Package 'aifeducation'

October 14, 2025

Type Package

Title Artificial Intelligence for Education

Version 1.1.2

Description In social and educational settings, the use of Artificial Intelligence (AI) is a challenging task. Relevant data is often only available in handwritten forms, or the use of data is restricted by privacy policies. This often leads to small data sets. Furthermore, in the educational and social sciences, data is often unbalanced in terms of frequencies. To support educators as well as educational and social researchers in using the potentials of AI for their work, this package provides a unified interface for neural nets in 'PyTorch' to deal with natural language problems. In addition, the package ships with a shiny app, providing a graphical user interface. This allows the usage of AI for people without skills in writing python/R scripts. The tools integrate existing mathematical and statistical methods for dealing with small data sets via pseudo-labeling (e.g. Cascante-Bonilla et al. (2020) <doi:10.48550/arXiv.2001.06001>) and imbalanced data via the creation of synthetic cases (e.g. Islam et al. (2012) <doi:10.1016/j.asoc.2021.108288>). Performance evaluation of AI is connected to measures from content analysis which educational and social researchers are generally more familiar with (e.g. Berding & Pargmann (2022) <doi:10.30819/5581>, Gwet (2014) <ISBN:978-0-9708062-8-4>, Krippendorff (2019) <doi:10.4135/9781071878781>). Estimation of energy consumption and CO2 emissions during model training is done with the 'python' library 'codecarbon'. Finally, all objects created with this package allow to share trained AI models with other people.

License GPL-3

URL https://fberding.github.io/aifeducation/

BugReports https://github.com/cran/aifeducation/issues

Depends R (>= 3.5.0)

Imports doParallel, foreach, iotarelr(>= 0.1.5), methods, Rcpp (>= 1.0.10), reshape2, reticulate (>= 1.42.0), rlang, stringi, utils

Suggests bslib, DT, fs, future, ggplot2, knitr, pkgdown, promises, readtext, readxl, rmarkdown, shiny(>= 1.9.0), shinyFiles, shinyWidgets, shinycssloaders, sortable, testthat (>= 3.0.0)

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

SystemRequirements PyTorch (see vignette ``Get started")

Config/Needs/website rmarkdown

NeedsCompilation yes

Author Berding Florian [aut, cre] (ORCID: <https://orcid.org/0000-0002-3593-1695>), Tykhonova Yuliia [aut] (ORCID: <https://orcid.org/0009-0006-9015-1006>), Pargmann Julia [ctb] (ORCID: <https://orcid.org/0000-0003-3616-0172>), Leube Anna [ctb] (ORCID: <https://orcid.org/0009-0001-6949-1608>), Riebenbauer Elisabeth [ctb] (ORCID: <https://orcid.org/0000-0002-8535-3694>), Rebmann Karin [ctb], Slopinski Andreas [ctb]

Maintainer Berding Florian <florian.berding@uni-hamburg.de>

Repository CRAN

Date/Publication 2025-10-14 14:50:02 UTC

Contents

`add_missing_args` *Add missing arguments to a list of arguments*

Description

This function is designed for taking the output of `summarize_args_for_long_task` as input. It adds the missing arguments. In general these are arguments that rely on objects of class R6 which can not be exported to a new R session.

Usage

```
add_missing_args(args, path_args, meta_args)
```

Arguments

`args`	Named `list` List for arguments for the method of a specific class.
`path_args`	Named `list` List of paths where the objects are stored on disk.
`meta_args`	Named `list` List containing arguments that are necessary in order to add the missing objects correctly.

Value

Returns a named `list` of all arguments that a method of a specific class requires.

See Also

Other Utils Studio Developers: `create_data_embeddings_description()`, `long_load_target_data()`, `summarize_args_for_long_task()`

`AIFEBaseModel` *Base class for objects using a pytorch model as core model.*

Description

Objects of this class containing fields and methods used in several other classes in 'AI for Education'.

This class is **not** designed for a direct application and should only be used by developers.

Value

A new object of this class.

Super class

`aifeducation::AIFEMaster` -> `AIFEBaseModel`

Methods

Public methods:

- AIFEBaseModel$count_parameter()
- AIFEBaseModel$clone()

Method `count_parameter()`**:** Method for counting the trainable parameters of a model.

Usage:

```
AIFEBaseModel$count_parameter()
```

Returns: Returns the number of trainable parameters of the model.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: AIFEMaster, BaseModelCore, ClassifiersBasedOnTextEmbeddings, DataManagerClassifier, LargeDataSetBase, ModelsBasedOnTextEmbeddings, TEClassifiersBasedOnProtoNet, TEClassifiersBasedOnRegular, TokenizerBase

---

`AIFEMaster` *Base class for most objects*

Description

Objects of this class containing fields and methods used in several other classes in 'AI for Education'.

This class is **not** designed for a direct application and should only be used by developers.

Value

A new object of this class.

Public fields

`last_training` ('list()')

List for storing the history, the configuration, and the results of the last training. This information will be overwritten if a new training is started.

- `last_training$start_time`: Time point when training started.
- `last_training$learning_time`: Duration of the training process.
- `last_training$finish_time`: Time when the last training finished.
- `last_training$history`: History of the last training.
- `last_training$data`: Object of class `table` storing the initial frequencies of the passed data.
- `last_training$config`: List storing the configuration used for the last training.

Methods

Public methods:

- AIFEMaster$get_model_info()
- AIFEMaster$set_publication_info()
- AIFEMaster$get_publication_info()
- AIFEMaster$set_model_license()
- AIFEMaster$get_model_license()
- AIFEMaster$set_documentation_license()
- AIFEMaster$get_documentation_license()
- AIFEMaster$set_model_description()
- AIFEMaster$get_model_description()
- AIFEMaster$get_package_versions()
- AIFEMaster$get_sustainability_data()
- AIFEMaster$get_ml_framework()
- AIFEMaster$is_configured()
- AIFEMaster$is_trained()
- AIFEMaster$get_private()
- AIFEMaster$get_all_fields()
- AIFEMaster$get_model_config()
- AIFEMaster$clone()

Method `get_model_info()`**:** Method for requesting the model information.

Usage:

```
AIFEMaster$get_model_info()
```

Returns: `list` of all relevant model information.

Method `set_publication_info()`**:** Method for setting publication information of the model.

Usage:

```
AIFEMaster$set_publication_info(authors, citation, url = NULL)
```

Arguments:

`authors` List of authors.

`citation` Free text citation.

`url` URL of a corresponding homepage.

Returns: Function does not return a value. It is used for setting the private members for publication information.

Method `get_publication_info()`**:** Method for requesting the bibliographic information of the model.

Usage:

```
AIFEMaster$get_publication_info()
```

Returns: `list` with all saved bibliographic information.

Method `set_model_license()`**:** Method for setting the license of the model.

Usage:

```
AIFEMaster$set_model_license(license = "CC BY")
```

Arguments:

`license` `string` containing the abbreviation of the license or the license text.

Returns: Function does not return a value. It is used for setting the private member for the software license of the model.

Method `get_model_license()`**:** Method for getting the license of the model.

Usage:

```
AIFEMaster$get_model_license()
```

Arguments:

`license` `string` containing the abbreviation of the license or the license text.

Returns: `string` representing the license for the model.

Method `set_documentation_license()`**:** Method for setting the license of the model's documentation.

Usage:

```
AIFEMaster$set_documentation_license(license = "CC BY")
```

Arguments:

`license` `string` containing the abbreviation of the license or the license text.

Returns: Function does not return a value. It is used for setting the private member for the documentation license of the model.

Method `get_documentation_license()`**:** Method for getting the license of the model's documentation.

Usage:

```
AIFEMaster$get_documentation_license()
```

Arguments:

`license` `string` containing the abbreviation of the license or the license text.

Returns: Returns the license as a `string`.

Method `set_model_description()`**:** Method for setting a description of the model.

Usage:

```
AIFEMaster$set_model_description(
  eng = NULL,
  native = NULL,
  abstract_eng = NULL,
  abstract_native = NULL,
  keywords_eng = NULL,
  keywords_native = NULL
)
```

Arguments:

`eng string` A text describing the training, its theoretical and empirical background, and output in English.

`native string` A text describing the training , its theoretical and empirical background, and output in the native language of the model.

`abstract_eng string` A text providing a summary of the description in English.

`abstract_native string` A text providing a summary of the description in the native language of the model.

`keywords_eng vector` of keyword in English.

`keywords_native vector` of keyword in the native language of the model.

Returns: Function does not return a value. It is used for setting the private members for the description of the model.

Method `get_model_description()`**:** Method for requesting the model description.

Usage:

```
AIFEMaster$get_model_description()
```

Returns: `list` with the description of the classifier in English and the native language.

Method `get_package_versions()`**:** Method for requesting a summary of the R and python packages' versions used for creating the model.

Usage:

```
AIFEMaster$get_package_versions()
```

Returns: Returns a `list` containing the versions of the relevant R and python packages.

Method `get_sustainability_data()`**:** Method for requesting a summary of tracked energy consumption during training and an estimate of the resulting CO2 equivalents in kg.

Usage:

```
AIFEMaster$get_sustainability_data(track_mode = "training")
```

Arguments:

`track_mode string` Determines the stept to which the data refer. Allowed values: 'training', 'inference'

Returns: Returns a `list` containing the tracked energy consumption, CO2 equivalents in kg, information on the tracker used, and technical information on the training infrastructure.

Method `get_ml_framework()`**:** Method for requesting the machine learning framework used for the model.

Usage:

```
AIFEMaster$get_ml_framework()
```

Returns: Returns a `string` describing the machine learning framework used for the classifier.

Method `is_configured()`**:** Method for checking if the model was successfully configured. An object can only be used if this value is `TRUE`.

Usage:

```
AIFEMaster$is_configured()
```

Returns: `bool` `TRUE` if the model is fully configured. `FALSE` if not.

Method `is_trained()`**:** Check if the TEFeatureExtractor is trained.

Usage:

```
AIFEMaster$is_trained()
```

Returns: Returns `TRUE` if the object is trained and `FALSE` if not.

Method `get_private()`**:** Method for requesting all private fields and methods. Used for loading and updating an object.

Usage:

```
AIFEMaster$get_private()
```

Returns: Returns a `list` with all private fields and methods.

Method `get_all_fields()`**:** Return all fields.

Usage:

```
AIFEMaster$get_all_fields()
```

Returns: Method returns a `list` containing all public and private fields of the object.

Method `get_model_config()`**:** Method for requesting the model configuration.

Usage:

```
AIFEMaster$get_model_config()
```

Returns: Returns a `list` with all configuration parameters used during configuration.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `BaseModelCore`, `ClassifiersBasedOnTextEmbeddings`, `DataManagerClassifier`, `LargeDataSetBase`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnProtoNet`, `TEClassifiersBasedOnRegular`, `TokenizerBase`

`auto_n_cores` *Number of cores for multiple tasks*

Description

Function for getting the number of cores that should be used for parallel processing of tasks. The number of cores is set to 75 % of the available cores. If the environment variable `CI` is set to `"true"` or if the process is running on cran 2 is returned.

Usage

```
auto_n_cores()
```

Value

Returns `int` as the number of cores.

See Also

Other Utils Developers: `create_object()`, `create_synthetic_units_from_matrix()`, `generate_id()`, `get_n_chunks()`, `get_synthetic_cases_from_matrix()`, `get_time_stamp()`, `matrix_to_array_c()`, `tensor_to_matrix_c()`, `to_categorical_c()`

BaseModelBert *BERT-Transformer*

Description

Represents models based on BERT.

Value

Does return a new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::BaseModelCore` -> `BaseModelBert`

Methods

Public methods:

- `BaseModelBert$configure()`
- `BaseModelBert$clone()`

Method `configure()`**:** Configures a new object of this class.

Usage:

```
BaseModelBert$configure(
  tokenizer,
  max_position_embeddings = 512L,
  hidden_size = 768L,
  num_hidden_layers = 12L,
  num_attention_heads = 12L,
  intermediate_size = 3072L,
  hidden_act = "GELU",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1
)
```

Arguments:

`tokenizer` `TokenizerBase` Tokenizer for the model.

`max_position_embeddings` `int` Number of maximum position embeddings. This parameter also determines the maximum length of a sequence which can be processed with the model. Allowed values: `10 <= x <= 4048`

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding. Allowed values: `1 <= x <= 2048`

`num_hidden_layers` `int` Number of hidden layers. Allowed values: `1 <= x`

`num_attention_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`hidden_act` `string` Name of the activation function. Allowed values: 'GELU', 'relu', 'silu', 'gelu_new'

`hidden_dropout_prob` `double` Ratio of dropout. Allowed values: `0 <= x <= 0.6`

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities. Allowed values: `0 <= x <= 0.6`

Returns: Does nothing return.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), Proceedings of the 2019 Conference of the North (pp. 4171–4186). Association for Computational Linguistics. doi:10.18653/v1/N191423

See Also

Other Base Model: `BaseModelDebertaV2`, `BaseModelFunnel`, `BaseModelMPNet`, `BaseModelModernBert`, `BaseModelRoberta`

`BaseModelCore` *Abstract class for all BaseModels*

Description

This class contains all methods shared by all BaseModels.

Value

Does return a new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `BaseModelCore`

Public fields

`Tokenizer` ('TokenizerBase')
Objects of class `TokenizerBase`.

Methods

Public methods:

- `BaseModelCore$create_from_hf()`
- `BaseModelCore$train()`
- `BaseModelCore$count_parameter()`
- `BaseModelCore$plot_training_history()`
- `BaseModelCore$get_special_tokens()`
- `BaseModelCore$get_tokenizer_statistics()`
- `BaseModelCore$fill_mask()`
- `BaseModelCore$save()`
- `BaseModelCore$load_from_disk()`
- `BaseModelCore$get_model()`
- `BaseModelCore$get_model_type()`
- `BaseModelCore$get_final_size()`
- `BaseModelCore$get_flops_estimates()`
- `BaseModelCore$set_publication_info()`
- `BaseModelCore$estimate_sustainability_inference_fill_mask()`
- `BaseModelCore$calc_flops_architecture_based()`
- `BaseModelCore$clone()`

Method `create_from_hf()`**:** Creates BaseModel from a pretrained model

Usage:

```
BaseModelCore$create_from_hf(model_dir = NULL, tokenizer_dir = NULL)
```

Arguments:

`model_dir`

`tokenizer_dir` `string` Path to the directory where the tokenizer is saved. Allowed values: any

Returns: Does return a new object of this class.

Method `train()`**:** Traines a BaseModel

Usage:

```
BaseModelCore$train(
  text_dataset,
  p_mask = 0.15,
  whole_word = TRUE,
  val_size = 0.1,
  n_epoch = 1L,
  batch_size = 12L,
  max_sequence_length = 250L,
  full_sequences_only = FALSE,
  min_seq_len = 50L,
  learning_rate = 0.003,
  sustain_track = FALSE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15L,
  sustain_log_level = "warning",
  trace = TRUE,
  pytorch_trace = 1L,
  log_dir = NULL,
  log_write_interval = 2L
)
```

Arguments:

`text_dataset`

`p_mask`

`whole_word`

`val_size`

`n_epoch`

`batch_size`

`max_sequence_length`

`full_sequences_only`

`min_seq_len`

`learning_rate`

`sustain_track`

`sustain_iso_code`

`sustain_region`

`sustain_interval`

`sustain_log_level`

`trace`

`pytorch_trace`

`log_dir`

`log_write_interval`

Returns: Does nothing return.

Method `count_parameter()`**:** Method for counting the trainable parameters of a model.

Usage:

```
BaseModelCore$count_parameter()
```

Returns: Returns the number of trainable parameters of the model.

Method `plot_training_history()`**:** Method for requesting a plot of the training history. This method requires the *R* package 'ggplot2' to work.

Usage:

```
BaseModelCore$plot_training_history(
  y_min = NULL,
  y_max = NULL,
  text_size = 10L
)
```

Arguments:

`y_min`

`y_max`

`text_size`

Returns: Returns a plot of class `ggplot` visualizing the training process.

Method `get_special_tokens()`**:** Method for receiving the special tokens of the model

Usage:

```
BaseModelCore$get_special_tokens()
```

Returns: Returns a `matrix` containing the special tokens in the rows and their type, token, and id in the columns.

Method `get_tokenizer_statistics()`**:** Tokenizer statistics

Usage:

```
BaseModelCore$get_tokenizer_statistics()
```

Returns: Returns a `data.frame` containing the tokenizer's statistics.

Method `fill_mask()`**:** Method for calculating tokens behind mask tokens.

Usage:

```
BaseModelCore$fill_mask(masked_text, n_solutions = 5L)
```

Arguments:

`masked_text`

`n_solutions`

Returns: Returns a `list` containing a `data.frame` for every mask. The `data.frame` contains the solutions in the rows and reports the score, token id, and token string in the columns.

Method `save()`**:** Method for saving a model on disk.

Usage:

```
BaseModelCore$save(dir_path, folder_name)
```

Arguments:

`dir_path` Path to the directory where to save the object.

`folder_name` `string` Name of the folder where the model should be saved. Allowed values: any

Returns: Function does nothing return. It is used to save an object on disk.

Method `load_from_disk()`**:** Loads an object from disk and updates the object to the current version of the package.

