>
---------------------	---

Description

Inspect immutable foundation-model cache entries

Usage

```
foundationCacheInfo(
  model = NULL,
  cache_dir = tools::R_user_dir("PhysioML", "cache"),
  verify = TRUE
)
```

Arguments

model	Optional exact adapter filter.
cache_dir	Caller-owned cache root.
verify	Whether to revalidate every manifest and asset.

Value

A path-free data frame, one row per READY entry.

Examples

```
# An empty cache root returns a zero-row inventory, offline.
foundationCacheInfo(cache_dir = tempfile("physioml-cache"))
```

foundationEmbed	<i>Obtain governed foundation-model embeddings</i>
-----------------	--

Description

The exact model ID, immutable revision, complete manifest hash, ordered channels, sampling rate, units, and window contract are recorded. Python and all provider packages remain caller-managed. A first upstream download is explicit; package code never installs software or selects an environment.

Usage

```

foundationEmbed(
  x,
  model = c("moment", "chronos2", "labram", "bendr"),
  model_id = NULL,
  revision,
  manifest_sha256,
  assay_name = NULL,
  cases = NULL,
  channels = NULL,
  window_samples = NULL,
  stride_samples = NULL,
  reduction = c("mean", "none"),
  batch_size = 16L,
  device = "cpu",
  cache_dir = tools::R_user_dir("PhysioML", "cache"),
  allow_download = FALSE,
  offline = FALSE
)

```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> .
<code>model</code>	Exact adapter name.
<code>model_id</code>	Optional exact upstream model identifier.
<code>revision</code>	Immutable 40-character commit hash.
<code>manifest_sha256</code>	Expected canonical cache-manifest SHA-256.
<code>assay_name, cases, channels</code>	Exact input selections.
<code>window_samples, stride_samples</code>	Complete-window settings.
<code>reduction</code>	Exact documented embedding reduction.
<code>batch_size</code>	Positive whole-number batch size.
<code>device</code>	Currently exactly "cpu".
<code>cache_dir</code>	Caller-owned cache root.
<code>allow_download</code>	Explicit first-download consent.
<code>offline</code>	Whether all network access is forbidden.

Details

Stable dimensions are promised only for one exact model, revision, and preprocessing contract. The output is not a clinical prediction and does not establish fairness, causal invariance, or domain invariance.

Value

A path-free, serializable physio_foundation_embedding.

Examples

```
arr <- array(stats::rnorm(400 * 2 * 2), dim = c(400, 2, 2))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4"), unit = "uV"),
  samplingRate = 200
)

# Needs a pinned revision, the expected manifest hash, a caller-managed
# Python env, and (on first use) explicit download consent.
emb <- foundationEmbed(
  pe,
  model = "moment",
  revision = "411e288267f82cce86296dbe4d6c8bc533cc162f",
  manifest_sha256 = strrep("0", 64), # replace with the real manifest hash
  allow_download = TRUE
)
```

freezeLayers

Freeze named parts of a fitted torch model

Description

The fitted model is cloned before requires_grad is changed. Exact matching is the default; prefix matching is component-aware.

Usage

```
freezeLayers(
  model,
  trainable = NULL,
  frozen = NULL,
  match = c("exact", "prefix"),
  strict = TRUE
)
```

Arguments

model	A fitted physio_dl_model.
trainable, frozen	Optional unique exact parameter names or prefixes.
match	Exact selection mode.
strict	Whether unmatched rules are errors.

Value

A plain `physio_transfer_model`.

Examples

```
arr <- array(stats::rnorm(50 * 2 * 6), dim = c(50, 2, 6))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)
# Needs the optional torch and luz backends.
if (requireNamespace("torch", quietly = TRUE) &&
    requireNamespace("luz", quietly = TRUE)) {
  train <- peDataset(
    pe, targets = rep(c("a", "b"), 3),
    task = "classification", class_levels = c("a", "b")
  )
  fit <- trainModel(train, model = "cnn1d", epochs = 1L)
  frozen <- freezeLayers(fit)
  frozen$freeze_manifest
}
```

importONNX

Import a governed PhysioML ONNX graph

Description

Validates a graph against its canonical, hash-bound PhysioML model card and returns raw persistent bytes rather than a live Python runtime session.

