Package ‘DeepPatientLevelPrediction’

August 17, 2022

Type Package

Title Deep Learning For Patient Level Prediction Using Data In The OMOP Common Data Model

Version 0.0.1

Date 2021-06-07

Maintainer Egill Fridgeirsson <e.fridgeirsson@erasmusmc.nl>

Description A package for creating deep learning patient level prediction models following the OHDSI PatientLevelPrediction framework.

License Apache License 2.0

URL https://ohdsi.github.io/PatientLevelPrediction, https://github.com/OHDSI/DeepPatientLevelPrediction

BugReports https://github.com/OHDSI/DeepPatientLevelPrediction/issues

VignetteBuilder knitr

Depends R (>= 3.5.0)

Imports dplyr,
data.table,
FeatureExtraction (>= 3.0.0),
ParallelLogger (>= 2.0.0),
PatientLevelPrediction,
rlang,
torch (>= 0.8.0)

Suggests devtools,
Eunomia,
knitr,
markdown,
plyr,
testthat

Remotes ohdsi/PatientLevelPrediction@develop,
ohdsi/FeatureExtraction,
ohdsi/Eunomia

RoxygenNote 7.2.0

Encoding UTF-8

Config/testthat/edition 3

R topics documented:

`Dataset` *A torch dataset*

Description

A torch dataset

Usage

```
Dataset(data, labels = NULL, numericalIndex = NULL, all = FALSE)
```

Arguments

`data`	a dataframe like object with the covariates
`labels`	a dataframe with the labels
`numericalIndex`	in what column numeric data is in (if any)
`all`	if True then returns all features instead of splitting num/cat

---

`DeepPatientLevelPrediction`
DeepPatientLevelPrediction

Description

A package containing deep learning extensions for developing prediction models using data in the OMOP CDM

doubleLayerNN *Double layer neural network*

Description

Double layer neural network

Usage

```
doubleLayerNN(inputN, layer1, layer2, outputN, layer_dropout)
```

Arguments

`inputN`	Input neurons
`layer1`	Layer 1 neurons
`layer2`	Layer 2 neurons
`outputN`	output neurons
`layer_dropout`	layer_dropout to use

EarlyStopping *Earlystopping class*

Description

Stops training if a loss or metric has stopped improving

Methods

Public methods:

- `EarlyStopping$new()`
- `EarlyStopping$call()`
- `EarlyStopping$clone()`

Method `new()`**:** Creates a new earlystopping object

Usage:

```
EarlyStopping$new(patience = 3, delta = 0, verbose = TRUE)
```

Arguments:

`patience` Stop after this number of epochs if loss doesn't improve

`delta` How much does the loss need to improve to count as improvement

`verbose` If information should be printed out

Returns: a new earlystopping object

Method `call()`**:** call the earlystopping object and increment a counter if loss is not improving

Usage:

```
EarlyStopping$call(metric)
```

Arguments:

`metric` the current metric value

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`Estimator` *Estimator*

Description

A generic R6 class that wraps around a torch nn module and can be used to fit and predict the model defined in that module.

Methods

Public methods:

- `Estimator$new()`
- `Estimator$fit()`
- `Estimator$fitEpoch()`
- `Estimator$score()`
- `Estimator$finishFit()`
- `Estimator$fitWholeTrainingSet()`
- `Estimator$save()`
- `Estimator$predictProba()`
- `Estimator$predict()`
- `Estimator$batchToDevice()`
- `Estimator$itemOrDefaults()`
- `Estimator$clone()`

Method `new()`**:** Creates a new estimator

Usage:

```
Estimator$new(
  baseModel,
  modelParameters,
  fitParameters,
  optimizer = torch::optim_adam,
  criterion = torch::nn_bce_with_logits_loss,
  scheduler = torch::lr_reduce_on_plateau,
  device = "cpu",
  patience = 4
)
```

Arguments:

`baseModel` The torch nn module to use as model

`modelParameters` Parameters to initialize the baseModel

`fitParameters` Parameters required for the estimator fitting

`optimizer` A torch optimizer to use, default is Adam

`criterion` The torch loss function to use, defaults to binary cross entropy with logits

`scheduler` learning rate scheduler to use

`device` Which device to use for fitting, default is cpu

`patience` Patience to use for early stopping

Method `fit()`: fits the estimator

Usage:

```
Estimator$fit(dataset, testDataset)
```

Arguments:

`dataset` a torch dataset to use for model fitting

`testDataset` a torch dataset to use for early stopping

Method `fitEpoch()`: fits estimator for one epoch (one round through the data)