Usage:

```
BaseModelCore$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the object set is stored.

Returns: Function does nothin return. It loads an object from disk.

Method `get_model()`**:** Get 'PyTorch' model

Usage:

```
BaseModelCore$get_model()
```

Returns: Returns the underlying 'PyTorch' model.

Method `get_model_type()`**:** Type of the underlying model.

Usage:

```
BaseModelCore$get_model_type()
```

Returns: Returns a `string` describing the model's architecture.

Method `get_final_size()`**:** Size of the final layer.

Usage:

```
BaseModelCore$get_final_size()
```

Returns: Returns an `int` describing the number of dimensions of the last hidden layer.

Method `get_flops_estimates()`**:** Flop estimates

Usage:

```
BaseModelCore$get_flops_estimates()
```

Returns: Returns a `data.frame` containing statistics about the flops.

Method `set_publication_info()`**:** Method for setting the bibliographic information of the model.

Usage:

```
BaseModelCore$set_publication_info(type, authors, citation, url = NULL)
```

Arguments:

`type` `string` Type of information which should be changed/added. `developer`, and `modifier` are possible.

`authors` List of people.

`citation` `string` Citation in free text.

`url` `string` Corresponding URL if applicable.

Returns: Function does not return a value. It is used to set the private members for publication information of the model.

Method `estimate_sustainability_inference_fill_mask()`**:** Calculates the energy consumption for inference of the given task.

Usage:

```
BaseModelCore$estimate_sustainability_inference_fill_mask(
  text_dataset = NULL,
  n = NULL,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15L,
  sustain_log_level = "warning",
  trace = TRUE
)
```

Arguments:

`text_dataset`

`n`

`sustain_iso_code`

`sustain_region`

`sustain_interval`

`sustain_log_level`

`trace`

Returns: Returns nothing. Method saves the statistics internally. The statistics can be accessed with the method `get_sustainability_data("inference")`

Method `calc_flops_architecture_based()`**:** Calculates FLOPS based on model's architecture.

Usage:

```
BaseModelCore$calc_flops_architecture_based(batch_size, n_batches, n_epochs)
```

Arguments:

`batch_size`

`n_batches`

`n_epochs`

Returns: Returns a `data.frame` storing the estimates.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `ClassifiersBasedOnTextEmbeddings`, `DataManagerClassifier`, `LargeDataSetBase`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnProtoNet`, `TEClassifiersBasedOnRegular`, `TokenizerBase`

`BaseModelDebertaV2` *DeBERTa V2*

Description

Represents models based on DeBERTa version 2.

Value

Does return a new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::BaseModelCore` -> `BaseModelDebertaV2`

Methods

Public methods:

- `BaseModelDebertaV2$configure()`
- `BaseModelDebertaV2$clone()`

Method `configure()`**:** Configures a new object of this class.

Usage:

```
BaseModelDebertaV2$configure(
  tokenizer,
  max_position_embeddings = 512L,
  hidden_size = 768L,
  num_hidden_layers = 12L,
  num_attention_heads = 12L,
  intermediate_size = 3072L,
  hidden_act = "GELU",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1
)
```

Arguments:

`tokenizer` `TokenizerBase` Tokenizer for the model.

`max_position_embeddings` `int` Number of maximum position embeddings. This parameter also determines the maximum length of a sequence which can be processed with the model. Allowed values: `10 <= x <= 4048`

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding. Allowed values: `1 <= x <= 2048`

`num_hidden_layers` `int` Number of hidden layers. Allowed values: `1 <= x`

`num_attention_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`hidden_act` `string` Name of the activation function. Allowed values: 'GELU', 'relu', 'silu', 'gelu_new'

`hidden_dropout_prob` `double` Ratio of dropout. Allowed values: `0 <= x <= 0.6`

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities. Allowed values: `0 <= x <= 0.6`

Returns: Does nothing return.

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

Usage:

```
BaseModelDebertaV2$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

He, P., Liu, X., Gao, J. & Chen, W. (2020). DeBERTa: Decoding-enhanced BERT with Disentangled Attention. doi:10.48550/arXiv.2006.03654

See Also

Other Base Model: `BaseModelBert`, `BaseModelFunnel`, `BaseModelMPNet`, `BaseModelModernBert`, `BaseModelRoberta`

`BaseModelFunnel` *Funnel transformer*

Description

Represents models based on the Funnel-Transformer.

Value

Does return a new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::BaseModelCore` -> `BaseModelFunnel`

Methods

Public methods:

- BaseModelFunnel$configure()
- BaseModelFunnel$clone()

Method `configure()`**:** Configures a new object of this class.

Usage:

```
BaseModelFunnel$configure(
  tokenizer,
  max_position_embeddings = 512L,
  hidden_size = 768L,
  block_sizes = c(4L, 4L, 4L),
  num_attention_heads = 12L,
  intermediate_size = 3072L,
  num_decoder_layers = 2L,
  d_head = 64L,
  funnel_pooling_type = "Mean",
  hidden_act = "GELU",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1,
  activation_dropout = 0
)
```

Arguments:

`tokenizer` `TokenizerBase` Tokenizer for the model.

`max_position_embeddings` `int` Number of maximum position embeddings. This parameter also determines the maximum length of a sequence which can be processed with the model. Allowed values: `10 <= x <= 4048`

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding. Allowed values: `1 <= x <= 2048`

`block_sizes` `vector` `vector` of `int` determining the number and sizes of each block.

`num_attention_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`num_decoder_layers` `int` Number of decoding layers. Allowed values: `1 <= x`

`d_head` `int` Number of neurons of the final layer. Allowed values: `1 <= x`

`funnel_pooling_type` `string` Method for pooling over the seqence length. Allowed values: 'Mean', 'Max'

`hidden_act` `string` Name of the activation function. Allowed values: 'GELU', 'relu', 'silu', 'gelu_new'

`hidden_dropout_prob` `double` Ratio of dropout. Allowed values: `0 <= x <= 0.6`

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities. Allowed values: `0 <= x <= 0.6`

`activation_dropout` `double` Dropout probability between the layers of the feed-forward blocks. Allowed values: `0 <= x <= 0.6`

num_hidden_layers int Number of hidden layers. Allowed values: 1 <= x

Returns: Does nothing return.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Dai, Z., Lai, G., Yang, Y. & Le, Q. V. (2020). Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. doi:10.48550/arXiv.2006.03236

See Also

Other Base Model: BaseModelBert, BaseModelDebertaV2, BaseModelMPNet, BaseModelModernBert, BaseModelRoberta

---

BaseModelModernBert *ModernBert*

Description

Represents models based on Modern Bert.

Value

Does return a new object of this class.

Super classes

aifeducation::AIFEMaster -> aifeducation::AIFEBaseModel -> aifeducation::BaseModelCore -> BaseModelModernBert

Methods

Public methods:

- BaseModelModernBert$configure()
- BaseModelModernBert$clone()

Method configure(): Configures a new object of this class.

Usage:

```
BaseModelModernBert$configure(
  tokenizer,
  max_position_embeddings = 512L,
  hidden_size = 768L,
  num_hidden_layers = 12L,
  num_attention_heads = 12L,
  global_attn_every_n_layers = 3L,
  intermediate_size = 3072L,
  hidden_activation = "GELU",
  embedding_dropout = 0.1,
  mlp_dropout = 0.1,
  attention_dropout = 0.1
)
```

Arguments:

`tokenizer` `TokenizerBase` Tokenizer for the model.

`max_position_embeddings` `int` Number of maximum position embeddings. This parameter also determines the maximum length of a sequence which can be processed with the model. Allowed values: `10 <= x <= 4048`

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding. Allowed values: `1 <= x <= 2048`

`num_hidden_layers` `int` Number of hidden layers. Allowed values: `1 <= x`

`num_attention_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`global_attn_every_n_layers` `int` Number determining to use a global attention every x-th layer. Allowed values: `2 <= x <= 36`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`hidden_activation` `string` Name of the activation function. Allowed values: 'GELU', 'relu', 'silu', 'gelu_new'

`embedding_dropout` `double` Dropout chance for the embeddings. Allowed values: `0 <= x <= 0.6`

`mlp_dropout` `double` Dropout rate for the mlp layer. Allowed values: `0 <= x <= 0.6`

`attention_dropout` `double` Ratio of dropout for attention probabilities. Allowed values: `0 <= x <= 0.6`

Returns: Does nothing return.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), Proceedings of the 2019 Conference of the North (pp. 4171–4186). Association for Computational Linguistics. doi:10.18653/v1/N191423

See Also

Other Base Model: BaseModelBert, BaseModelDebertaV2, BaseModelFunnel, BaseModelMPNet, BaseModelRoberta

---

`BaseModelMPNet` *MPNet*

Description

Represents models based on MPNet.

Value

Does return a new object of this class.

Super classes

aifeducation::AIFEMaster -> aifeducation::AIFEBaseModel -> aifeducation::BaseModelCore -> BaseModelMPNet

Methods

Public methods:

- BaseModelMPNet$configure()
- BaseModelMPNet$train()
- BaseModelMPNet$clone()

Method `configure()`**:** Configures a new object of this class.

Usage:

```
BaseModelMPNet$configure(
  tokenizer,
  max_position_embeddings = 512L,
  hidden_size = 768L,
  num_hidden_layers = 12L,
  num_attention_heads = 12L,
  intermediate_size = 3072L,
  hidden_act = "GELU",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1
)
```

Arguments:

`tokenizer` `TokenizerBase` Tokenizer for the model.

`max_position_embeddings` `int` Number of maximum position embeddings. This parameter also determines the maximum length of a sequence which can be processed with the model. Allowed values: `10 <= x <= 4048`

`hidden_size int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding. Allowed values: `1 <= x <= 2048`

`num_hidden_layers int` Number of hidden layers. Allowed values: `1 <= x`

`num_attention_heads int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`intermediate_size int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`hidden_act string` Name of the activation function. Allowed values: 'GELU', 'relu', 'silu', 'gelu_new'

`hidden_dropout_prob double` Ratio of dropout. Allowed values: `0 <= x <= 0.6`

`attention_probs_dropout_prob double` Ratio of dropout for attention probabilities. Allowed values: `0 <= x <= 0.6`

Returns: Does nothing return.

Method `train()`**:** Traines a BaseModel

Usage:

```
BaseModelMPNet$train(
  text_dataset,
  p_mask = 0.15,
  p_perm = 0.15,
  whole_word = TRUE,
  val_size = 0.1,
  n_epoch = 1L,
  batch_size = 12L,
  max_sequence_length = 250L,
  full_sequences_only = FALSE,
  min_seq_len = 50L,
  learning_rate = 0.003,
  sustain_track = FALSE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15L,
  sustain_log_level = "warning",
  trace = TRUE,
  pytorch_trace = 1L,
  log_dir = NULL,
  log_write_interval = 2L
)
```

Arguments:

`text_dataset`

`p_mask`

`p_perm`

`whole_word`

`val_size`

`n_epoch`

`batch_size`

max_sequence_length
full_sequences_only
min_seq_len
learning_rate
sustain_track
sustain_iso_code
sustain_region
sustain_interval
sustain_log_level
trace
pytorch_trace
log_dir
log_write_interval

Returns: Does nothing return.

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References

Song,K., Tan, X., Qin, T., Lu, J. & Liu, T.-Y. (2020). MPNet: Masked and Permuted Pre-training for Language Understanding. doi:10.48550/arXiv.2004.09297

See Also

Other Base Model: `BaseModelBert`, `BaseModelDebertaV2`, `BaseModelFunnel`, `BaseModelModernBert`, `BaseModelRoberta`

BaseModelRoberta *RoBERTa*

Description

Represents models based on RoBERTa.

Value

Does return a new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::BaseModelCore` -> `BaseModelRoberta`

Methods

Public methods:

- BaseModelRoberta$configure()
- BaseModelRoberta$clone()

Method `configure()`**:** Configures a new object of this class.

Usage:

```
BaseModelRoberta$configure(
  tokenizer,
  max_position_embeddings = 512L,
  hidden_size = 768L,
  num_hidden_layers = 12L,
  num_attention_heads = 12L,
  intermediate_size = 3072L,
  hidden_act = "GELU",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1
)
```

Arguments:

`tokenizer` `TokenizerBase` Tokenizer for the model.

`max_position_embeddings` `int` Number of maximum position embeddings. This parameter also determines the maximum length of a sequence which can be processed with the model. Allowed values: `10 <= x <= 4048`

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding. Allowed values: `1 <= x <= 2048`

`num_hidden_layers` `int` Number of hidden layers. Allowed values: `1 <= x`

`num_attention_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`hidden_act` `string` Name of the activation function. Allowed values: 'GELU', 'relu', 'silu', 'gelu_new'

`hidden_dropout_prob` `double` Ratio of dropout. Allowed values: `0 <= x <= 0.6`

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities. Allowed values: `0 <= x <= 0.6`

Returns: Does nothing return.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. doi:10.48550/arXiv.1907.11692

See Also

Other Base Model: `BaseModelBert`, `BaseModelDebertaV2`, `BaseModelFunnel`, `BaseModelMPNet`, `BaseModelModernBert`

`BaseModelsIndex` *List of all available BaseModels*

Description

Named `list` containing all BaseModels as a string.

Usage

```
BaseModelsIndex
```

Format

An object of class `list` of length 6.

See Also

Other Parameter Dictionary: `DataSetsIndex`, `TokenizerIndex`, `get_TEClassifiers_class_names()`, `get_called_args()`, `get_depr_obj_names()`, `get_magnitude_values()`, `get_param_def()`, `get_param_dict()`, `get_param_doc_desc()`

`build_documentation_for_model` *Generate documentation for a classifier class*

Description

Function for generating the documentation of a model.

Usage

```
build_documentation_for_model(
  model_name,
  cls_type = NULL,
  core_type = NULL,
  input_type = "text_embeddings"
)
```

Arguments

model_name	string Name of the model.
cls_type	string Type of classification
core_type	string Name of the core type.
input_type	bool Name of the input type necessary for training and predicting.

Value

Returns a string containing the description written in rmarkdown.

Note

Function is designed to be used with roxygen2 in the regular documentation.

See Also

Other Utils Documentation: build_layer_stack_documentation_for_vignette(), get_desc_for_core_model_archit
get_dict_cls_type(), get_dict_core_models(), get_dict_input_types(), get_layer_dict(),
get_layer_documentation(), get_parameter_documentation()

---

build_layer_stack_documentation_for_vignette

Generate documentation of all layers for an vignette or article

---

Description

Function for generating the whole documentation for an article used on the packages home page.

Usage

```
build_layer_stack_documentation_for_vignette()
```

Value

Returns a string containing the description written in rmarkdown.

Note

Function is designed to be used with inline r code in rmarkdown vignettes/articles.

See Also

Other Utils Documentation: build_documentation_for_model(), get_desc_for_core_model_architecture(),
get_dict_cls_type(), get_dict_core_models(), get_dict_input_types(), get_layer_dict(),
get_layer_documentation(), get_parameter_documentation()

calc_standard_classification_measures

Calculate recall, precision, and f1-scores

Description

Function for calculating recall, precision, and f1-scores.

Usage

```
calc_standard_classification_measures(true_values, predicted_values)
```

Arguments

true_values	factor containing the true labels/categories.
predicted_values	factor containing the predicted labels/categories.

Value

Returns a matrix which contains the cases categories in the rows and the measures (precision, recall, f1) in the columns.

See Also

Other performance measures: cohens_kappa(), fleiss_kappa(), get_coder_metrics(), gwet_ac(), kendalls_w(), kripp_alpha()

calc_tokenizer_statistics

Estimate tokenizer statistics

Description

Function for estimating the tokenizer statistics described by Kaya & Tantuğ (2024).

Usage

```
calc_tokenizer_statistics(
  dataset,
  step = "creation",
  statistics_max_tokens_length = 512L
)
```

Arguments

dataset	Object of class datasets.arrow_dataset.Dataset. The data set must contain a column "length" containing the number of tokens for every sequence and a column "word_ids" containing the word ids within every sequence.
step	string indicating to which step the statistics belong. Recommended values are • "creation" for the creation of the tokenizer. • "initial_training" for the first training of the transformer. • "fine_tuning" for all following trainings of the transformer. • "training" for a training run of the transformer.
statistics_max_tokens_length	int Maximum sequence length for calculating the statistics. Allowed values: 20 <= x <= 8192

Value

Returns a list with the following entries:

- n_sequences: Number of sequences
- n_words: Number for words in whole corpus
- n_tokens: Number of tokens in the whole corpus
- mu_t: eqn(n_tokens/n_sequences)
- mu_w: eqn(n_words/n_sequences)
- mu_g: eqn(n_tokens/n_words)

References

Kaya, Y. B., & Tantuğ, A. C. (2024). Effect of tokenization granularity for Turkish large language models. Intelligent Systems with Applications, 21, 200335. https://doi.org/10.1016/j.iswa.2024.200335

---

cat_message *Print message (cat())*

Description

Prints a message msg if trace parameter is TRUE with current date with cat() function.

Usage

```
cat_message(msg, trace)
```

Arguments

msg	string Message that should be printed.
trace	bool Silent printing (FALSE) or not (TRUE).

Value

This function returns nothing.

See Also

Other Utils Log Developers: clean_pytorch_log_transformers(), output_message(), print_message(), read_log(), read_loss_log(), reset_log(), reset_loss_log(), write_log()

`check_adjust_n_samples_on_CI` *Set sample size for argument combinations*

Description

Function adjust the number of samples depending on the test environment. On continuous integration it is limited to a random sample of combinations.

Usage

```
check_adjust_n_samples_on_CI(n_samples_requested, n_CI = 50L)
```

Arguments

`n_samples_requested`	`int` Number of samples if the test do not run on continuous integration.
`n_CI`	`int` Number of samples if the test run on continuous integration.

Value

Returns an `int` depending on the test environment.

See Also

Other Utils TestThat Developers: generate_args_for_tests(), generate_embeddings(), generate_tensors(), get_current_args_for_print(), get_fixed_test_tensor(), get_test_data_for_classifiers(), random_bool_on_CI()

check_aif_py_modules *Check if all necessary python modules are available*

Description

This function checks if all python modules necessary for the package 'aifeducation' to work are available.

Usage

```
check_aif_py_modules(trace = TRUE)
```

Arguments

trace	bool TRUE if a list with all modules and their availability should be printed to the console.

Value

The function prints a table with all relevant packages and shows which modules are available or unavailable.

If all relevant modules are available, the functions returns TRUE. In all other cases it returns FALSE

See Also

Other Installation and Configuration: get_recommended_py_versions(), install_aifeducation(), install_aifeducation_studio(), install_py_modules(), prepare_session(), set_transformers_logger(), update_aifeducation()

check_all_args *Check arguments automatically*

Description

This function performs checks for every provided argument. It can only check arguments that are defined in the central parameter dictionary. See get_param_dict for more details.