Usage

```
importONNX(
  path,
  metadata_path = paste0(path, ".json"),
  providers = "CpuExecutionProvider",
  verify = TRUE
)
```

Arguments

<code>path</code>	Path to an ONNX graph.
<code>metadata_path</code>	Path to its canonical JSON model card.
<code>providers</code>	Execution provider; currently CPU only.
<code>verify</code>	Whether to run the ONNX checker and graph/card introspection.

Value

A persistent `physio_onnx_model`.

Examples

```
# Reads a governed graph plus its canonical JSON model card (ONNX Runtime).
path <- tempfile(fileext = ".onnx")
# `path` and paste0(path, ".json") are produced by exportONNX().
model <- importONNX(path)
```

minirocket	<i>Fit or apply the MiniRocket transform</i>
------------	--

Description

Uses `aeon.transformations.collection.convolution_based.MiniRocket`. A fitted model records the realized feature width because `aeon` rounds the requested kernel count according to the MiniRocket kernel family.

Usage

```
minirocket(
  x,
  model = NULL,
  assay_name = NULL,
  channels = NULL,
  cases = NULL,
  n_kernels = 10000L,
  seed = 1L,
  deterministic = TRUE,
  backend = "aeon"
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> .
<code>model</code>	NULL or a fitted <code>physio_rocket_model</code> .
<code>assay_name</code> , <code>channels</code> , <code>cases</code>	Input selection; see catch22() .
<code>n_kernels</code>	One finite whole number of kernels.
<code>seed</code>	One whole-number backend seed in $[0, 2^{31} - 1]$.
<code>deterministic</code>	One non-missing logical. TRUE requires the fixed-seed, single-worker <code>aeon</code> path and records that contract. Determinism is scoped to the recorded backend versions and platform.
<code>backend</code>	Exactly "aeon".

Value

A `physio_rocket_transform`; see `rocket()`.

References

Dempster A, Schmidt DF, Webb GI (2021). MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification. doi:[10.1145/3447548.3467231](https://doi.org/10.1145/3447548.3467231)

Examples

```
arr <- array(stats::rnorm(60 * 2 * 4), dim = c(60, 2, 4))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)

# The transform needs a caller-managed Python environment with NumPy and aeon.
fit <- minirocket(pe, n_kernels = 84)
dim(fit$features)
```

onnxPredict

Predict through ONNX Runtime on CPU

Description

Materializes a validated raw ONNX payload only for the current call, disables provider fallback, and reuses the shared prediction formatter.

Usage

```
onnxPredict(
  model,
  newdata,
  batch_size = 64L,
  type = c("class", "probability", "response", "logit"),
  providers = "CPUExecutionProvider"
)
```

Arguments

<code>model</code>	A validated <code>physio_onnx_model</code> .
<code>newdata</code>	A compatible <code>physio_torch_dataset</code> .
<code>batch_size</code>	Positive whole-number inference batch size.
<code>type</code>	Exact output type.
<code>providers</code>	Execution provider; currently CPU only.

Value

The same identity-preserving output shapes as `predictModel()`.

Examples

```
# `model` is from importONNX(); `newdata` is a matching peDataset() (CPU ORT).
onnxPredict(model, newdata, type = "class")
```

peDataset	Create a case-aware torch Dataset
-----------	-----------------------------------

Description

Converts a `PhysioExperiment` assay to fixed channel-by-time items. Split cases before calling this function, fit normalization on training cases only, and pass `normalization_stats` to validation and prediction datasets. Class tensors are zero-based `torch_long`; input and regression tensors are `torch_float32`. A one-item batch always retains its batch dimension.

Usage

```
peDataset(
  x,
  targets = NULL,
  assay_name = NULL,
  cases = NULL,
  channels = NULL,
  window_samples = NULL,
  stride_samples = NULL,
  normalization = c("none", "zscore", "robust"),
  normalization_stats = NULL,
  task = c("classification", "regression"),
  class_levels = NULL,
  dtype = "float32"
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> ; attach subject keys with <code>withCaseData()</code> .
<code>targets</code>	Optional one-per-selected-case target vector.
<code>assay_name, cases, channels</code>	Exact assay and identity selections.
<code>window_samples, stride_samples</code>	Optional complete-window dimensions.
<code>normalization</code>	Exact normalization method.

```

normalization_stats    Frozen statistics from a training dataset.
task                   Exact task name.
class_levels           Optional ordered classification labels.
dtype                  Currently exactly "float32".