Usage:

```
Estimator$fitEpoch(dataset, batchIndex)
```

Arguments:

`dataset` torch dataset to use for fitting

`batchIndex` indices of batches

Method `score()`: calculates loss and auc after training for one epoch

Usage:

```
Estimator$score(dataset, batchIndex)
```

Arguments:

`dataset` The torch dataset to use to evaluate loss and auc

`batchIndex` Indices of batches in the dataset

Returns: list with average loss and auc in the dataset

Method `finishFit()`: operations that run when fitting is finished

Usage:

```
Estimator$finishFit(valAUCs, modelStateDict, valLosses, epoch, learnRates)
```

Arguments:

`valAUCs` validation AUC values

`modelStateDict` fitted model parameters

`valLosses` validation losses

`epoch` list of epochs fit

`learnRates` learning rate sequence used so far

Method `fitWholeTrainingSet()`**:** Fits whole training set on a specific number of epochs TODO What happens when learning rate changes per epochs? Ideally I would copy the learning rate strategy from before and adjust for different sizes ie more iterations/updates???

Usage:

```
Estimator$fitWholeTrainingSet(dataset, learnRates = NULL)
```

Arguments:

`dataset` torch dataset

`learnRates` learnRateSchedule from CV

Method `save()`**:** save model and those parameters needed to reconstruct it

Usage:

```
Estimator$save(path, name)
```

Arguments:

`path` where to save the model

`name` name of file

Returns: the path to saved model

Method `predictProba()`**:** predicts and outputs the probabilities

Usage:

```
Estimator$predictProba(dataset)
```

Arguments:

`dataset` Torch dataset to create predictions for

Returns: predictions as probabilities

Method `predict()`**:** predicts and outputs the class

Usage:

```
Estimator$predict(dataset, threshold = NULL)
```

Arguments:

`dataset` A torch dataset to create predictions for

`threshold` Which threshold to use for predictions

Returns: The predicted class for the data in the dataset

Method `batchToDevice()`**:** sends a batch of data to device assumes batch includes lists of tensors to arbitrary nested depths

Usage:

```
Estimator$batchToDevice(batch)
```

Arguments:

`batch` the batch to send, usually a list of torch tensors

Returns: the batch on the required device

Method `itemOrDefaults()`**:** select item from list, and if it's null sets a default

Usage:

```
Estimator$itemOrDefaults(list, item, default = NULL)
```

Arguments:

`list` A list with items

`item` Which list item to retrieve

`default` The value to return if list doesn't have item

Returns: the list item or default

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`fitDeepNNTorch` *Fits a deep neural network*

Description

Fits a deep neural network

Usage

```
fitDeepNNTorch(trainData, modelSettings, search = "grid", analysisId)
```

Arguments

`trainData`	Training data object
`modelSettings`	modelSettings object
`search`	Which kind of search strategy to use
`analysisId`	Analysis Id

fitEstimator *fitEstimator*

Description

fits a deep learning estimator to data.

Usage

```
fitEstimator(trainData, modelSettings, analysisId, ...)
```

Arguments

`trainData`	the data to use
`modelSettings`	modelSettings object
`analysisId`	Id of the analysis
`...`	Extra inputs

gridCvDeep *gridCvDeep*

Description

Performs grid search for a deep learning estimator

Usage

```
gridCvDeep(mappedData, labels, settings, modelLocation, paramSearch)
```

Arguments

`mappedData`	Mapped data with covariates
`labels`	Dataframe with the outcomes
`settings`	Settings of the model
`modelLocation`	Where to save the model
`paramSearch`	model parameters to perform search over

`predictDeepEstimator` *predictDeepEstimator*

Description

the prediction function for the estimator

Usage

```
predictDeepEstimator(plpModel, data, cohort)
```

Arguments

`plpModel`	the plpModel
`data`	plp data object or a torch dataset
`cohort`	data.frame with the rowIds of the people

`predictDeepNN` *Create predictions for a deep neural network*

Description

Create predictions for a deep neural network

Usage

```
predictDeepNN(plpModel, data, cohort)
```

Arguments

`plpModel`	The plpModel to predict for
`data`	The data to make predictions for
`cohort`	The cohort to use

setDeepNNTorch *settings for a Deep neural network*

Description

settings for a Deep neural network

Usage

```
setDeepNNTorch(
  units = list(c(128, 64), 128),
  layer_dropout = c(0.2),
  lr = c(1e-04),
  decay = c(1e-05),
  outcome_weight = c(1),
  batch_size = c(10000),
  epochs = c(100),
  device = "cpu",
  seed = NULL
)
```