Usage

```
check_all_args(args)
```

Arguments

args	Named list containing the arguments and their values.

Value

Function does nothing return. It raises an error the arguments are not valid.

See Also

Other Utils Checks Developers: `check_class_and_type()`

`check_class_and_type` *Check class and type*

Description

Function for checking if an object is of a specific type or class.

Usage

```
check_class_and_type(
  object,
  object_name = NULL,
  type_classes = "bool",
  allow_NULL = FALSE,
  min = NULL,
  max = NULL,
  allowed_values = NULL
)
```

Arguments

`object`	Any R object.
`object_name`	`string` Name of the object. This is helpful for debugging.
`type_classes`	`vector` of `strings` containing the type or classes which the object should belong to.
`allow_NULL`	`bool` If `TRUE` allow the object to be `NULL`.
`min`	`double` or `int` Minimal value for the object.
`max`	`double` or `int` Maximal value for the object.
`allowed_values`	`vector` of `strings` determining the allowed values. If all strings are allowed set this argument to `NULL`.

Value

Function does nothing return. It raises an error if the object is not of the specified type.

Note

parameter `min`, `max`, and `allowed_values` do not apply if `type_classes` is a class.

allowed_values does only apply if `type_classes` is `string`.

See Also

Other Utils Checks Developers: `check_all_args()`

`ClassifiersBasedOnTextEmbeddings`

Abstract class for all classifiers that use numerical representations of texts instead of words.

Description

Base class for classifiers relying on EmbeddedText or LargeDataSetForTextEmbeddings generated with a TextEmbeddingModel.

Objects of this class containing fields and methods used in several other classes in 'AI for Education'.

This class is **not** designed for a direct application and should only be used by developers.

Value

A new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `ClassifiersBasedOnTextEmbeddings`

Public fields

`feature_extractor` ('list()')
: List for storing information and objects about the feature_extractor.

`reliability` ('list()')

List for storing central reliability measures of the last training.

- `reliability$test_metric`: Array containing the reliability measures for the test data for every fold and step (in case of pseudo-labeling).
- `reliability$test_metric_mean`: Array containing the reliability measures for the test data. The values represent the mean values for every fold.
- `reliability$raw_iota_objects`: List containing all iota_object generated with the package `iotarelr` for every fold at the end of the last training for the test data.
- `reliability$raw_iota_objects$iota_objects_end`: List of objects with class `iotarelr_iota2` containing the estimated iota reliability of the second generation for the final model for every fold for the test data.
- `reliability$raw_iota_objects$iota_objects_end_free`: List of objects with class `iotarelr_iota2` containing the estimated iota reliability of the second generation for the final model for every fold for the test data. Please note that the model is estimated without forcing the Assignment Error Matrix to be in line with the assumption of weak superiority.

- `reliability$iota_object_end`: Object of class `iotarelr_iota2` as a mean of the individual objects for every fold for the test data.
- `reliability$iota_object_end_free`: Object of class `iotarelr_iota2` as a mean of the individual objects for every fold. Please note that the model is estimated without forcing the Assignment Error Matrix to be in line with the assumption of weak superiority.
- `reliability$standard_measures_end`: Object of class `list` containing the final measures for precision, recall, and f1 for every fold.
- `reliability$standard_measures_mean`: `matrix` containing the mean measures for precision, recall, and f1.

Methods

Public methods:

- `ClassifiersBasedOnTextEmbeddings$predict()`
- `ClassifiersBasedOnTextEmbeddings$check_embedding_model()`
- `ClassifiersBasedOnTextEmbeddings$check_feature_extractor_object_type()`
- `ClassifiersBasedOnTextEmbeddings$requires_compression()`
- `ClassifiersBasedOnTextEmbeddings$save()`
- `ClassifiersBasedOnTextEmbeddings$load_from_disk()`
- `ClassifiersBasedOnTextEmbeddings$adjust_target_levels()`
- `ClassifiersBasedOnTextEmbeddings$plot_training_history()`
- `ClassifiersBasedOnTextEmbeddings$plot_coding_stream()`
- `ClassifiersBasedOnTextEmbeddings$clone()`

Method `predict()`**:** Method for predicting new data with a trained neural net.

Usage:

```
ClassifiersBasedOnTextEmbeddings$predict(
  newdata,
  batch_size = 32L,
  ml_trace = 1L
)
```

Arguments:

`newdata` Object of class TextEmbeddingModel or LargeDataSetForTextEmbeddings for which predictions should be made. In addition, this method allows to use objects of class `array` and `datasets.arrow_dataset.Dataset`. However, these should be used only by developers.

`batch_size` `int` Size of batches.

`ml_trace` `int` `ml_trace=0` does not print any information on the process from the machine learning framework.

Returns: Returns a `data.frame` containing the predictions and the probabilities of the different labels for each case.

Method `check_embedding_model()`**:** Method for checking if the provided text embeddings are created with the same TextEmbeddingModel as the classifier.

Usage:

```
ClassifiersBasedOnTextEmbeddings$check_embedding_model(
  text_embeddings,
  require_compressed = FALSE
)
```

Arguments:

`text_embeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`require_compressed` `TRUE` if a compressed version of the embeddings are necessary. Compressed embeddings are created by an object of class TEFeatureExtractor.

Returns: `TRUE` if the underlying TextEmbeddingModel is the same. `FALSE` if the models differ.

Method `check_feature_extractor_object_type()`**:** Method for checking an object of class TEFeatureExtractor.

Usage:

```
ClassifiersBasedOnTextEmbeddings$check_feature_extractor_object_type(
  feature_extractor
)
```

Arguments:

`feature_extractor` Object of class TEFeatureExtractor

Returns: This method does nothing returns. It raises an error if

- the object is `NULL`
- the object does not rely on the same machine learning framework as the classifier
- the object is not trained.

Method `requires_compression()`**:** Method for checking if provided text embeddings must be compressed via a TEFeatureExtractor before processing.

Usage:

```
ClassifiersBasedOnTextEmbeddings$requires_compression(text_embeddings)
```

Arguments:

`text_embeddings` Object of class EmbeddedText, LargeDataSetForTextEmbeddings, `array` or `datasets.arrow_dataset.Dataset`.

Returns: Return `TRUE` if a compression is necessary and `FALSE` if not.

Method `save()`**:** Method for saving a model.

Usage:

```
ClassifiersBasedOnTextEmbeddings$save(dir_path, folder_name)
```

Arguments:

`dir_path` `string` Path of the directory where the model should be saved.

`folder_name` `string` Name of the folder that should be created within the directory.

Returns: Function does not return a value. It saves the model to disk.

Method `load_from_disk()`**:** loads an object from disk and updates the object to the current version of the package.

Usage:

```
ClassifiersBasedOnTextEmbeddings$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the object set is stored.

Returns: Method does not return anything. It loads an object from disk.

Method `adjust_target_levels()`**:** Method transforms the levels of a factor into numbers corresponding to the models definition.

Usage:

```
ClassifiersBasedOnTextEmbeddings$adjust_target_levels(data_targets)
```

Arguments:

`data_targets` `factor` containing the labels for cases stored in embeddings. Factor must be named and has to use the same names as used in in the embeddings.

Returns: Method returns a `factor` containing the numerical representation of categories/classes.

Method `plot_training_history()`**:** Method for requesting a plot of the training history. This method requires the *R* package 'ggplot2' to work.

Usage:

```
ClassifiersBasedOnTextEmbeddings$plot_training_history(
  final_training = FALSE,
  pl_step = NULL,
  measure = "loss",
  y_min = NULL,
  y_max = NULL,
  add_min_max = TRUE,
  text_size = 10L
)
```

Arguments:

`final_training` `bool` If `FALSE` the values of the performance estimation are used. If `TRUE` only the epochs of the final training are used.

`pl_step` `int` Number of the step during pseudo labeling to plot. Only relevant if the model was trained with active pseudo labeling.

`measure` `string` Measure to plot. Allowed values:

- `"avg_iota"` = Average Iota
- `"loss"` = Loss
- `"accuracy"` = Accuracy
- `"balanced_accuracy"` = Balanced Accuracy

`y_min` Minimal value for the y-axis. Set to `NULL` for an automatic adjustment.

`y_max` Maximal value for the y-axis. Set to `NULL` for an automatic adjustment.

`add_min_max` `bool` If `TRUE` the minimal and maximal values during performance estimation are port of the plot. If `FALSE` only the mean values are shown. Parameter is ignored if `final_training=TRUE`.

`text_size` Size of the text.

Returns: Returns a plot of class `ggplot` visualizing the training process.

Method `plot_coding_stream()`**:** Method for requesting a plot the coding stream. The plot shows how the cases of different categories/classes are assigned to a the available classes/categories. The visualization is helpful for analyzing the consequences of coding errors.

Usage:

```
ClassifiersBasedOnTextEmbeddings$plot_coding_stream(
  label_categories_size = 3L,
  key_size = 0.5,
  text_size = 10L
)
```

Arguments:

`label_categories_size` `double` determining the size of the label for each true and assigned category within the plot.

`key_size` `double` determining the size of the legend.

`text_size` `double` determining the size of the text within the legend.

Returns: Returns a plot of class `ggplot` visualizing the training process.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `BaseModelCore`, `DataManagerClassifier`, `LargeDataSetBase`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnProtoNet`, `TEClassifiersBasedOnRegula`, `TokenizerBase`

`class_vector_to_py_dataset` *Convert class vector to arrow data set*

Description

Function converts a vector of class indices into an arrow data set.

Usage

```
class_vector_to_py_dataset(vector)
```

Arguments

`vector`	`vector` of class indices.

Value

Returns a data set of class `datasets.arrow_dataset.Dataset` containing the class indices.

See Also

Other Utils Python Data Management Developers: `data.frame_to_py_dataset()`, `get_batches_index()`, `prepare_r_array_for_dataset()`, `py_dataset_to_embeddings()`, `reduce_to_unique()`, `tensor_list_to_numpy()`, `tensor_to_numpy()`

---

clean_pytorch_log_transformers

Clean pytorch log of transformers

Description

Function for preparing and cleaning the log created by an object of class Trainer from the python library 'transformer's.

Usage

```
clean_pytorch_log_transformers(log)
```

Arguments

`log`	`data.frame` containing the log.

Value

Returns a `data.frame` containing epochs, loss, and val_loss.

See Also

Other Utils Log Developers: `cat_message()`, `output_message()`, `print_message()`, `read_log()`, `read_loss_log()`, `reset_log()`, `reset_loss_log()`, `write_log()`

cohens_kappa *Calculate Cohen's Kappa*

Description

This function calculates different version of Cohen's Kappa.

Usage

```
cohens_kappa(rater_one, rater_two)
```

Arguments

rater_one	factor rating of the first coder.
rater_two	factor ratings of the second coder.

Value

Returns a list containing the results for Cohen' Kappa if no weights are applied (kappa_unweighted), if weights are applied and the weights increase linear (kappa_linear), and if weights are applied and the weights increase quadratic (kappa_squared).

References

Cohen, J (1968). Weighted kappa: Nominal scale agreement with provision for scaled disagreement or partial credit. Psychological Bulletin, 70(4), 213–220. doi:10.1037/h0026256

Cohen, J (1960). A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement, 20(1), 37–46. doi:10.1177/001316446002000104

See Also

Other performance measures: calc_standard_classification_measures(), fleiss_kappa(), get_coder_metrics(), gwet_ac(), kendalls_w(), kripp_alpha()

create_dir *Create directory if not exists*

Description

Check whether the passed dir_path directory exists. If not, creates a new directory and prints a msg message if trace is TRUE.

Usage

```
create_dir(dir_path, trace, msg = "Creating Directory", msg_fun = TRUE)
```

Arguments

dir_path	string A new directory path that should be created.
trace	bool Whether a msg message should be printed.
msg	string A message that should be printed if trace is TRUE.
msg_fun	func Function used for printing the message.

Value

TRUE or FALSE depending on whether the shiny app is active.

See Also

Other Utils File Management Developers: get_file_extension()

---

create_object *Create object#'*

Description

Support function for creating objects.

Usage

```
create_object(class)
```

Arguments

class	string Name of the class to be created.

Value

Returns an object of the requested class.

See Also

Other Utils Developers: auto_n_cores(), create_synthetic_units_from_matrix(), generate_id(), get_n_chunks(), get_synthetic_cases_from_matrix(), get_time_stamp(), matrix_to_array_c(), tensor_to_matrix_c(), to_categorical_c()

`create_synthetic_units_from_matrix` *Create synthetic units*

Description

Function for creating synthetic cases in order to balance the data for training with TEClassifierRegular or TEClassifierProtoNet]. This is an auxiliary function for use with get_synthetic_cases_from_matrix to allow parallel computations.

Usage

```
create_synthetic_units_from_matrix(
  matrix_form,
  target,
  required_cases,
  k,
  method,
  cat,
  k_s,
  max_k
)
```

Arguments

`matrix_form`	Named `matrix` containing the text embeddings in matrix form. In most cases this object is taken from EmbeddedText$embeddings.
`target`	Named `factor` containing the labels/categories of the corresponding cases.
`required_cases`	`int` Number of cases necessary to fill the gab between the frequency of the class under investigation and the major class.
`k`	`int` The number of nearest neighbors during sampling process.
`method`	`vector` containing strings of the requested methods for generating new cases. Currently "knnor" from this package is available.
`cat`	`string` The category for which new cases should be created.
`k_s`	`int` Number of ks in the complete generation process.
`max_k`	`int` The maximum number of nearest neighbors during sampling process.

Value

Returns a `list` which contains the text embeddings of the new synthetic cases as a named `data.frame` and their labels as a named `factor`.

See Also

Other Utils Developers: auto_n_cores(), create_object(), generate_id(), get_n_chunks(), get_synthetic_cases_from_matrix(), get_time_stamp(), matrix_to_array_c(), tensor_to_matrix_c(), to_categorical_c()

---

data.frame_to_py_dataset

Convert data.frame to arrow data set

Description

Function for converting a data.frame into a pyarrow data set.

Usage

```
data.frame_to_py_dataset(data_frame)
```

Arguments

data_frame	Object of class data.frame.

Value

Returns the data.frame as a pyarrow data set of class datasets.arrow_dataset.Dataset.

See Also

Other Utils Python Data Management Developers: class_vector_to_py_dataset(), get_batches_index(), prepare_r_array_for_dataset(), py_dataset_to_embeddings(), reduce_to_unique(), tensor_list_to_numpy(), tensor_to_numpy()

DataManagerClassifier *Data manager for classification tasks*

Description

Abstract class for managing the data and samples during training a classifier. DataManagerClassifier is used with all classifiers based on text embeddings.

Value

Objects of this class are used for ensuring the correct data management for training different types of classifiers. They are also used for data augmentation by creating synthetic cases with different techniques.

Public fields

`config` ('list')
: Field for storing configuration of the DataManagerClassifier.

`state` ('list')
: Field for storing the current state of the DataManagerClassifier.

`datasets` ('list')
: Field for storing the data sets used during training. All elements of the list are data sets of class `datasets.arrow_dataset.Dataset`. The following data sets are available:
  - data_labeled: all cases which have a label.
  - data_unlabeled: all cases which have no label.
  - data_labeled_synthetic: all synthetic cases with their corresponding labels.
  - data_labeled_pseudo: subset of data_unlabeled if pseudo labels were estimated by a classifier.

`name_idx` ('named vector')
: Field for storing the pairs of indexes and names of every case. The pairs for labeled and unlabeled data are separated.

`samples` ('list')
: Field for storing the assignment of every cases to a train, validation or test data set depending on the concrete fold. Only the indexes and not the names are stored. In addition, the list contains the assignment for the final training which excludes a test data set. If the DataManagerClassifier uses `i` folds the sample for the final training can be requested with `i+1`.

Methods

Public methods:

Method `new()`**:** Creating a new instance of this class.

Usage:

```
DataManagerClassifier$new(
  data_embeddings,
  data_targets,
  class_levels,
  folds = 5L,
  val_size = 0.25,
  pad_value = -100L,
  one_hot_encoding = TRUE,
  add_matrix_map = TRUE,
  sc_methods = "knnor",
  sc_min_k = 1L,
  sc_max_k = 10L,
  trace = TRUE,
  n_cores = auto_n_cores()
)
```

Arguments:

`data_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`data_targets` `factor` containing the labels for cases stored in embeddings. Factor must be named and has to use the same names as used in in the embeddings.

`class_levels` `vector` containing the levels (categories or classes) within the target data. Please note that order matters. For ordinal data please ensure that the levels are sorted correctly with later levels indicating a higher category/class. For nominal data the order does not matter.

`folds` `int` determining the number of cross-fold samples. Allowed values: `1 <= x`

`val_size` `double` between 0 and 1, indicating the proportion of cases which should be used for the validation sample during the estimation of the model. The remaining cases are part of the training data. Allowed values: `0 < x < 1`

`pad_value` `int` Value indicating padding. This value should no be in the range of regluar values for computations. Thus it is not recommended to chance this value. Default is `-100`. Allowed values: `x <= -100`

`one_hot_encoding` `bool` If `TRUE` all labels are converted to one hot encoding.

`add_matrix_map` `bool` If `TRUE` all embeddings are transformed into a two dimensional matrix. The number of rows equals the number of cases. The number of columns equals `times*features`.

`sc_methods` `string` containing the method for generating synthetic cases. Allowed values: 'knnor'

`sc_min_k` `int` determining the minimal number of k which is used for creating synthetic units. Allowed values: `1 <= x`

`sc_max_k` `int` determining the maximal number of k which is used for creating synthetic units. Allowed values: `1 <= x`

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

`n_cores` `int` Number of cores which should be used during the calculation of synthetic cases. Only relevant if `use_sc=TRUE`. Allowed values: `1 <= x`

Returns: Method returns an initialized object of class DataManagerClassifier.

Method `get_config()`**:** Method for requesting the configuration of the DataManagerClassifier.

Usage:

```
DataManagerClassifier$get_config()
```

Returns: Returns a `list` storing the configuration of the DataManagerClassifier.

Method `get_labeled_data()`**:** Method for requesting the complete labeled data set.

Usage:

```
DataManagerClassifier$get_labeled_data()
```

Returns: Returns an object of class `datasets.arrow_dataset.Dataset` containing all cases with labels.

Method `get_unlabeled_data()`**:** Method for requesting the complete unlabeled data set.

Usage:

```
DataManagerClassifier$get_unlabeled_data()
```

Returns: Returns an object of class `datasets.arrow_dataset.Dataset` containing all cases without labels.

Method `get_samples()`**:** Method for requesting the assignments to train, validation, and test data sets for every fold and the final training.

Usage:

```
DataManagerClassifier$get_samples()
```

Returns: Returns a `list` storing the assignments to a train, validation, and test data set for every fold. In the case of the sample for the final training the test data set is always empty (`NULL`).

Method `set_state()`**:** Method for setting the current state of the DataManagerClassifier.

Usage:

```
DataManagerClassifier$set_state(iteration, step = NULL)
```

Arguments:

`iteration` `int` determining the current iteration of the training. That is iteration determines the fold to use for training, validation, and testing. If *i* is the number of fold *i+1* request the sample for the final training. For requesting the sample for the final training iteration can take a string `"final"`.

`step` `int` determining the step for estimating and using pseudo labels during training. Only relevant if training is requested with pseudo labels.

Returns: Method does not return anything. It is used for setting the internal state of the DataManager.

Method `get_n_folds()`**:** Method for requesting the number of folds the DataManagerClassifier can use with the current data.

Usage:

```
DataManagerClassifier$get_n_folds()
```

Returns: Returns the number of folds the DataManagerClassifier uses.

Method `get_n_classes()`**:** Method for requesting the number of classes.

Usage:

```
DataManagerClassifier$get_n_classes()
```

Returns: Returns the number classes.

Method `get_statistics()`**:** Method for requesting descriptive sample statistics.

Usage:

```
DataManagerClassifier$get_statistics()
```

Returns: Returns a table describing the absolute frequencies of the labeled and unlabeled data. The rows contain the length of the sequences while the columns contain the labels.

Method `contains_unlabeled_data()`**:** Method for checking if the dataset contains cases without labels.

Usage:

```
DataManagerClassifier$contains_unlabeled_data()
```

Returns: Returns `TRUE` if the dataset contains cases without labels. Returns `FALSE` if all cases have labels.

Method `get_dataset()`**:** Method for requesting a data set for training depending in the current state of the DataManagerClassifier.

Usage:

```
DataManagerClassifier$get_dataset(
  inc_labeled = TRUE,
  inc_unlabeled = FALSE,
  inc_synthetic = FALSE,
  inc_pseudo_data = FALSE
)
```

Arguments:

`inc_labeled bool` If `TRUE` the data set includes all cases which have labels.

`inc_unlabeled bool` If `TRUE` the data set includes all cases which have no labels.

`inc_synthetic bool` If `TRUE` the data set includes all synthetic cases with their corresponding labels.

`inc_pseudo_data bool` If `TRUE` the data set includes all cases which have pseudo labels.

Returns: Returns an object of class `datasets.arrow_dataset.Dataset` containing the requested kind of data along with all requested transformations for training. Please note that this method returns a data sets that is designed for training only. The corresponding validation data set is requested with `get_val_dataset` and the corresponding test data set with `get_test_dataset`.

Method `get_val_dataset()`**:** Method for requesting a data set for validation depending in the current state of the DataManagerClassifier.

Usage:

```
DataManagerClassifier$get_val_dataset()
```

Returns: Returns an object of class `datasets.arrow_dataset.Dataset` containing the requested kind of data along with all requested transformations for validation. The corresponding data set for training can be requested with `get_dataset` and the corresponding data set for testing with `get_test_dataset`.

Method `get_test_dataset()`**:** Method for requesting a data set for testing depending in the current state of the DataManagerClassifier.

Usage:

```
DataManagerClassifier$get_test_dataset()
```

Returns: Returns an object of class `datasets.arrow_dataset.Dataset` containing the requested kind of data along with all requested transformations for validation. The corresponding data set for training can be requested with `get_dataset` and the corresponding data set for validation with `get_val_dataset`.

Method `create_synthetic()`**:** Method for generating synthetic data used during training. The process uses all labeled data belonging to the current state of the DataManagerClassifier.

Usage:

```
DataManagerClassifier$create_synthetic(trace = TRUE, inc_pseudo_data = FALSE)
```

Arguments:

`trace` `bool` If `TRUE` information on the process are printed to the console.

`inc_pseudo_data` `bool` If `TRUE` data with pseudo labels are used in addition to the labeled data for generating synthetic cases.

Returns: This method does nothing return. It generates a new data set for synthetic cases which are stored as an object of class `datasets.arrow_dataset.Dataset` in the field `datasets$data_labeled_synthetic`. Please note that a call of this method will override an existing data set in the corresponding field.

Method `add_replace_pseudo_data()`**:** Method for adding data with pseudo labels generated by a classifier

Usage:

```
DataManagerClassifier$add_replace_pseudo_data(inputs, labels)
```

Arguments:

`inputs` `array` or `matrix` representing the input data.

`labels` `factor` containing the corresponding pseudo labels.

Returns: This method does nothing return. It generates a new data set for synthetic cases which are stored as an object of class `datasets.arrow_dataset.Dataset` in the field `datasets$data_labeled_pseudo`. Please note that a call of this method will override an existing data set in the corresponding field.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `BaseModelCore`, `ClassifiersBasedOnTextEmbeddings`, `LargeDataSetBase`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnProtoNet`, `TEClassifiersBasedOnRegula`
`TokenizerBase`

DataSetsIndex *List of all available types of data sets*

Description

Named `list` containing all available types of data sets as a string.

Usage

```
DataSetsIndex
```

Format

An object of class `list` of length 3.

See Also

Other Parameter Dictionary: `BaseModelsIndex`, `TokenizerIndex`, `get_TEClassifiers_class_names()`, `get_called_args()`, `get_depr_obj_names()`, `get_magnitude_values()`, `get_param_def()`, `get_param_dict()`, `get_param_doc_desc()`

EmbeddedText *Abstract class for small data sets containing text embeddings*

Description

Object of class `R6` which stores the text embeddings generated by an object of class TextEmbeddingModel. The text embeddings are stored within memory/RAM. In the case of a high number of documents the data may not fit into memory/RAM. Thus, please use this object only for a small sample of texts. In general, it is recommended to use an object of class LargeDataSetForTextEmbeddings which can deal with any number of texts.

Value

Returns an object of class EmbeddedText. These objects are used for storing and managing the text embeddings created with objects of class TextEmbeddingModel. Objects of class EmbeddedText serve as input for objects of class TEClassifierRegular, TEClassifierProtoNet, and TEFeatureExtractor. The main aim of this class is to provide a structured link between embedding models and classifiers. Since objects of this class save information on the text embedding model that created the text embedding it ensures that only embedding generated with same embedding model are combined. Furthermore, the stored information allows objects to check if embeddings of the correct text embedding model are used for training and predicting.

Public fields

`embeddings` ('data.frame()')
: data.frame containing the text embeddings for all chunks. Documents are in the rows. Embedding dimensions are in the columns.

Methods

Public methods:

- `EmbeddedText$configure()`
- `EmbeddedText$save()`
- `EmbeddedText$is_configured()`
- `EmbeddedText$load_from_disk()`
- `EmbeddedText$get_model_info()`
- `EmbeddedText$get_model_label()`
- `EmbeddedText$get_times()`
- `EmbeddedText$get_features()`
- `EmbeddedText$get_original_features()`
- `EmbeddedText$get_pad_value()`
- `EmbeddedText$is_compressed()`
- `EmbeddedText$add_feature_extractor_info()`
- `EmbeddedText$get_feature_extractor_info()`
- `EmbeddedText$convert_to_LargeDataSetForTextEmbeddings()`
- `EmbeddedText$n_rows()`
- `EmbeddedText$get_all_fields()`
- `EmbeddedText$set_package_versions()`
- `EmbeddedText$get_package_versions()`
- `EmbeddedText$clone()`

Method `configure()`**:** Creates a new object representing text embeddings.

Usage:

```
EmbeddedText$configure(
  embeddings,
  model_name = NA,
  model_label = NA,
  model_date = NA,
  model_method = NA,
  model_version = NA,
  model_language = NA,
  param_seq_length = NA,
  param_chunks = NULL,
  param_features = NULL,
  param_overlap = NULL,
  param_emb_layer_min = NULL,
  param_emb_layer_max = NULL,
  param_emb_pool_type = NULL,
```

```
  param_aggregation = NULL,
  param_pad_value = -100L
)
```

Arguments:

`embeddings` `data.frame` containing the text embeddings.

`model_name` `string` Name of the model that generates this embedding.

`model_label` `string` Label of the model that generates this embedding.

`model_date` `string` Date when the embedding generating model was created.

`model_method` `string` Method of the underlying embedding model.

`model_version` `string` Version of the model that generated this embedding.

`model_language` `string` Language of the model that generated this embedding.

`param_seq_length` `int` Maximum number of tokens that processes the generating model for a chunk.

`param_chunks` `int` Maximum number of chunks which are supported by the generating model.

`param_features` `int` Number of dimensions of the text embeddings.

`param_overlap` `int` Number of tokens that were added at the beginning of the sequence for the next chunk by this model. #'

`param_emb_layer_min` `int` or `string` determining the first layer to be included in the creation of embeddings.

`param_emb_layer_max` `int` or `string` determining the last layer to be included in the creation of embeddings.

`param_emb_pool_type` `string` determining the method for pooling the token embeddings within each layer.

`param_aggregation` `string` Aggregation method of the hidden states. Deprecated. Only included for backward compatibility.

`param_pad_value` `int` Value indicating padding. This value should no be in the range of regluar values for computations. Thus it is not recommended to chance this value. Default is `-100`. Allowed values: `x <= -100`

Returns: Returns an object of class EmbeddedText which stores the text embeddings produced by an objects of class TextEmbeddingModel.

Method `save()`**:** Saves a data set to disk.

Usage:

```
EmbeddedText$save(dir_path, folder_name, create_dir = TRUE)
```

Arguments:

`dir_path` Path where to store the data set.

`folder_name` `string` Name of the folder for storing the data set.

`create_dir` `bool` If `True` the directory will be created if it does not exist.

Returns: Method does not return anything. It write the data set to disk.

Method `is_configured()`**:** Method for checking if the model was successfully configured. An object can only be used if this value is `TRUE`.

Usage:

```
EmbeddedText$is_configured()
```

Returns: `bool` `TRUE` if the model is fully configured. `FALSE` if not.

Method `load_from_disk()`**:** loads an object of class EmbeddedText from disk and updates the object to the current version of the package.

Usage:

```
EmbeddedText$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the data set set is stored.

Returns: Method does not return anything. It loads an object from disk.

Method `get_model_info()`**:** Method for retrieving information about the model that generated this embedding.

Usage:

```
EmbeddedText$get_model_info()
```

Returns: `list` contains all saved information about the underlying text embedding model.

Method `get_model_label()`**:** Method for retrieving the label of the model that generated this embedding.

Usage:

```
EmbeddedText$get_model_label()
```

Returns: `string` Label of the corresponding text embedding model

Method `get_times()`**:** Number of chunks/times of the text embeddings.

Usage:

```
EmbeddedText$get_times()
```

Returns: Returns an `int` describing the number of chunks/times of the text embeddings.

Method `get_features()`**:** Number of actual features/dimensions of the text embeddings.In the case a feature extractor was used the number of features is smaller as the original number of features. To receive the original number of features (the number of features before applying a feature extractor) you can use the method `get_original_features` of this class.

Usage:

```
EmbeddedText$get_features()
```

Returns: Returns an `int` describing the number of features/dimensions of the text embeddings.

Method `get_original_features()`**:** Number of original features/dimensions of the text embeddings.

Usage:

```
EmbeddedText$get_original_features()
```

Returns: Returns an `int` describing the number of features/dimensions if no feature extractor) is used or before a feature extractor) is applied.

Method `get_pad_value()`**:** Value for indicating padding.

Usage:

```
EmbeddedText$get_pad_value()
```

Returns: Returns an `int` describing the value used for padding.

Method `is_compressed()`**:** Checks if the text embedding were reduced by a feature extractor.

Usage:

```
EmbeddedText$is_compressed()
```

Returns: Returns `TRUE` if the number of dimensions was reduced by a feature extractor. If not return `FALSE`.

Method `add_feature_extractor_info()`**:** Method setting information on the feature extractor that was used to reduce the number of dimensions of the text embeddings. This information should only be used if a feature extractor was applied.

Usage:

```
EmbeddedText$add_feature_extractor_info(
  model_name,
  model_label = NA,
  features = NA,
  method = NA,
  noise_factor = NA,
  optimizer = NA
)
```

Arguments:

`model_name` `string` Name of the underlying TextEmbeddingModel.

`model_label` `string` Label of the underlying TextEmbeddingModel.

`features` `int` Number of dimension (features) for the **compressed** text embeddings.

`method` `string` Method that the TEFeatureExtractor applies for genereating the compressed text embeddings.

`noise_factor` `double` Noise factor of the TEFeatureExtractor.

`optimizer` `string` Optimizer used during training the TEFeatureExtractor.

Returns: Method does nothing return. It sets information on a feature extractor.

Method `get_feature_extractor_info()`**:** Method for receiving information on the feature extractor that was used to reduce the number of dimensions of the text embeddings.

Usage:

```
EmbeddedText$get_feature_extractor_info()
```

Returns: Returns a `list` with information on the feature extractor. If no feature extractor was used it returns `NULL`.

Method `convert_to_LargeDataSetForTextEmbeddings()`**:** Method for converting this object to an object of class LargeDataSetForTextEmbeddings.

Usage:

```
EmbeddedText$convert_to_LargeDataSetForTextEmbeddings()
```

Returns: Returns an object of class LargeDataSetForTextEmbeddings which uses memory mapping allowing to work with large data sets.

Method `n_rows()`**:** Number of rows.

Usage:

```
EmbeddedText$n_rows()
```

Returns: Returns the number of rows of the text embeddings which represent the number of cases.

Method `get_all_fields()`**:** Return all fields.

Usage:

```
EmbeddedText$get_all_fields()
```

Returns: Method returns a `list` containing all public and private fields of the object.

Method `set_package_versions()`**:** Method for setting the package version for 'aifeducation', 'reticulate', 'torch', and 'numpy' to the currently used versions.

Usage:

```
EmbeddedText$set_package_versions()
```

Returns: Method does not return anything. It is used to set the private fields fo package versions.

Method `get_package_versions()`**:** Method for requesting a summary of the R and python packages' versions used for creating the model.

Usage:

```
EmbeddedText$get_package_versions()
```

Returns: Returns a `list` containing the versions of the relevant R and python packages.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Data Management: `LargeDataSetForText`, `LargeDataSetForTextEmbeddings`

fleiss_kappa *Calculate Fleiss' Kappa*

Description

This function calculates Fleiss' Kappa.

Usage

```
fleiss_kappa(rater_one, rater_two, additional_raters = NULL)
```

Arguments

rater_one	factor rating of the first coder.
rater_two	factor ratings of the second coder.
additional_raters	list Additional raters with same requirements as rater_one and rater_two. If there are no additional raters set to NULL.

Value

Returns the value for Fleiss' Kappa.

References

Fleiss, J. L. (1971). Measuring nominal scale agreement among many raters. Psychological Bulletin, 76(5), 378–382. doi:10.1037/h0031619

See Also

Other performance measures: calc_standard_classification_measures(), cohens_kappa(), get_coder_metrics(), gwet_ac(), kendalls_w(), kripp_alpha()

generate_args_for_tests *Generate combinations of arguments*

Description

Function generates a specific number of combinations for a method. These are used for automating tests of objects.

Usage

```
generate_args_for_tests(
  object_name,
  method,
  var_objects = list(),
  necessary_objects = list(),
 var_override = list(sustain_interval = 30L, trace = FALSE, epochs = 50L, batch_size =
  20L, ml_trace = 0L, n_cores = 2L, data_folds = 2L, pl_max_steps = 2L, pl_max = 1L,
   pl_anchor = 1L, pl_min = 0L, sustain_track = TRUE, sustain_iso_code = "DEU",
  sustain_log_level = "error", data_val_size = 0.25, lr_rate = 0.001, lr_warm_up_ratio
   = 0.01)
)
```

Arguments

object_name	string Name of the object to generate the arguments for.
method	string Name of the method of the object to generate the arguments for.
var_objects	list of other objects which should be combined with the other arguments.
necessary_objects	list of other objects which are part of every combination.
var_override	Named list containing the arguments which should be set to a specific value for all combinations.

Value

Returns a list with combinations of arguments.

Note

var_objects, necessary_objects, and var_override the names must exactly match the name of the parameter. Otherwise they are not applied. Names of arguments which are not part a a method are ignored. #'

See Also

Other Utils TestThat Developers: check_adjust_n_samples_on_CI(), generate_embeddings(), generate_tensors(), get_current_args_for_print(), get_fixed_test_tensor(), get_test_data_for_classifier(), random_bool_on_CI()

---

generate_embeddings *Generate test embeddings*

Description

Functions generates a random test embedding that can be used for testing methods and functions. The embeddings have the shape (Batch, Times,Features).

Usage

```
generate_embeddings(times, features, seq_len, pad_value)
```

Arguments

times	int Maximal length of a sequence.
features	int Number of features of the sequence.
seq_len	Numeric vector containing the length of the given cases. The length of this vector determines the value for 'Batch'. Values must be at least 1 and maximal times.
pad_value	int Value used to indicate padding.

Value

Returns an array with dim (length(seq_len),times,features).

Note

To generate a 'PyTorch' object please use generate_tensors.