```

Value

A `physio_torch_dataset`.

Examples

```

# Prepare the input offline: a time x channel x case assay with channel labels.
arr <- array(stats::rnorm(50 * 2 * 6), dim = c(50, 2, 6))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)
# Building the Dataset itself needs the optional torch backend.
if (requireNamespace("torch", quietly = TRUE)) {
  ds <- peDataset(
    pe,
    targets = rep(c("rest", "task"), 3),
    task = "classification",
    class_levels = c("rest", "task")
  )
}

```

peModel

Construct a reference torch model

Description

Builds compact EEGNet, CNN1D, causal TCN, or unidirectional/bidirectional LSTM modules. Classification models return logits; regression models return raw values. Input is always batch by channel by time.

Usage

```

peModel(
  architecture = c("eegnet", "cnn1d", "tcn", "lstm"),
  n_channels,
  n_time,
  n_outputs,
  task = c("classification", "regression"),
  sampling_rate = NULL,
  dropout = 0.5,

```

```

    hidden = 32L,
    seed = 1L,
    ...
  )

```

Arguments

architecture	Exact architecture name.
n_channels, n_time, n_outputs	Positive whole dimensions.
task	Exact task name.
sampling_rate	Optional positive sampling rate.
dropout	Dropout probability in $[0, 1)$.
hidden	Positive hidden width.
seed	CPU initialization seed.
...	Architecture-specific named settings.

Value

A `physio_torch_module`.

Examples

```

# Requires the optional torch backend.
if (requireNamespace("torch", quietly = TRUE)) {
  model <- peModel(
    architecture = "cnn1d",
    n_channels = 2, n_time = 50, n_outputs = 2,
    task = "classification"
  )
}

```

peReducedFeatures	<i>Create a case-by-feature design matrix</i>
-------------------	---

Description

Adapts `catch22()`, `rocket()`, or `minirocket()` output without adding an outcome or changing the feature transform. For convolution methods, pass a model fitted only on training cases when transforming validation/test cases.

Usage

```
peReducedFeatures(
  x,
  method = c("catch22", "rocket", "minirocket"),
  model = NULL,
  assay_name = NULL,
  channels = NULL,
  cases = NULL,
  ...
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> .
<code>method</code>	Exactly one of "catch22", "rocket", or "minirocket".
<code>model</code>	NULL or a fitted convolution-transform model. Must be NULL for catch22.
<code>assay_name, channels, cases</code>	Input selection; see catch22() .
<code>...</code>	Method-specific arguments.

Value

A numeric matrix with class `physio_feature_matrix`. Attributes preserve exact case identity, feature provenance, typed diagnostics, and the fitted model when applicable.

Examples

```
arr <- array(stats::rnorm(40 * 2 * 3), dim = c(40, 2, 3))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)
# catch22 features need the optional Rcatch22 backend.
if (requireNamespace("Rcatch22", quietly = TRUE)) {
  features <- peReducedFeatures(pe, method = "catch22")
  dim(features)
}
```

predictModel

Predict with a trained deep-learning model

Description

Inference uses evaluation mode and disabled gradients. The Dataset must exactly match the training channel, time, task, class, dtype, and frozen normalization contracts.

Usage

```
predictModel(
  model,
  newdata,
  batch_size = 64L,
  type = c("class", "probability", "response", "logit"),
  num_workers = 0L
)
```

Arguments

model	A fitted <code>physio_dl_model</code> .
newdata	A compatible target-free or targeted Dataset.
batch_size	Positive whole-number inference batch size.
type	Exact output type.
num_workers	Currently exactly zero.

Value

A numeric matrix or identity-preserving data frame.

Examples