Arguments

`units`	A list of vectors for neurons per layer
`layer_dropout`	Dropout to use per layer
`lr`	Learning rate ot use
`decay`	Weight decay to use
`outcome_weight`	Weight for minority outcome in cost function
`batch_size`	Batch size to use
`epochs`	How many epochs to use
`device`	Which device to use
`seed`	A seed to make experiments more reproducible

setResNet *setResNet*

Description

Creates settings for a ResNet model

Usage

```
setResNet(
  numLayers = c(1:8),
  sizeHidden = c(2^(6:9)),
  hiddenFactor = c(1:4),
  residualDropout = c(seq(0, 0.5, 0.05)),
  hiddenDropout = c(seq(0, 0.5, 0.05)),
  sizeEmbedding = c(2^(6:9)),
  weightDecay = c(1e-06, 0.001),
  learningRate = c(0.01, 3e-04, 1e-05),
  seed = NULL,
  hyperParamSearch = "random",
  randomSample = 100,
  device = "cpu",
  batchSize = 1024,
  epochs = 30
)
```

Arguments

`numLayers` Number of layers in network, default: 1:16

`sizeHidden` Amount of neurons in each default layer, default: 2^(6:10) (64 to 1024)

`hiddenFactor` How much to grow the amount of neurons in each ResLayer, default: 1:4

`residualDropout`
How much dropout to apply after last linear layer in ResLayer, default: seq(0, 0.3, 0.05)

`hiddenDropout` How much dropout to apply after first linear layer in ResLayer, default: seq(0, 0.3, 0.05)

`sizeEmbedding` Size of embedding layer, default: 2^(6:9) (64 to 512)

`weightDecay` Weight decay to apply, default: c(1e-6, 1e-3)

`learningRate` Learning rate to use. default: c(1e-2, 1e-5)

`seed` Seed to use for sampling hyperparameter space

`hyperParamSearch`
Which kind of hyperparameter search to use random sampling or exhaustive grid search. default: ’random’

`randomSample` How many random samples from hyperparameter space to use

`device` Which device to run analysis on, either ’cpu’ or ’cuda’, default: ’cpu’

`batchSize` Size of batch, default: 1024

`epochs` Number of epochs to run, default: 10

Details

Model architecture from by https://arxiv.org/abs/2106.11959

`setTransformer` *create settings for training a non-temporal transformer*

Description

A transformer model

Usage

```
setTransformer(
  numBlocks = 3,
  dimToken = 96,
  dimOut = 1,
  numHeads = 8,
  attDropout = 0.25,
  ffnDropout = 0.25,
  resDropout = 0,
  dimHidden = 512,
  weightDecay = 1e-06,
  learningRate = 3e-04,
  batchSize = 1024,
  epochs = 10,
  device = "cpu",
  hyperParamSearch = "random",
  randomSamples = 100,
  seed = NULL
)
```

Arguments

`numBlocks`	number of transformer blocks
`dimToken`	dimension of each token (embedding size)
`dimOut`	dimension of output, usually 1 for binary problems
`numHeads`	number of attention heads
`attDropout`	dropout to use on attentions
`ffnDropout`	dropout to use in feedforward block
`resDropout`	dropout to use in residual connections
`dimHidden`	dimension of the feedworward block
`weightDecay`	weightdecay to use
`learningRate`	learning rate to use
`batchSize`	batchSize to use
`epochs`	How many epochs to run the model for
`device`	Which device to use, cpu or cuda

`hyperParamSearch`	what kind of hyperparameter search to do, default ’random’
`randomSamples`	How many samples to use in hyperparameter search if random
`seed`	Random seed to use

Details

from https://arxiv.org/abs/2106.11959

`singleLayerNN` *A single layer neural network*

Description

A single layer neural network

Usage

```
singleLayerNN(inputN, layer1, outputN = 2, layer_dropout)
```

Arguments

`inputN`	Input neurons
`layer1`	Layer 1 neurons
`outputN`	Output neurons
`layer_dropout`	Layer dropout to use

`tripleLayerNN` *Triple layer neural network*

Description

Triple layer neural network

Usage

```
tripleLayerNN(inputN, layer1, layer2, layer3, outputN, layer_dropout)
```

Arguments

`inputN`	Input neurons
`layer1`	amount of layer 1 neurons
`layer2`	amount of layer 2 neurons
`layer3`	amount of layer 3 neurons
`outputN`	Number of output neurons
`layer_dropout`	The dropout to use in layer