See Also

Other Utils TestThat Developers: check_adjust_n_samples_on_CI(), generate_args_for_tests(), generate_tensors(), get_current_args_for_print(), get_fixed_test_tensor(), get_test_data_for_classifier(), random_bool_on_CI()

---

generate_id *Generate ID suffix for objects*

Description

Function for generating an ID suffix for objects of class TextEmbeddingModel, TEClassifierRegular, and TEClassifierProtoNet.

Usage

```
generate_id(length = 16L)
```

Arguments

length	int determining the length of the id suffix.

Value

Returns a string of the requested length.

See Also

Other Utils Developers: auto_n_cores(), create_object(), create_synthetic_units_from_matrix(), get_n_chunks(), get_synthetic_cases_from_matrix(), get_time_stamp(), matrix_to_array_c(), tensor_to_matrix_c(), to_categorical_c()

`generate_tensors` *Generate test tensors*

Description

Functions generates a random test tensor that can be used for testing methods and functions based on 'PyTorch'. The tensors have the shape (Batch, Times,Features).

Usage

```
generate_tensors(times, features, seq_len, pad_value)
```

Arguments

`times`	`int` Maximal length of a sequence.
`features`	`int` Number of features of the sequence.
`seq_len`	Numeric `vector` containing the length of the given cases. The length of this vector determines the value for 'Batch'. Values must be at least 1 and maximal `times`.
`pad_value`	`int` Value used to indicate padding.

Value

Returns an object of class `Tensor` from 'PyTorch'.

Note

To request a *R* array please use generate_embeddings.

See Also

Other Utils TestThat Developers: check_adjust_n_samples_on_CI(), generate_args_for_tests(), generate_embeddings(), get_current_args_for_print(), get_fixed_test_tensor(), get_test_data_for_classif
random_bool_on_CI()

get_alpha_3_codes *Country Alpha 3 Codes*

Description

Function for requesting a vector containing the alpha-3 codes for most countries.

Usage

```
get_alpha_3_codes()
```

Value

Returns a vector containing the alpha-3 codes for most countries.

See Also

Other Utils Sustainability Developers: summarize_tracked_sustainability()

get_batches_index *Assign cases to batches*

Description

Function groups cases into batches.

Usage

```
get_batches_index(number_rows, batch_size, zero_based = FALSE)
```

Arguments

number_rows	int representing the number of cases or rows of a matrix or array.
batch_size	int size of a batch.
zero_based	bool If TRUE the indices of the cases within each batch are zero based. One based if FALSE.

Value

Returns a list of batches. Each entry in the list contains a vector of int representing the cases belonging to that batch.

See Also

Other Utils Python Data Management Developers: class_vector_to_py_dataset(), data.frame_to_py_dataset(), prepare_r_array_for_dataset(), py_dataset_to_embeddings(), reduce_to_unique(), tensor_list_to_numpy(), tensor_to_numpy()

get_called_args *Called arguments*

Description

Function for receiving all arguments that were called by a method or function.

Usage

```
get_called_args(n = 1L)
```

Arguments

n	int level of the nested environments where to extract the arguments.

Value

Returns a named list of all arguments and their values.

See Also

Other Parameter Dictionary: BaseModelsIndex, DataSetsIndex, TokenizerIndex, get_TEClassifiers_class_names(), get_depr_obj_names(), get_magnitude_values(), get_param_def(), get_param_dict(), get_param_doc_desc()

get_coder_metrics *Calculate reliability measures based on content analysis*

Description

This function calculates different reliability measures which are based on the empirical research method of content analysis.

Usage

```
get_coder_metrics(
  true_values = NULL,
  predicted_values = NULL,
  return_names_only = FALSE
)
```

Arguments

true_values	factor containing the true labels/categories.
predicted_values	factor containing the predicted labels/categories.
return_names_only	bool If TRUE returns only the names of the resulting vector. Use FALSE to request computation of the values.

Value

If `return_names_only = FALSE` returns a `vector` with the following reliability measures:

- **iota_index**: Iota Index from the Iota Reliability Concept Version 2.
- **min_iota2**: Minimal Iota from Iota Reliability Concept Version 2.
- **avg_iota2**: Average Iota from Iota Reliability Concept Version 2.
- **max_iota2**: Maximum Iota from Iota Reliability Concept Version 2.
- **min_alpha**: Minmal Alpha Reliability from Iota Reliability Concept Version 2.
- **avg_alpha**: Average Alpha Reliability from Iota Reliability Concept Version 2.
- **max_alpha**: Maximum Alpha Reliability from Iota Reliability Concept Version 2.
- **static_iota_index**: Static Iota Index from Iota Reliability Concept Version 2.
- **dynamic_iota_index**: Dynamic Iota Index Iota Reliability Concept Version 2.
- **kalpha_nominal**: Krippendorff's Alpha for nominal variables.
- **kalpha_ordinal**: Krippendorff's Alpha for ordinal variables.
- **kendall**: Kendall's coefficient of concordance W with correction for ties.
- **c_kappa_unweighted**: Cohen's Kappa unweighted.
- **c_kappa_linear**: Weighted Cohen's Kappa with linear increasing weights.
- **c_kappa_squared**: Weighted Cohen's Kappa with quadratic increasing weights.
- **kappa_fleiss**: Fleiss' Kappa for multiple raters without exact estimation.
- **percentage_agreement**: Percentage Agreement.
- **balanced_accuracy**: Average accuracy within each class.
- **gwet_ac1_nominal**: Gwet's Agreement Coefficient 1 (AC1) for nominal data which is unweighted.
- **gwet_ac2_linear**: Gwet's Agreement Coefficient 2 (AC2) for ordinal data with linear weights.
- **gwet_ac2_quadratic**: Gwet's Agreement Coefficient 2 (AC2) for ordinal data with quadratic weights.

If `return_names_only = TRUE` returns only the names of the vector elements.

See Also

Other performance measures: `calc_standard_classification_measures()`, `cohens_kappa()`, `fleiss_kappa()`, `gwet_ac()`, `kendalls_w()`, `kripp_alpha()`

get_current_args_for_print *Print arguments*

Description

Functions prints the used arguments. The aim of this function is to print the arguments to the console that resulted in a failed test.

Usage

```
get_current_args_for_print(arg_list)
```

Arguments

arg_list	Named list of arguments. The list should be generated with generate_args_for_tests.

Value

Function does nothing return.

See Also

Other Utils TestThat Developers: check_adjust_n_samples_on_CI(), generate_args_for_tests(), generate_embeddings(), generate_tensors(), get_fixed_test_tensor(), get_test_data_for_classifiers(), random_bool_on_CI()

get_depr_obj_names *Get names of deprecated objects*

Description

Function returns the names of all objects that are deprecated.

Usage

```
get_depr_obj_names()
```

Value

Returns a vector containing the names.

See Also

Other Parameter Dictionary: BaseModelsIndex, DataSetsIndex, TokenizerIndex, get_TEClassifiers_class_names(), get_called_args(), get_magnitude_values(), get_param_def(), get_param_dict(), get_param_doc_desc()

get_desc_for_core_model_architecture *Generate documentation for core models*

Description

Function for generating the documentation of a specific core model.

Usage

```
get_desc_for_core_model_architecture(
  name,
  title_format = "bold",
  inc_img = FALSE
)
```

Arguments

name	string Name of the core model.
title_format	string Kind of format of the title.
inc_img	bool Include a visualization of the layer.

Value

Returns a string containing the description written in rmarkdown.

See Also

Other Utils Documentation: build_documentation_for_model(), build_layer_stack_documentation_for_vignette(
get_dict_cls_type(), get_dict_core_models(), get_dict_input_types(), get_layer_dict(),
get_layer_documentation(), get_parameter_documentation()

get_file_extension *Get file extension*

Description

Function for requesting the file extension

Usage

```
get_file_extension(file_path)
```

Arguments

file_path	string Path to a file.

Value

Returns the extension of a file as a string.

See Also

Other Utils File Management Developers: `create_dir()`

`get_fixed_test_tensor` *Generate static test tensor*

Description

Function generates a static test tensor which is always the same.

Usage

```
get_fixed_test_tensor(pad_value)
```

Arguments

`pad_value`	`int` Value used to indicate padding.

Value

Returns an object of class `Tensor` which is always the same except padding. Shape (5,3,7).

See Also

Other Utils TestThat Developers: `check_adjust_n_samples_on_CI()`, `generate_args_for_tests()`, `generate_embeddings()`, `generate_tensors()`, `get_current_args_for_print()`, `get_test_data_for_classifiers(`
`random_bool_on_CI()`

`get_layer_documentation`
Generate layer documentation

Description

Function for generating the documentation of a specific layer.

Usage

```
get_layer_documentation(
  layer_name,
  title_format = "bold",
  subtitle_format = "italic",
  inc_img = FALSE,
  inc_params = FALSE,
  inc_references = FALSE
)
```

Arguments

layer_name	string Name of the layer.
title_format	string Kind of format of the title.
subtitle_format	string Kind of format for all sub-titles.
inc_img	bool Include a visualization of the layer.
inc_params	bool Include a description of every parameter of the layer.
inc_references	bool Include a list of literature references for the layer.

Value

Returns a string containing the description written in rmarkdown.

See Also

Other Utils Documentation: build_documentation_for_model(), build_layer_stack_documentation_for_vignette(
get_desc_for_core_model_architecture(), get_dict_cls_type(), get_dict_core_models(),
get_dict_input_types(), get_layer_dict(), get_parameter_documentation()

---

get_magnitude_values *Magnitudes of an argument*

Description

Function calculates different magnitude for a numeric argument.

Usage

```
get_magnitude_values(magnitude, n_elements = 9L, max = NULL, min = NULL)
```

Arguments

magnitude	double Factor using for creating the magnitude.
n_elements	int Number of values to return.
max	double The maximal value.
min	double The minimal value.

Value

Returns a numeric vector with the generated values. The values are calculated with the following formula: max * magnitude^i for i=1,...,n_elements. Only values equal or greater min are returned.

See Also

Other Parameter Dictionary: BaseModelsIndex, DataSetsIndex, TokenizerIndex, get_TEClassifiers_class_names(), get_called_args(), get_depr_obj_names(), get_param_def(), get_param_dict(), get_param_doc_desc()

---

get_n_chunks *Get the number of chunks/sequences for each case*

Description

Function for calculating the number of chunks/sequences for every case.

Usage

```
get_n_chunks(text_embeddings, features, times, pad_value = -100L)
```

Arguments

text_embeddings	data.frame or array containing the text embeddings.
features	int Number of features within each sequence.
times	int Number of sequences.
pad_value	int Value indicating padding. This value should no be in the range of regluar values for computations. Thus it is not recommended to chance this value. Default is -100. Allowed values: x <= -100

Value

Namedvector of integers representing the number of chunks/sequences for every case.

See Also

Other Utils Developers: auto_n_cores(), create_object(), create_synthetic_units_from_matrix(), generate_id(), get_synthetic_cases_from_matrix(), get_time_stamp(), matrix_to_array_c(), tensor_to_matrix_c(), to_categorical_c()

get_parameter_documentation *Generate layer documentation*

Description

Function for generating the documentation of a specific layer.

Usage

```
get_parameter_documentation(
  param_name,
  param_dict,
  as_list = TRUE,
  inc_param_name = TRUE
)
```

Arguments

param_name	string Name of the parameter.
param_dict	list storing the parameter description.
as_list	bool If TRUE returns the element as part of a list.
inc_param_name	bool If TRUE the documentation includes the name of the parameter.

Value

Returns a string containing the description written in rmarkdown.

See Also

Other Utils Documentation: build_documentation_for_model(), build_layer_stack_documentation_for_vignette(
get_desc_for_core_model_architecture(), get_dict_cls_type(), get_dict_core_models(),
get_dict_input_types(), get_layer_dict(), get_layer_documentation()

get_param_def *Definition of an argument*

Description

Function returns the definition of an argument. Please note that only definitions of arguments can be requested which are used for transformers or classifier models.

Usage

```
get_param_def(param_name)
```

Arguments

param_name	string Name of the parameter to request its definition.

Value

Returns a list with the definition of the argument. See get_param_dict for more details.

See Also

Other Parameter Dictionary: BaseModelsIndex, DataSetsIndex, TokenizerIndex, get_TEClassifiers_class_names(), get_called_args(), get_depr_obj_names(), get_magnitude_values(), get_param_dict(), get_param_doc_desc()

---

get_param_dict *Get dictionary of all parameters*

Description

Function provides a list containing important characteristics of the parameter used in the models. The list does contain only the definition of arguments for transformer models and all classifiers. The arguments of other functions in this package are documented separately.

The aim of this list is to automatize argument checking and widget generation for *AI for Education - Studio*.

Usage

```
get_param_dict()
```

Value

Returns a named list. The names correspond to specific arguments. The list contains a list for every argument with the following components:

- type: The type of allowed values.
- allow_null: A bool indicating if the argument can be set to NULL.
- min: The minimal value the argument can be. Set to NULL if not relevant. Set to -Inf if there is no minimum.
- max: The maximal value the argument can be. Set to NULL if not relevant. Set to Inf if there is no Minimum.
- desc: A string which includes the description of the argument written in markdown. This string is for the documentation the parameter.
- values_desc: A named list containing a description of every possible value. The names must exactly match the strings in allowed_values. Descriptions should be written in markdown.
- allowed_values: vector of allowed values. This is only relevant if the argument is not numeric. During the checking of the arguments it is checked if the provided values can be found in this vector. If all values are allowed set to NULL.

- default_value: The default value of the argument. If there is no default set to NULL.
- default_historic: Historic default value. This can be necessary for backward compatibility.
- gui_box: string Name of the box in AI for Education - Studio where the argument appears. If it should not appear set to NULL.
- gui_label: string Label of the controlling widget in AI for Education - Studio.

See Also

Other Parameter Dictionary: BaseModelsIndex, DataSetsIndex, TokenizerIndex, get_TEClassifiers_class_names(), get_called_args(), get_depr_obj_names(), get_magnitude_values(), get_param_def(), get_param_doc_desc()

---

get_param_doc_desc *Description of an argument*

Description

Function provides the description of an argument in markdown. Its aim is to be used for documenting the parameter of functions.

Usage

```
get_param_doc_desc(param_name)
```

Arguments

param_name	string Name of the parameter to request its definition.

Value

Returns a string which contains the description of the argument in markdown. The concrete format depends on the type of the argument.

See Also

Other Parameter Dictionary: BaseModelsIndex, DataSetsIndex, TokenizerIndex, get_TEClassifiers_class_names(), get_called_args(), get_depr_obj_names(), get_magnitude_values(), get_param_def(), get_param_dict()

get_py_package_version

Get versions of a specific python package

Description

Function for requesting the version of a specific python package.

Usage

```
get_py_package_version(package_name)
```

Arguments

package_name	string Name of the package.

Value

Returns the version as string or NA if the package does not exist or no version is available.

See Also

Other Utils Python Developers: get_py_package_versions(), load_all_py_scripts(), load_py_scripts(), run_py_file()

get_py_package_versions

Get versions of python components

Description

Function for requesting a summary of the versions of all critical python components.

Usage

```
get_py_package_versions()
```

Value

Returns a list that contains the version number of python and the versions of critical python packages. If a package is not available version is set to NA.

See Also

Other Utils Python Developers: get_py_package_version(), load_all_py_scripts(), load_py_scripts(), run_py_file()

get_recommended_py_versions

Recommended version of python packages

Description

Returns the minimum and maximum versions of the core python packages used in *aifeducation*. It is recommended to use packages of these version. Packages of other versions can result in errors or unexpected results.

Usage

```
get_recommended_py_versions()
```

Value

Returns a `data.frame` with the packages in the columns and the minimum and maximum version in the rows.

See Also

Other Installation and Configuration: `check_aif_py_modules()`, `install_aifeducation()`, `install_aifeducation_stu`
`install_py_modules()`, `prepare_session()`, `set_transformers_logger()`, `update_aifeducation()`

get_synthetic_cases_from_matrix

Create synthetic cases for balancing training data

Description

This function creates synthetic cases for balancing the training with classifier models.

Usage

```
get_synthetic_cases_from_matrix(
  matrix_form,
  times,
  features,
  target,
  sequence_length,
  method = "knnor",
  min_k = 1L,
  max_k = 6L
)
```

Arguments

matrix_form	Named matrix containing the text embeddings in a matrix form.
times	int for the number of sequences/times.
features	int for the number of features within each sequence.
target	Named factor containing the labels of the corresponding embeddings.
sequence_length	int Length of the text embedding sequences.
method	vector containing strings of the requested methods for generating new cases. Currently "knnor" from this package is available.
min_k	int The minimal number of nearest neighbors during sampling process.
max_k	int The maximum number of nearest neighbors during sampling process.

Value

list with the following components:

- syntetic_embeddings: Named data.frame containing the text embeddings of the synthetic cases.
- syntetic_targets: Named factor containing the labels of the corresponding synthetic cases.
- n_syntetic_units: table showing the number of synthetic cases for every label/category.

See Also

Other Utils Developers: auto_n_cores(), create_object(), create_synthetic_units_from_matrix(), generate_id(), get_n_chunks(), get_time_stamp(), matrix_to_array_c(), tensor_to_matrix_c(), to_categorical_c()

---

get_TEClassifiers_class_names

Get names of classifiers

Description

Function returns the names of all classifiers which are child classes of a specific super class.

Usage

```
get_TEClassifiers_class_names(super_class = NULL)
```

Arguments

super_class	string Name of the super class the classifiers should be a child of. To request the names of all classifiers set this argument to NULL.

Value

Returns a vector containing the names of the classifiers.

See Also

Other Parameter Dictionary: BaseModelsIndex, DataSetsIndex, TokenizerIndex, get_called_args(), get_depr_obj_names(), get_magnitude_values(), get_param_def(), get_param_dict(), get_param_doc_desc()

`get_test_data_for_classifiers` *Get test data*

Description

Function returns example data for testing the package

Usage

```
get_test_data_for_classifiers(class_range = c(2L, 3L), path_test_embeddings)
```

Arguments

`class_range`	`vector` containing the number of classes.
`path_test_embeddings`	`string` Path to the location where the test data is stored.