```
arr <- array(stats::rnorm(50 * 2 * 6), dim = c(50, 2, 6))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)
# Needs the optional torch and luz backends.
if (requireNamespace("torch", quietly = TRUE) &&
    requireNamespace("luz", quietly = TRUE)) {
  train <- peDataset(
    pe, targets = rep(c("a", "b"), 3),
    task = "classification", class_levels = c("a", "b")
  )
  fit <- trainModel(train, model = "cnn1d", epochs = 1L)
  predictModel(fit, train, type = "class")
}
```

rocket

Fit or apply the ROCKET transform

Description

Uses `aeon.transformations.collection.convolution_based.Rocket` through an optional, caller-configured [reticulate](#) backend. With `model = NULL`, fits only on the selected cases and returns a reusable persistent model. With a model, validates the complete input contract and transforms without refitting.

Usage

```
rocket(
  x,
  model = NULL,
  assay_name = NULL,
  channels = NULL,
  cases = NULL,
  n_kernels = 10000L,
  seed = 1L,
  normalize = TRUE,
  backend = "aeon"
)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> .
<code>model</code>	NULL or a fitted <code>physio_rocket_model</code> .
<code>assay_name</code> , <code>channels</code> , <code>cases</code>	Input selection; see catch22() .
<code>n_kernels</code>	One finite whole number of kernels.
<code>seed</code>	One whole-number backend seed in $[0, 2^{31} - 1]$.
<code>normalize</code>	One non-missing logical passed to <code>aeon</code> 's <code>normalise</code> .
<code>backend</code>	Exactly "aeon".

Details

Input assays are time x channel or time x channel x case and must contain finite, equal-length series of at least nine samples. The fitted model binds exact channel labels/order and time length. Requested kernel count and realized feature width are recorded separately because backend transforms may adjust the width.

Models persist the backend with Python pickle in a hashed raw vector. A matching SHA-256 detects corruption but does not make deserialization safe: only reuse model objects from trusted sources. `PhysioML` validates class, metadata, hash, input identity, and backend version before unpickling.

Value

A `physio_rocket_transform` containing a finite case-by-feature matrix, exact case identities, a persistent fitted model, settings, and typed diagnostics.

References

Dempster A, Petitjean F, Webb GI (2020). ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34, 1454-1495. doi:[10.1007/s1061802000701z](https://doi.org/10.1007/s1061802000701z)

Examples

```
arr <- array(stats::rnorm(60 * 2 * 4), dim = c(60, 2, 4))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)

# The transform needs a caller-managed Python environment with NumPy and aeon.
fit <- rocket(pe, n_kernels = 50)
dim(fit$features)
```

trainModel

Train a reference torch model through luz

Description

The caller supplies case-disjoint training and validation Datasets. When subject metadata is attached with `withCaseData()`, both datasets must carry it and subjects must also be disjoint. Legacy unannotated inputs retain case-only checking; missing metadata does not establish subject separation. Validation must reuse training normalization statistics. CPU execution, no worker processes, and deterministic caller-state restoration are enforced in this implementation.

Usage

```
trainModel(
  train,
  valid = NULL,
  model = c("eegnet", "cnn1d", "tcn", "lstm"),
  epochs = 20L,
  batch_size = 32L,
  learning_rate = 0.001,
  weight_decay = 0,
  callbacks = NULL,
  seed = 1L,
  device = "cpu",
  num_workers = 0L,
  verbose = FALSE,
  ...
)
```

Arguments

train, valid	Targeted physio_torch_dataset objects.
model	Exact architecture name or a physio_torch_module.

epochs, batch_size	Positive whole-number training settings.
learning_rate	Positive Adam learning rate.
weight_decay	Non-negative Adam weight decay.
callbacks	NULL or a list of luz callback instances.
seed	Whole-number CPU seed.
device	Currently exactly "cpu".
num_workers	Currently exactly zero.
verbose	Whether luz prints progress.
...	Architecture settings when model is a name.

Value

A physio_dl_model.