Value

Returns a `list` with test data.

See Also

Other Utils TestThat Developers: check_adjust_n_samples_on_CI(), generate_args_for_tests(), generate_embeddings(), generate_tensors(), get_current_args_for_print(), get_fixed_test_tensor(), random_bool_on_CI()

get_time_stamp *Time stamp*

Description

Function returns the time on the machine at the moment of calling.

Usage

```
get_time_stamp()
```

Value

Returns a `string` with date and time in format "%y-%m-%d %H:%M:%S".

See Also

Other Utils Developers: `auto_n_cores()`, `create_object()`, `create_synthetic_units_from_matrix()`, `generate_id()`, `get_n_chunks()`, `get_synthetic_cases_from_matrix()`, `matrix_to_array_c()`, `tensor_to_matrix_c()`, `to_categorical_c()`

gwet_ac *Calculate Gwet's AC1 and AC2*

Description

This function calculates Gwets Agreement Coefficients.

Usage

```
gwet_ac(rater_one, rater_two, additional_raters = NULL)
```

Arguments

`rater_one`	`factor` rating of the first coder.
`rater_two`	`factor` ratings of the second coder.
`additional_raters`	`list` Additional raters with same requirements as `rater_one` and `rater_two`. If there are no additional raters set to `NULL`.

Value

Returns a `list` with the following entries

- ac1: Gwet's Agreement Coefficient 1 (AC1) for nominal data which is unweighted.
- ac2_linear: Gwet's Agreement Coefficient 2 (AC2) for ordinal data with linear weights.
- ac2_quadratic: Gwet's Agreement Coefficient 2 (AC2) for ordinal data with quadratic weights.

Note

Weights are calculated as described in Gwet (2021).

Missing values are supported.

References

Gwet, K. L. (2021). Handbook of inter-rater reliability: The definitive guide to measuring the extent of agreement among raters (Fifth edition, volume 1). AgreeStat Analytics.

See Also

Other performance measures: `calc_standard_classification_measures()`, `cohens_kappa()`, `fleiss_kappa()`, `get_coder_metrics()`, `kendalls_w()`, `kripp_alpha()`

`HuggingFaceTokenizer` *HuggingFaceTokenizer*

Description

Abstract class for all tokenizers used with the 'transformers' library.

Value

Does return a new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::TokenizerBase` -> `HuggingFaceTokenizer`

Methods

Public methods:

- `HuggingFaceTokenizer$create_from_hf()`
- `HuggingFaceTokenizer$clone()`

Method `create_from_hf()`**:** Creates a tokenizer from a pretrained model

Usage:

```
HuggingFaceTokenizer$create_from_hf(model_dir)
```

Arguments:

`model_dir`

Returns: Does return a new object of this class.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Tokenizer: WordPieceTokenizer

`install_aifeducation` *Install aifeducation on a machine*

Description

Function for installing 'aifeducation' on a machine.

Using a virtual environment (`use_conda=FALSE`) If 'python' is already installed the installed version is used. In the case that the required version of 'python' is different from the existing version the new version is installed. In all other cases python will be installed on the system.

#' Using a conda environment (`use_conda=TRUE`) If 'miniconda' is already existing on the machine no installation of 'miniconda' is applied. In this case the system checks for update and updates 'miniconda' to the newest version. If 'miniconda' is not found on the system it will be installed.

Usage

```
install_aifeducation(
  install_aifeducation_studio = TRUE,
  python_version = "3.12",
  cuda_version = "12.9",
  use_conda = FALSE
)
```

Arguments

`install_aifeducation_studio`	`bool` If `TRUE` all necessary R packages are installed for using AI for Education Studio.
`python_version`	`string` Python version to use/install.
`cuda_version`	`string` determining the requested version of cuda.
`use_conda`	`bool` If `TRUE` installation installs 'miniconda' and uses 'conda' as package manager. If `FALSE` installation installs python and uses virtual environments for package management.

Value

Function does nothing return. It installs python, optional R packages, and necessary 'python' packages on a machine.

Note

On MAC OS torch will be installed without support for cuda.

See Also

Other Installation and Configuration: check_aif_py_modules(), get_recommended_py_versions(), install_aifeducation_studio(), install_py_modules(), prepare_session(), set_transformers_logger(), update_aifeducation()

`install_aifeducation_studio` *Install 'AI for Education - Studio' on a machine*

Description

Function installs/updates all relevant R packages necessary to run the shiny app "AI for Education - Studio'.

Usage

```
install_aifeducation_studio()
```

Value

Function does nothing return. It installs/updates R packages.

See Also

Other Installation and Configuration: check_aif_py_modules(), get_recommended_py_versions(), install_aifeducation(), install_py_modules(), prepare_session(), set_transformers_logger(), update_aifeducation()

`install_py_modules` *Installing necessary python modules to an environment*

Description

Function for installing the necessary python modules.

Usage

```
install_py_modules(
  envname = "aifeducation",
  transformer_version = "<=4.56.1",
  tokenizers_version = "<=0.22.0",
  pandas_version = "<=2.3.2",
  datasets_version = "<=3.6.0",
  codecarbon_version = "<=3.0.4",
  safetensors_version = "<=0.6.2",
```

```
  torcheval_version = "<=0.0.7",
  accelerate_version = "<=1.10.1",
  calflops_version = "<=0.3.2",
  pytorch_cuda_version = "12.9",
  python_version = "3.12",
  remove_first = FALSE,
  use_conda = FALSE
)
```

Arguments

envname	string Name of the environment where the packages should be installed.
transformer_version	string determining the desired version of the python library 'transformers'.
tokenizers_version	string determining the desired version of the python library 'tokenizers'.
pandas_version	string determining the desired version of the python library 'pandas'.
datasets_version	string determining the desired version of the python library 'datasets'.
codecarbon_version	string determining the desired version of the python library 'codecarbon'.
safetensors_version	string determining the desired version of the python library 'safetensors'.
torcheval_version	string determining the desired version of the python library 'torcheval'.
accelerate_version	string determining the desired version of the python library 'accelerate'.
calflops_version	string determining the desired version of the python library 'calflops'.
pytorch_cuda_version	string determining the desired version of 'cuda' for 'PyTorch'. To install 'PyTorch' without cuda set to NULL.
python_version	string Python version to use.
remove_first	bool If TRUE removes the environment completely before recreating the environment and installing the packages. If FALSE the packages are installed in the existing environment without any prior changes.
use_conda	bool If TRUE uses 'conda' for package management. If FALSE uses virtual environments for package management.

Value

Returns no values or objects. Function is used for installing the necessary python libraries in a conda environment.

Note

Function tries to identify the type of operating system. In the case that MAC OS is detected 'PyTorch' is installed without support for cuda.

See Also

Other Installation and Configuration: check_aif_py_modules(), get_recommended_py_versions(), install_aifeducation(), install_aifeducation_studio(), prepare_session(), set_transformers_logger(), update_aifeducation()

---

kendalls_w *Calculate Kendall's coefficient of concordance w*

Description

This function calculates Kendall's coefficient of concordance w with and without correction.

Usage

```
kendalls_w(rater_one, rater_two, additional_raters = NULL)
```

Arguments

rater_one	factor rating of the first coder.
rater_two	factor ratings of the second coder.
additional_raters	list Additional raters with same requirements as rater_one and rater_two. If there are no additional raters set to NULL.

Value

Returns a list containing the results for Kendall's coefficient of concordance w with and without correction.

See Also

Other performance measures: calc_standard_classification_measures(), cohens_kappa(), fleiss_kappa(), get_coder_metrics(), gwet_ac(), kripp_alpha()

---

knnor *K-Nearest Neighbor OveRsampling approach (KNNOR)*

Description

K-Nearest Neighbor OveRsampling approach (KNNOR)

Usage

```
knnor(dataset, k, aug_num, cycles_number_limit = 100L)
```

Arguments

dataset	list containing the following fields: • embeddings: an 2-D array (matrix) with size batch x times*features • labels: an 1-D array (vector) of integers with batch elements
k	unsigned integer number of nearest neighbors
aug_num	unsigned integer number of datapoints to be augmented
cycles_number_limit	unsigned integer number of maximum try cycles

Value

Returns artificial points (2-D array (matrix) with size aug_numxtimes*features‘)

References

Islam, A., Belhaouari, S. B., Rehman, A. U. & Bensmail, H. (2022). KNNOR: An oversampling technique for imbalanced datasets. Applied Soft Computing, 115, 108288. https://doi.org/10.1016/j.asoc.2021.108288

---

knnor_is_same_class *Validate a new point*

Description

Function written in C++ for validating a new point (KNNOR-Validation)

Usage

```
knnor_is_same_class(new_point, dataset, labels, k)
```

Arguments

new_point	1-D array (vector) new data point to be validated before adding (with times*features elements)
dataset	2-D array (matrix) current embeddings (with size batch x times*features)
labels	1-D array (vector) of integers with batch elements
k	unsigned integer number of nearest neighbors

Value

Returns TRUE if a new point can be added, otherwise - FALSE

kripp_alpha *Calculate Krippendorff's Alpha*

Description

This function calculates different Krippendorff's Alpha for nominal and ordinal variables.

Usage

```
kripp_alpha(rater_one, rater_two, additional_raters = NULL)
```

Arguments

rater_one	factor rating of the first coder.
rater_two	factor ratings of the second coder.
additional_raters	list Additional raters with same requirements as rater_one and rater_two. If there are no additional raters set to NULL.

Value

Returns a list containing the results for Krippendorff's Alpha for nominal and ordinal data.

Note

Missing values are supported.

References

Krippendorff, K. (2019). Content Analysis: An Introduction to Its Methodology (4th Ed.). SAGE

See Also

Other performance measures: calc_standard_classification_measures(), cohens_kappa(), fleiss_kappa(), get_coder_metrics(), gwet_ac(), kendalls_w()

`LargeDataSetBase` *Abstract base class for large data sets*

Description

This object contains public and private methods which may be useful for every large data sets. Objects of this class are not intended to be used directly.

Value

Returns a new object of this class.

Methods

Public methods:

- `LargeDataSetBase$n_cols()`
- `LargeDataSetBase$n_rows()`
- `LargeDataSetBase$get_colnames()`
- `LargeDataSetBase$get_dataset()`
- `LargeDataSetBase$reduce_to_unique_ids()`
- `LargeDataSetBase$select()`
- `LargeDataSetBase$get_ids()`
- `LargeDataSetBase$save()`
- `LargeDataSetBase$load_from_disk()`
- `LargeDataSetBase$load()`
- `LargeDataSetBase$set_package_versions()`
- `LargeDataSetBase$get_package_versions()`
- `LargeDataSetBase$get_all_fields()`
- `LargeDataSetBase$clone()`

Method `n_cols()`**:** Number of columns in the data set.

Usage:

```
LargeDataSetBase$n_cols()
```

Returns: `int` describing the number of columns in the data set.

Method `n_rows()`**:** Number of rows in the data set.

Usage:

```
LargeDataSetBase$n_rows()
```

Returns: `int` describing the number of rows in the data set.

Method `get_colnames()`**:** Get names of the columns in the data set.

Usage:

```
LargeDataSetBase$get_colnames()
```

Returns: `vector` containing the names of the columns as `strings`.

Method `get_dataset()`**:** Get data set.

Usage:

```
LargeDataSetBase$get_dataset()
```

Returns: Returns the data set of this object as an object of class `datasets.arrow_dataset.Dataset`.

Method `reduce_to_unique_ids()`**:** Reduces the data set to a data set containing only unique ids. In the case an id exists multiple times in the data set the first case remains in the data set. The other cases are dropped.
Attention Calling this method will change the data set in place.

Usage:

```
LargeDataSetBase$reduce_to_unique_ids()
```

Returns: Method does not return anything. It changes the data set of this object in place.

Method `select()`**:** Returns a data set which contains only the cases belonging to the specific indices.

Usage:

```
LargeDataSetBase$select(indicies)
```

Arguments:

`indicies` `vector` of `int` for selecting rows in the data set. **Attention** The indices are zero-based.

Returns: Returns a data set of class `datasets.arrow_dataset.Dataset` with the selected rows.

Method `get_ids()`**:** Get ids

Usage:

```
LargeDataSetBase$get_ids()
```

Returns: Returns a `vector` containing the ids of every row as `strings`.

Method `save()`**:** Saves a data set to disk.

Usage:

```
LargeDataSetBase$save(dir_path, folder_name, create_dir = TRUE)
```

Arguments:

`dir_path` Path where to store the data set.

`folder_name` `string` Name of the folder for storing the data set.

`create_dir` `bool` If `True` the directory will be created if it does not exist.

Returns: Method does not return anything. It write the data set to disk.

Method `load_from_disk()`**:** loads an object of class LargeDataSetBase from disk 'and updates the object to the current version of the package.

Usage:

```
LargeDataSetBase$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the data set set is stored.

Returns: Method does not return anything. It loads an object from disk.

Method `load()`**:** Loads a data set from disk.

Usage:

```
LargeDataSetBase$load(dir_path)
```

Arguments:

`dir_path` Path where the data set is stored.

Returns: Method does not return anything. It loads a data set from disk.

Method `set_package_versions()`**:** Method for setting the package version for 'aifeducation', 'reticulate', 'torch', and 'numpy' to the currently used versions.

Usage:

```
LargeDataSetBase$set_package_versions()
```

Returns: Method does not return anything. It is used to set the private fields fo package versions.

Method `get_package_versions()`**:** Method for requesting a summary of the R and python packages' versions used for creating the model.

Usage:

```
LargeDataSetBase$get_package_versions()
```

Returns: Returns a `list` containing the versions of the relevant R and python packages.

Method `get_all_fields()`**:** Return all fields.

Usage:

```
LargeDataSetBase$get_all_fields()
```

Returns: Method returns a `list` containing all public and private fields of the object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `BaseModelCore`, `ClassifiersBasedOnTextEmbeddings`, `DataManagerClassifier`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnProtoNet`, `TEClassifiersBasedOnRegular`, `TokenizerBase`

LargeDataSetForText *Abstract class for large data sets containing raw texts*

Description

This object stores raw texts. The data of this objects is not stored in memory directly. By using memory mapping these objects allow to work with data sets which do not fit into memory/RAM.

Value

Returns a new object of this class.

Super class

aifeducation::LargeDataSetBase -> LargeDataSetForText

Methods

Public methods:

- LargeDataSetForText$new()
- LargeDataSetForText$add_from_files_txt()
- LargeDataSetForText$add_from_files_pdf()
- LargeDataSetForText$add_from_files_xlsx()
- LargeDataSetForText$add_from_data.frame()
- LargeDataSetForText$get_private()
- LargeDataSetForText$clone()

Method `new()`**:** Method for creation of LargeDataSetForText instance. It can be initialized with `init_data` parameter if passed (Uses `add_from_data.frame()` method if `init_data` is `data.frame`).

Usage:

```
LargeDataSetForText$new(init_data = NULL)
```

Arguments:

`init_data` Initial `data.frame` for dataset.

Returns: A new instance of this class initialized with `init_data` if passed.

Method `add_from_files_txt()`**:** Method for adding raw texts saved within .txt files to the data set. Please note the the directory should contain one folder for each .txt file. In order to create an informative data set every folder can contain the following additional files:

- bib_entry.txt: containing a text version of the bibliographic information of the raw text.
- license.txt: containing a statement about the license to use the raw text such as "CC BY".
- url_license.txt: containing the url/link to the license in the internet.
- text_license.txt: containing the license in raw text.

- url_source.txt: containing the url/link to the source in the internet.
  The id of every .txt file is the file name without file extension. Please be aware to provide unique file names. Id and raw texts are mandatory, bibliographic and license information are optional.

Usage:

```
LargeDataSetForText$add_from_files_txt(
  dir_path,
  batch_size = 500L,
  log_file = NULL,
  log_write_interval = 2L,
  log_top_value = 0L,
  log_top_total = 1L,
  log_top_message = NA,
  clean_text = TRUE,
  trace = TRUE
)
```

Arguments:

`dir_path` Path to the directory where the files are stored.

`batch_size int` determining the number of files to process at once.

`log_file string` Path to the file where the log should be saved. If no logging is desired set this argument to `NULL`.

`log_write_interval int` Time in seconds determining the interval in which the logger should try to update the log files. Only relevant if `log_file` is not `NULL`.

`log_top_value int` indicating the current iteration of the process.

`log_top_total int` determining the maximal number of iterations.

`log_top_message string` providing additional information of the process.

`clean_text bool` If `TRUE` the text is modified to improve the quality of the following analysis:

- Some special symbols are removed.
- All spaces at the beginning and the end of a row are removed.
- Multiple spaces are reduced to single space.
- All rows with a number from 1 to 999 at the beginning or at the end are removed (header and footer).
- List of content is removed.
- Hyphenation is made undone.
- Line breaks within a paragraph are removed.
- Multiple line breaks are reduced to a single line break.

`trace bool` If `TRUE` information on the progress is printed to the console.

Returns: The method does not return anything. It adds new raw texts to the data set.

Method `add_from_files_pdf()`**:** Method for adding raw texts saved within .pdf files to the data set. Please note the the directory should contain one folder for each .pdf file. In order to create an informative data set every folder can contain the following additional files:

- bib_entry.txt: containing a text version of the bibliographic information of the raw text.
- license.txt: containing a statement about the license to use the raw text such as "CC BY".

- url_license.txt: containing the url/link to the license in the internet.
- text_license.txt: containing the license in raw text.
- url_source.txt: containing the url/link to the source in the internet.
  The id of every .pdf file is the file name without file extension. Please be aware to provide unique file names. Id and raw texts are mandatory, bibliographic and license information are optional.

Usage:

```
LargeDataSetForText$add_from_files_pdf(
  dir_path,
  batch_size = 500L,
  log_file = NULL,
  log_write_interval = 2L,
  log_top_value = 0L,
  log_top_total = 1L,
  log_top_message = NA,
  clean_text = TRUE,
  trace = TRUE
)
```

Arguments:

`dir_path` Path to the directory where the files are stored.

`batch_size` `int` determining the number of files to process at once.

`log_file` `string` Path to the file where the log should be saved. If no logging is desired set this argument to `NULL`.

`log_write_interval` `int` Time in seconds determining the interval in which the logger should try to update the log files. Only relevant if `log_file` is not `NULL`.

`log_top_value` `int` indicating the current iteration of the process.

`log_top_total` `int` determining the maximal number of iterations.

`log_top_message` `string` providing additional information of the process.

`clean_text` `bool` If `TRUE` the text is modified to improve the quality of the following analysis:

- Some special symbols are removed.
- All spaces at the beginning and the end of a row are removed.
- Multiple spaces are reduced to single space.
- All rows with a number from 1 to 999 at the beginning or at the end are removed (header and footer).
- List of content is removed.
- Hyphenation is made undone.
- Line breaks within a paragraph are removed.
- Multiple line breaks are reduced to a single line break.

`trace` `bool` If `TRUE` information on the progress is printed to the console.

Returns: The method does not return anything. It adds new raw texts to the data set.

Method `add_from_files_xlsx()`**:** Method for adding raw texts saved within .xlsx files to the data set. The method assumes that the texts are saved in the rows and that the columns store the id and the raw texts in the columns. In addition, a column for the bibliography information and

the license can be added. The column names for these rows must be specified with the following arguments. They must be the same for all .xlsx files in the chosen directory. Id and raw texts are mandatory, bibliographic, license, license's url, license's text, and source's url are optional. Additional columns are dropped.

Usage:

```
LargeDataSetForText$add_from_files_xlsx(
  dir_path,
  trace = TRUE,
  id_column = "id",
  text_column = "text",
  bib_entry_column = "bib_entry",
  license_column = "license",
  url_license_column = "url_license",
  text_license_column = "text_license",
  url_source_column = "url_source",
  log_file = NULL,
  log_write_interval = 2L,
  log_top_value = 0L,
  log_top_total = 1L,
  log_top_message = NA
)
```

Arguments:

`dir_path` Path to the directory where the files are stored.

`trace` `bool` If `TRUE` prints information on the progress to the console.

`id_column` `string` Name of the column storing the ids for the texts.

`text_column` `string` Name of the column storing the raw text.

`bib_entry_column` `string` Name of the column storing the bibliographic information of the texts.

`license_column` `string` Name of the column storing information about the licenses.

`url_license_column` `string` Name of the column storing information about the url to the license in the internet.

`text_license_column` `string` Name of the column storing the license as text.

`url_source_column` `string` Name of the column storing information about about the url to the source in the internet.

`log_file` `string` Path to the file where the log should be saved. If no logging is desired set this argument to `NULL`.

`log_write_interval` `int` Time in seconds determining the interval in which the logger should try to update the log files. Only relevant if `log_file` is not `NULL`.

`log_top_value` `int` indicating the current iteration of the process.

`log_top_total` `int` determining the maximal number of iterations.

`log_top_message` `string` providing additional information of the process.

Returns: The method does not return anything. It adds new raw texts to the data set.

Method `add_from_data.frame()`**:** Method for adding raw texts from a `data.frame`

Usage:

```
LargeDataSetForText$add_from_data.frame(data_frame)
```

Arguments:

`data_frame` Object of class `data.frame` with at least the following columns "id","text","bib_entry", "license", "url_license", "text_license", and "url_source". If "id" and7or "text" is missing an error occurs. If the other columns are not present in the `data.frame` they are added with empty values(`NA`). Additional columns are dropped.

Returns: The method does not return anything. It adds new raw texts to the data set.

Method `get_private()`**:** Method for requesting all private fields and methods. Used for loading and updating an object.

Usage:

```
LargeDataSetForText$get_private()
```

Returns: Returns a `list` with all private fields and methods.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Data Management: `EmbeddedText`, `LargeDataSetForTextEmbeddings`

`LargeDataSetForTextEmbeddings` *Abstract class for large data sets containing text embeddings*

Description

This object stores text embeddings which are usually produced by an object of class TextEmbeddingModel. The data of this objects is not stored in memory directly. By using memory mapping these objects allow to work with data sets which do not fit into memory/RAM.

LargeDataSetForTextEmbeddings are used for storing and managing the text embeddings created with objects of class TextEmbeddingModel. Objects of class LargeDataSetForTextEmbeddings serve as input for objects of class ClassifiersBasedOnTextEmbeddings and TEFeatureExtractor. The main aim of this class is to provide a structured link between embedding models and classifiers. Since objects of this class save information on the text embedding model that created the text embedding it ensures that only embeddings generated with same embedding model are combined. Furthermore, the stored information allows objects to check if embeddings of the correct text embedding model are used for training and predicting.

This class is not designed for a direct use.

Value

Returns a new object of this class.

Super class

aifeducation::LargeDataSetBase -> LargeDataSetForTextEmbeddings

Methods

Public methods:

- LargeDataSetForTextEmbeddings$configure()
- LargeDataSetForTextEmbeddings$is_configured()
- LargeDataSetForTextEmbeddings$get_text_embedding_model_name()
- LargeDataSetForTextEmbeddings$get_model_info()
- LargeDataSetForTextEmbeddings$load_from_disk()
- LargeDataSetForTextEmbeddings$get_model_label()
- LargeDataSetForTextEmbeddings$add_feature_extractor_info()
- LargeDataSetForTextEmbeddings$get_feature_extractor_info()
- LargeDataSetForTextEmbeddings$is_compressed()
- LargeDataSetForTextEmbeddings$get_times()
- LargeDataSetForTextEmbeddings$get_features()
- LargeDataSetForTextEmbeddings$get_original_features()
- LargeDataSetForTextEmbeddings$get_pad_value()
- LargeDataSetForTextEmbeddings$add_embeddings_from_array()
- LargeDataSetForTextEmbeddings$add_embeddings_from_EmbeddedText()
- LargeDataSetForTextEmbeddings$add_embeddings_from_LargeDataSetForTextEmbeddings()
- LargeDataSetForTextEmbeddings$convert_to_EmbeddedText()
- LargeDataSetForTextEmbeddings$clone()

Method `configure()`**:** Creates a new object representing text embeddings.

Usage:

```
LargeDataSetForTextEmbeddings$configure(
  model_name = NA,
  model_label = NA,
  model_date = NA,
  model_method = NA,
  model_version = NA,
  model_language = NA,
  param_seq_length = NA,
  param_chunks = NULL,
  param_features = NULL,
  param_overlap = NULL,
  param_emb_layer_min = NULL,
  param_emb_layer_max = NULL,
  param_emb_pool_type = NULL,
```

```
  param_pad_value = -100L,
  param_aggregation = NULL
)
```

Arguments:

`model_name string` Name of the model that generates this embedding.

`model_label string` Label of the model that generates this embedding.

`model_date string` Date when the embedding generating model was created.

`model_method string` Method of the underlying embedding model.

`model_version string` Version of the model that generated this embedding.

`model_language string` Language of the model that generated this embedding.

`param_seq_length int` Maximum number of tokens that processes the generating model for a chunk.

`param_chunks int` Maximum number of chunks which are supported by the generating model.

`param_features int` Number of dimensions of the text embeddings.

`param_overlap int` Number of tokens that were added at the beginning of the sequence for the next chunk by this model.

`param_emb_layer_min int` or `string` determining the first layer to be included in the creation of embeddings.

`param_emb_layer_max int` or `string` determining the last layer to be included in the creation of embeddings.

`param_emb_pool_type string` determining the method for pooling the token embeddings within each layer.

`param_pad_value int` Value indicating padding. This value should no be in the range of regluar values for computations. Thus it is not recommended to chance this value. Default is `-100`. Allowed values: `x <= -100`

`param_aggregation string` Aggregation method of the hidden states. Deprecated. Only included for backward compatibility.

Returns: The method returns a new object of this class.

Method `is_configured()`**:** Method for checking if the model was successfully configured. An object can only be used if this value is `TRUE`.

Usage:

```
LargeDataSetForTextEmbeddings$is_configured()
```

Returns: `bool` `TRUE` if the model is fully configured. `FALSE` if not.

Method `get_text_embedding_model_name()`**:** Method for requesting the name (unique id) of the underlying text embedding model.

Usage:

```
LargeDataSetForTextEmbeddings$get_text_embedding_model_name()
```

Returns: Returns a `string` describing name of the text embedding model.

Method `get_model_info()`**:** Method for retrieving information about the model that generated this embedding.

Usage:

```
LargeDataSetForTextEmbeddings$get_model_info()
```

Returns: `list` containing all saved information about the underlying text embedding model.

Method `load_from_disk()`**:** loads an object of class LargeDataSetForTextEmbeddings from disk and updates the object to the current version of the package.

Usage:

```
LargeDataSetForTextEmbeddings$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the data set set is stored.

Returns: Method does not return anything. It loads an object from disk.

Method `get_model_label()`**:** Method for retrieving the label of the model that generated this embedding.

Usage:

```
LargeDataSetForTextEmbeddings$get_model_label()
```

Returns: `string` Label of the corresponding text embedding model

Method `add_feature_extractor_info()`**:** Method setting information on the TEFeatureExtractor that was used to reduce the number of dimensions of the text embeddings. This information should only be used if a TEFeatureExtractor was applied.

Usage:

```
LargeDataSetForTextEmbeddings$add_feature_extractor_info(
  model_name,
  model_label = NA,
  features = NA,
  method = NA,
  noise_factor = NA,
  optimizer = NA
)
```

Arguments:

`model_name` `string` Name of the underlying TextEmbeddingModel.

`model_label` `string` Label of the underlying TextEmbeddingModel.

`features` `int` Number of dimension (features) for the **compressed** text embeddings.

`method` `string` Method that the TEFeatureExtractor applies for genereating the compressed text embeddings.

`noise_factor` `double` Noise factor of the TEFeatureExtractor.

`optimizer` `string` Optimizer used during training the TEFeatureExtractor.

Returns: Method does nothing return. It sets information on a TEFeatureExtractor.

Method `get_feature_extractor_info()`**:** Method for receiving information on the TEFeatureExtractor that was used to reduce the number of dimensions of the text embeddings.

Usage:

```
LargeDataSetForTextEmbeddings$get_feature_extractor_info()
```

Returns: Returns a `list` with information on the TEFeatureExtractor. If no TEFeatureExtractor was used it returns `NULL`.

Method `is_compressed()`**:** Checks if the text embedding were reduced by a TEFeatureExtractor.

Usage:

```
LargeDataSetForTextEmbeddings$is_compressed()
```

Returns: Returns `TRUE` if the number of dimensions was reduced by a TEFeatureExtractor. If not return `FALSE`.

Method `get_times()`**:** Number of chunks/times of the text embeddings.

Usage:

```
LargeDataSetForTextEmbeddings$get_times()
```

Returns: Returns an `int` describing the number of chunks/times of the text embeddings.

Method `get_features()`**:** Number of actual features/dimensions of the text embeddings.In the case a TEFeatureExtractor was used the number of features is smaller as the original number of features. To receive the original number of features (the number of features before applying a TEFeatureExtractor) you can use the method `get_original_features` of this class.

Usage:

```
LargeDataSetForTextEmbeddings$get_features()
```

Returns: Returns an `int` describing the number of features/dimensions of the text embeddings.

Method `get_original_features()`**:** Number of original features/dimensions of the text embeddings.

Usage:

```
LargeDataSetForTextEmbeddings$get_original_features()
```

Returns: Returns an `int` describing the number of features/dimensions if no TEFeatureExtractor) is used or before a TEFeatureExtractor) is applied.

Method `get_pad_value()`**:** Value for indicating padding.

Usage:

```
LargeDataSetForTextEmbeddings$get_pad_value()
```

Returns: Returns an `int` describing the value used for padding.

Method `add_embeddings_from_array()`**:** Method for adding new data to the data set from an `array`. Please note that the method does not check if cases already exist in the data set. To reduce the data set to unique cases call the method `reduce_to_unique_ids`.

Usage:

```
LargeDataSetForTextEmbeddings$add_embeddings_from_array(embedding_array)
```

Arguments:

`embedding_array` `array` containing the text embeddings.

Returns: The method does not return anything. It adds new data to the data set.

Method `add_embeddings_from_EmbeddedText()`**:** Method for adding new data to the data set from an EmbeddedText. Please note that the method does not check if cases already exist in the data set. To reduce the data set to unique cases call the method `reduce_to_unique_ids`.

Usage:

```
LargeDataSetForTextEmbeddings$add_embeddings_from_EmbeddedText(EmbeddedText)
```

Arguments:

`EmbeddedText` Object of class EmbeddedText.

Returns: The method does not return anything. It adds new data to the data set.

Method `add_embeddings_from_LargeDataSetForTextEmbeddings()`**:** Method for adding new data to the data set from an LargeDataSetForTextEmbeddings. Please note that the method does not check if cases already exist in the data set. To reduce the data set to unique cases call the method `reduce_to_unique_ids`.

Usage:

```
LargeDataSetForTextEmbeddings$add_embeddings_from_LargeDataSetForTextEmbeddings(
  dataset
)
```

Arguments:

`dataset` Object of class LargeDataSetForTextEmbeddings.

Returns: The method does not return anything. It adds new data to the data set.

Method `convert_to_EmbeddedText()`**:** Method for converting this object to an object of class EmbeddedText.

Attention This object uses memory mapping to allow the usage of data sets that do not fit into memory. By calling this method the data set will be loaded and stored into memory/RAM. This may lead to an out-of-memory error.

Usage:

```
LargeDataSetForTextEmbeddings$convert_to_EmbeddedText()
```

Returns: LargeDataSetForTextEmbeddings an object of class EmbeddedText which is stored in the memory/RAM.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Data Management: `EmbeddedText`, `LargeDataSetForText`

load_all_py_scripts *Load and re-load all python scripts*

Description

Function loads or re-loads all python scripts within the package 'aifeducation'.

Usage

```
load_all_py_scripts()
```

Value

Function does nothing return. It loads the requested scripts.

See Also

Other Utils Python Developers: `get_py_package_version()`, `get_py_package_versions()`, `load_py_scripts()`, `run_py_file()`

load_from_disk *Loading objects created with 'aifeducation'*

Description

Function for loading objects created with 'aifeducation'.

Usage

```
load_from_disk(dir_path)
```

Arguments

`dir_path`	`string` Path to the directory where the model is stored.

Value

Returns an object of class TEClassifierRegular, TEClassifierProtoNet, TEFeatureExtractor, TextEmbeddingModel, LargeDataSetForTextEmbeddings, LargeDataSetForText or EmbeddedText.

See Also

Other Saving and Loading: `save_to_disk()`

load_py_scripts *Load and re-load python scripts*

Description

Function loads or re-loads python scripts within the package 'aifeducation'.

Usage

```
load_py_scripts(files)
```

Arguments

files	vector containing the file names of the scripts that should be loaded.

Value

Function does nothing return. It loads the requested scripts.

See Also

Other Utils Python Developers: get_py_package_version(), get_py_package_versions(), load_all_py_scripts(), run_py_file()

long_load_target_data *Load target data for long running tasks*

Description

Function loads the target data for a long running task.

Usage

```
long_load_target_data(file_path, selectet_column)
```

Arguments

file_path	string Path to the file storing the target data.
selectet_column	string Name of the column containing the target data.

Details

This function assumes that the target data is stored as a columns with the cases in the rows and the categories in the columns. The ids of the cases must be stored in a column called "id".

Value

Returns a named factor containing the target data.

See Also

Other Utils Studio Developers: add_missing_args(), create_data_embeddings_description(), summarize_args_for_long_task()

`matrix_to_array_c` *Reshape matrix to array*

Description

Function written in C++ for reshaping a matrix containing sequential data into an array for use with keras.

Usage

```
matrix_to_array_c(matrix, times, features)
```

Arguments

`matrix`	`matrix` containing the sequential data.
`times`	`uword` Number of sequences.
`features`	`uword` Number of features within each sequence.

Value

Returns an array. The first dimension corresponds to the cases, the second to the times, and the third to the features.

See Also

Other Utils Developers: auto_n_cores(), create_object(), create_synthetic_units_from_matrix(), generate_id(), get_n_chunks(), get_synthetic_cases_from_matrix(), get_time_stamp(), tensor_to_matrix_c(), to_categorical_c()

ModelsBasedOnTextEmbeddings

Base class for models using neural nets

Description

Abstract class for all models that do not rely on the python library 'transformers'. All models of this class require text embeddings as input. These are provided as objects of class EmbeddedText or LargeDataSetForTextEmbeddings.

Objects of this class containing fields and methods used in several other classes in 'AI for Education'.

This class is **not** designed for a direct application and should only be used by developers.

Value

A new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `ModelsBasedOnTextEmbeddings`

Methods

Public methods:

- `ModelsBasedOnTextEmbeddings$get_text_embedding_model()`
- `ModelsBasedOnTextEmbeddings$get_text_embedding_model_name()`
- `ModelsBasedOnTextEmbeddings$check_embedding_model()`
- `ModelsBasedOnTextEmbeddings$save()`
- `ModelsBasedOnTextEmbeddings$load_from_disk()`
- `ModelsBasedOnTextEmbeddings$plot_training_history()`
- `ModelsBasedOnTextEmbeddings$clone()`

Method `get_text_embedding_model()`**:** Method for requesting the text embedding model information.

Usage:

```
ModelsBasedOnTextEmbeddings$get_text_embedding_model()
```

Returns: `list` of all relevant model information on the text embedding model underlying the model.

Method `get_text_embedding_model_name()`**:** Method for requesting the name (unique id) of the underlying text embedding model.

Usage:

```
ModelsBasedOnTextEmbeddings$get_text_embedding_model_name()
```

Returns: Returns a `string` describing name of the text embedding model.

Method `check_embedding_model()`: Method for checking if the provided text embeddings are created with the same TextEmbeddingModel as the model.

Usage:

```
ModelsBasedOnTextEmbeddings$check_embedding_model(text_embeddings)
```

Arguments:

`text_embeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

Returns: `TRUE` if the underlying TextEmbeddingModel are the same. `FALSE` if the models differ.