Examples

```
arr <- array(stats::rnorm(50 * 2 * 6), dim = c(50, 2, 6))
pe <- PhysioExperiment::PhysioExperiment(
  assays = list(raw = arr),
  colData = S4Vectors::DataFrame(label = c("C3", "C4")),
  samplingRate = 100
)
# Training needs the optional torch and luz backends.
if (requireNamespace("torch", quietly = TRUE) &&
    requireNamespace("luz", quietly = TRUE)) {
  train <- peDataset(
    pe, targets = rep(c("a", "b"), 3),
    task = "classification", class_levels = c("a", "b")
  )
  fit <- trainModel(train, model = "cnn1d", epochs = 1L)
}
```

validateSubjectSplit	<i>Validate a split intended to generalize to unseen subjects</i>
----------------------	---

Description

Checks explicit identifiers only; this does not audit upstream preprocessing fit scopes, verify biological identity, or prevent leakage in external model software unless called before fitting. No backend is needed.

Usage

```
validateSubjectSplit(train, valid)
```

Arguments

`train, valid` Case identity data frames as in `withCaseData()`.

Value

Invisibly TRUE, or an error for shared cases or subjects, invalid labels, or inconsistent participant aliases.

Examples

```
train <- data.frame(case_id = c("c1", "c2"), subject_id = c("s1", "s2"))
valid <- data.frame(case_id = c("c3", "c4"), subject_id = c("s3", "s4"))
validateSubjectSplit(train, valid)
```

<code>withCaseData</code>	<i>Attach explicit subject identity to signal cases or feature rows</i>
---------------------------	---

Description

Establishes a case-to-subject mapping without inferring subjects from case names. Row order is reconciled by exact case ID. A cycle is a case, not an independent subject. Use cohort-derived subject/session keys when available.

Usage

```
withCaseData(x, case_data)
```

Arguments

<code>x</code>	A <code>PhysioExperiment</code> or finite numeric case-by-feature matrix with unique nonempty row names used as case IDs.
<code>case_data</code>	Data frame with unique <code>case_id</code> and repeated <code>subject_id</code> . Optional character identity columns are <code>participant_id</code> , <code>session_id</code> , <code>trial_id</code> , <code>side</code> , <code>cycle_id</code> . All supplied columns must be complete. An optional <code>input_case_index</code> from a <code>PhysioML</code> transform is accepted as unique positive whole numbers; signal selection computes its own indices. The table must cover all input cases exactly, before any case selection.

Value

The input with validated identity metadata. Experiments store it in `metadata(x)$ml_case_data`; matrices store it in attribute `case_data`. `PhysioML` input transforms and datasets preserve this information. Ordinary external matrix manipulation is not guaranteed to preserve attributes; reattach the corresponding table after subsetting feature rows.

See Also

[validateSubjectSplit\(\)](#), [peDataset\(\)](#), [peReducedFeatures\(\)](#)

Examples

```
# Attach subject identity to a case-by-feature matrix.
features <- matrix(
  stats::rnorm(8), nrow = 2,
  dimnames = list(c("c1", "c2"), c("a", "b", "c", "d"))
)
case_data <- data.frame(case_id = c("c1", "c2"), subject_id = c("s1", "s2"))
labelled <- withCaseData(features, case_data)
attr(labelled, "case_data")
```

Index

cacheFoundationModel, 3
catch22, 4
catch22(), 13, 17, 18, 20

domainAdapt, 5

exportONNX, 6

fineTune, 7
foundationCacheInfo, 9
foundationEmbed, 9
freezeLayers, 11

importONNX, 12

minirocket, 13
minirocket(), 17

onnxPredict, 14

peDataset, 15
peDataset(), 23
peModel, 16
peReducedFeatures, 17
peReducedFeatures(), 23
PhysioExperiment::defaultAssay(), 4
predictModel, 18
predictModel(), 15

Rcatch22::catch22_all(), 4
reticulate, 19
rocket, 19
rocket(), 14, 17

S4Vectors::DataFrame(), 4

trainModel, 21
trainModel(), 8

validateSubjectSplit, 22
validateSubjectSplit(), 23

withCaseData, 23
withCaseData(), 15, 21, 23