Method `save()`: Method for saving a model.

Usage:

```
ModelsBasedOnTextEmbeddings$save(dir_path, folder_name)
```

Arguments:

`dir_path` `string` Path of the directory where the model should be saved.

`folder_name` `string` Name of the folder that should be created within the directory.

Returns: Function does not return a value. It saves the model to disk.

Method `load_from_disk()`: loads an object from disk and updates the object to the current version of the package.

Usage:

```
ModelsBasedOnTextEmbeddings$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the object set is stored.

Returns: Method does not return anything. It loads an object from disk.

Method `plot_training_history()`: Method for requesting a plot of the training history. This method requires the *R* package 'ggplot2' to work.

Usage:

```
ModelsBasedOnTextEmbeddings$plot_training_history(
  final_training = FALSE,
  pl_step = NULL,
  measure = "loss",
  y_min = NULL,
  y_max = NULL,
  add_min_max = TRUE,
  text_size = 10L
)
```

Arguments:

`final_training` `bool` If `FALSE` the values of the performance estimation are used. If `TRUE` only the epochs of the final training are used.

`pl_step` `int` Number of the step during pseudo labeling to plot. Only relevant if the model was trained with active pseudo labeling.

`measure` Measure to plot.

`y_min` Minimal value for the y-axis. Set to `NULL` for an automatic adjustment.

`y_max` Maximal value for the y-axis. Set to `NULL` for an automatic adjustment.

`add_min_max` `bool` If `TRUE` the minimal and maximal values during performance estimation are port of the plot. If `FALSE` only the mean values are shown. Parameter is ignored if `final_training=TRUE`.

`text_size` Size of the text.

Returns: Returns a plot of class `ggplot` visualizing the training process. Prepare history data of objects Function for preparing the history data of a model in order to be plotted in AI for Education - Studio.
final `bool` If `TRUE` the history data of the final training is used for the data set. pl_step `int` If `use_pl=TRUE` select the step within pseudo labeling for which the data should be prepared. Returns a named `list` with the training history data of the model. The reported measures depend on the provided model.
Utils Studio Developers internal

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `BaseModelCore`, `ClassifiersBasedOnTextEmbeddings`, `DataManagerClassifier`, `LargeDataSetBase`, `TEClassifiersBasedOnProtoNet`, `TEClassifiersBasedOnRegular`, `TokenizerBase`

`output_message` *Print message*

Description

Prints a message `msg` if `trace` parameter is `TRUE` with current date with `message()` or `cat()` function.

Usage

```
output_message(msg, trace, msg_fun)
```

Arguments

`msg`	`string` Message that should be printed.
`trace`	`bool` Silent printing (`FALSE`) or not (`TRUE`).
`msg_fun`	`bool` value that determines what function should be used. `TRUE` for `message()`, `FALSE` for `cat()`.

Value

This function returns nothing.

See Also

Other Utils Log Developers: `cat_message()`, `clean_pytorch_log_transformers()`, `print_message()`, `read_log()`, `read_loss_log()`, `reset_log()`, `reset_loss_log()`, `write_log()`

`prepare_r_array_for_dataset` *Convert R array for arrow data set*

Description

Function converts a R array into a numpy array that can be added to an arrow data set. The array should represent embeddings.

Usage

```
prepare_r_array_for_dataset(r_array)
```

Arguments

`r_array`	`array` representing embeddings.

Value

Returns a numpy array.

See Also

Other Utils Python Data Management Developers: `class_vector_to_py_dataset()`, `data.frame_to_py_dataset()`, `get_batches_index()`, `py_dataset_to_embeddings()`, `reduce_to_unique()`, `tensor_list_to_numpy()`, `tensor_to_numpy()`

prepare_session *Function for setting up a python environment within R.*

Description

This functions checks for python and a specified environment. If the environment exists it will be activated. If python is already initialized it uses the current environment.

Usage

```
prepare_session(env_type = "auto", envname = "aifeducation")
```

Arguments

env_type	string If set to "venv" virtual environment is requested. If set to "conda" a 'conda' environment is requested. If set to "auto" the function tries to activate a virtual environment with the given name. If this environment does not exist it tries to activate a conda environment with the given name. If this fails the default virtual environment is used.
envname	string envname name of the requested environment.

Value

Function does not return anything. It is used for preparing python and R.

See Also

Other Installation and Configuration: check_aif_py_modules(), get_recommended_py_versions(), install_aifeducation(), install_aifeducation_studio(), install_py_modules(), set_transformers_logger(), update_aifeducation()

print_message *Print message (message())*

Description

Prints a message msg if trace parameter is TRUE with current date with message() function.

Usage

```
print_message(msg, trace)
```

Arguments

msg	string Message that should be printed.
trace	bool Silent printing (FALSE) or not (TRUE).

Value

This function returns nothing.

See Also

Other Utils Log Developers: `cat_message()`, `clean_pytorch_log_transformers()`, `output_message()`, `read_log()`, `read_loss_log()`, `reset_log()`, `reset_loss_log()`, `write_log()`

`py_dataset_to_embeddings`

Convert arrow data set to an arrow data set

Description

Function for converting an arrow data set into a data set that can be used to store and process embeddings.

Usage

```
py_dataset_to_embeddings(py_dataset)
```

Arguments

`py_dataset`	Object of class `datasets.arrow_dataset.Dataset`.

Value

Returns the data set of class `datasets.arrow_dataset.Dataset` with only two columns ("id","input"). "id" stores the name of the cases while "input" stores the embeddings.

See Also

Other Utils Python Data Management Developers: `class_vector_to_py_dataset()`, `data.frame_to_py_dataset()`, `get_batches_index()`, `prepare_r_array_for_dataset()`, `reduce_to_unique()`, `tensor_list_to_numpy()`, `tensor_to_numpy()`

random_bool_on_CI *Random bool on Continuous Integration*

Description

Function returns randomly TRUE or FALSE if on CI. It returns FALSE if it is not on CI.

Usage

```
random_bool_on_CI()
```

Value

Returns a bool.

See Also

Other Utils TestThat Developers: check_adjust_n_samples_on_CI(), generate_args_for_tests(), generate_embeddings(), generate_tensors(), get_current_args_for_print(), get_fixed_test_tensor(), get_test_data_for_classifiers()

read_log *Function for reading a log file in R*

Description

This function reads a log file at the given location. The log file should be created with write_log.

Usage

```
read_log(file_path)
```

Arguments

file_path	string Path to the log file.

Value

Returns a matrix containing the log file.

See Also

Other Utils Log Developers: cat_message(), clean_pytorch_log_transformers(), output_message(), print_message(), read_loss_log(), reset_log(), reset_loss_log(), write_log()

read_loss_log *Function for reading a log file containing a record of the loss during training.*

Description

This function reads a log file that contains values for every epoch for the loss. The values are grouped for training and validation data. The log contains values for test data if test data was available during training.

Usage

```
read_loss_log(path_loss)
```

Arguments

path_loss	string Path to the log file.

Details

In general the loss is written by a python function during model's training.

Value

Function returns a matrix that contains two or three row depending on the data inside the loss log. In the case of two rows the first represents the training data and the second the validation data. In the case of three rows the third row represents the values for test data. All Columns represent the epochs.

See Also

Other Utils Log Developers: cat_message(), clean_pytorch_log_transformers(), output_message(), print_message(), read_log(), reset_log(), reset_loss_log(), write_log()

reduce_to_unique *Reduce to unique cases*

Description

Function creates an arrow data set that contains only unique cases. That is, duplicates are removed.

Usage

```
reduce_to_unique(dataset_to_reduce, column_name)
```

Arguments

dataset_to_reduce	Object of class datasets.arrow_dataset.Dataset.
column_name	string Name of the column whose values should be unique.

Value

Returns a data set of class datasets.arrow_dataset.Dataset where the duplicates are removed according to the given column.

See Also

Other Utils Python Data Management Developers: class_vector_to_py_dataset(), data.frame_to_py_dataset(), get_batches_index(), prepare_r_array_for_dataset(), py_dataset_to_embeddings(), tensor_list_to_numpy(), tensor_to_numpy()

---

reset_log *Function that resets a log file.*

Description

This function writes a log file with default values. The file can be read with read_log.

Usage

```
reset_log(log_path)
```

Arguments

log_path	string Path to the log file.

Value

Function does nothing return. It is used to write an "empty" log file.

See Also

Other Utils Log Developers: cat_message(), clean_pytorch_log_transformers(), output_message(), print_message(), read_log(), read_loss_log(), reset_loss_log(), write_log()

reset_loss_log *Reset log for loss information*

Description

This function writes an empty log file for loss information.

Usage

```
reset_loss_log(log_path, epochs)
```

Arguments

log_path	string Path to the log file.
epochs	int Number of epochs for the complete training process.

Value

Function does nothing return. It writes a log file at the given location. The file is a .csv file that contains three rows. The first row takes the value for the training, the second for the validation, and the third row for the test data. The columns represent epochs.

See Also

Other Utils Log Developers: cat_message(), clean_pytorch_log_transformers(), output_message(), print_message(), read_log(), read_loss_log(), reset_log(), write_log()

run_py_file *Run python file*

Description

Used to run python files with reticulate::py_run_file() from folder python.

Usage

```
run_py_file(py_file_name)
```

Arguments

py_file_name	string Name of a python file to run. The file must be in the python folder of aifeducation package.

Value

This function returns nothing.

See Also

Other Utils Python Developers: `get_py_package_version()`, `get_py_package_versions()`, `load_all_py_scripts()`, `load_py_scripts()`

`save_to_disk` *Saving objects created with 'aifeducation'*

Description

Function for saving objects created with 'aifeducation'.

Usage

```
save_to_disk(object, dir_path, folder_name)
```

Arguments

`object`	Object of class TEClassifierRegular, TEClassifierProtoNet, TEFeatureExtractor, TextEmbeddingModel, LargeDataSetForTextEmbeddings, LargeDataSetForText or EmbeddedText which should be saved.
`dir_path`	`string` Path to the directory where the should model is stored.
`folder_name`	`string` Name of the folder where the files should be stored.

Value

Function does not return a value. It saves the model to disk.

No return value, called for side effects.

See Also

Other Saving and Loading: `load_from_disk()`

`set_transformers_logger` *Sets the level for logging information of the 'transformers' library*

Description

This function changes the level for logging information of the 'transformers' library. It influences the output printed to console for creating and training transformer models as well as TextEmbeddingModels.

Usage

```
set_transformers_logger(level = "ERROR")
```

Arguments

`level`	`string` Minimal level that should be printed to console. Four levels are available: INFO, WARNING, ERROR and DEBUG

Value

This function does not return anything. It is used for its side effects.

See Also

Other Installation and Configuration: `check_aif_py_modules()`, `get_recommended_py_versions()`, `install_aifeducation()`, `install_aifeducation_studio()`, `install_py_modules()`, `prepare_session()`, `update_aifeducation()`

---

start_aifeducation_studio

Aifeducation Studio

Description

Functions starts a shiny app that represents Aifeducation Studio.

Usage

```
start_aifeducation_studio(launch_browser = TRUE)
```

Arguments

`launch_browser`	`bool` If `TRUE` the system's default web browser is used for displaying the app.

Value

This function does nothing return. It is used to start a shiny app.

---

summarize_args_for_long_task

Summarize arguments from shiny input

Description

This function extracts the input relevant for a specific method of a specific class from shiny input.

In addition, it adds the path to all objects which can not be exported to another R session. These object must be loaded separately in the new session with the function `add_missing_args`. The paths are intended to be used with `shiny::ExtendedTask`. The final preparation of the arguments should be done with

The function can also be used to override the default value of a method or to add value for arguments which are not part of shiny input (use parameter `override_args`).

Usage

```
summarize_args_for_long_task(
  input,
  object_class,
  method = "configure",
  path_args = list(path_to_embeddings = NULL, path_to_textual_dataset = NULL,
   path_to_target_data = NULL, path_to_feature_extractor = NULL, destination_path =
    NULL, folder_name = NULL),
  override_args = list(),
  meta_args = list(py_environment_type = get_py_env_type(), py_env_name =
   get_py_env_name(), target_data_column = input$data_target_column, object_class =
    input$classifier_type)
)
```

Arguments

`input`	Shiny input.
`object_class`	`string` Class of the object.
`method`	`string` Method of the class for which the arguments should be extracted and prepared.
`path_args`	`list` List containing the path to object that can not be exported to another R session. These must be loaded in the session.
`override_args`	`list` List containing all arguments that should be set manually. The values override default values of the argument and values which are part of `input`.
`meta_args`	`list` List containing information that are not relevant for the arguments of the method but are necessary to set up the `shiny::ExtendedTask` correctly.

Value

Returns a named `list` with the following entries:

- args: Named `list` of all arguments necessary for the method of the class.
- path_args: Named `list` of all paths for loading the objects missing in args.
- meta_args: Named `list` of all arguments that are not part of the arguments of the method but which are necessary to set up the `shiny::ExtendedTask` correctly.

Note

Please not that all list are named list of the format (argument_name=values).

See Also

Other Utils Studio Developers: `add_missing_args()`, `create_data_embeddings_description()`, `long_load_target_data()`

`TEClassifierParallel` *Text embedding classifier with a neural net*

Description

Classification Type

This is a probability classifier that predicts a probability distribution for different classes/categories. This is the standard case most common in literature.

Parallel Core Architecture

This model is based on a parallel architecture. An input is passed to different types of layers separately. At the end the outputs are combined to create the final output of the whole model.

Transformer Encoder Layers

Description

The transformer encoder layers follow the structure of the encoder layers used in transformer models. A single layer is designed as described by Chollet, Kalinowski, and Allaire (2022, p. 373) with the exception that single components of the layers (such as the activation function, the kind of residual connection, the kind of normalization or the kind of attention) can be customized. All parameters with the prefix *tf_* can be used to configure this layer.

Feature Layer

Description

The feature layer is a dense layer that can be used to increase or decrease the number of features of the input data before passing the data into your model. The aim of this layer is to increase or reduce the complexity of the data for your model. The output size of this layer determines the number of features for all following layers. In the special case that the requested number of features equals the number of features of the text embeddings this layer is reduced to a dropout layer with masking capabilities. All parameters with the prefix *feat_* can be used to configure this layer.

Dense Layers

Description

A fully connected layer. The layer is applied to every step of a sequence. All parameters with the prefix *dense_* can be used to configure this layer.

Multiple N-Gram Layers

Description

This type of layer focuses on sub-sequence and performs an 1d convolutional operation. On a word and token level these sub-sequences can be interpreted as n-grams (Jacovi, Shalom & Goldberg 2018). The convolution is done across all features. The number of filters equals the number of features of the input tensor. Thus, the shape of the tensor is retained (Pham, Kruszewski & Boleda 2016).

The layer is able to consider multiple n-grams at the same time. In this case the convolution of the n-grams is done seprately and the resulting tensors are concatenated along the feature dimension. The number of filters for every n-gram is set to num_features/num_n-grams. Thus, the resulting tensor has the same shape as the input tensor.

Sub-sequences that are masked in the input are also masked in the output.

The output of this layer can be understand as the results of the n-gram filters. Stacking this layer allows the model to perform n-gram detection of n-grams (meta perspective). All parameters with the prefix *ng_conv_* can be used to configure this layer.

Recurrent Layers

Description

A regular recurrent layer either as Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM) layer. Uses PyTorchs implementation. All parameters with the prefix *rec_* can be used to configure this layer.

Merge Layer

Description

Layer for combining the output of different layers. All inputs must be sequential data of shape (Batch, Times, Features). First, pooling over time is applied extracting the minimal and/or maximal features. Second, the pooled tensors are combined by calculating their weighted sum. Different attention mechanism can be used to dynamically calculate the corresponding weights. This allows the model to decide which part of the data is most usefull. Finally, pooling over features is applied extracting a specific number of maximal and/or minimal features. A normalization of all input at the begining of the layer is possible. All parameters with the prefix *merge_* can be used to configure this layer.

Training and Prediction

For the creation and training of a classifier an object of class EmbeddedText or LargeDataSetForTextEmbeddings on the one hand and a factor on the other hand are necessary.

The object of class EmbeddedText or LargeDataSetForTextEmbeddings contains the numerical text representations (text embeddings) of the raw texts generated by an object of class TextEmbeddingModel. For supporting large data sets it is recommended to use LargeDataSetForTextEmbeddings instead of EmbeddedText.

The `factor` contains the classes/categories for every text. Missing values (unlabeled cases) are supported and can be used for pseudo labeling.

For predictions an object of class EmbeddedText or LargeDataSetForTextEmbeddings has to be used which was created with the same TextEmbeddingModel as for training.

Value

Returns a new object of this class ready for configuration or for loading a saved classifier.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `aifeducation::ClassifiersBasedOnTextEmbeddings` -> `aifeducation::TEClassifiersBasedOnRegular` -> `TEClassifierParallel`

Methods

Public methods:

- `TEClassifierParallel$configure()`

- TEClassifierParallel$clone()

Method `configure()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierParallel$configure(
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  shared_feat_layer = TRUE,
  feat_act_fct = "ELU",
  feat_size = 50L,
  feat_bias = TRUE,
  feat_dropout = 0,
  feat_parametrizations = "None",
  feat_normalization_type = "LayerNorm",
  ng_conv_act_fct = "ELU",
  ng_conv_n_layers = 1L,
  ng_conv_ks_min = 2L,
  ng_conv_ks_max = 4L,
  ng_conv_bias = FALSE,
  ng_conv_dropout = 0.1,
  ng_conv_parametrizations = "None",
  ng_conv_normalization_type = "LayerNorm",
  ng_conv_residual_type = "ResidualGate",
  dense_act_fct = "ELU",
  dense_n_layers = 1L,
  dense_dropout = 0.5,
  dense_bias = FALSE,
  dense_parametrizations = "None",
  dense_normalization_type = "LayerNorm",
  dense_residual_type = "ResidualGate",
  rec_act_fct = "Tanh",
  rec_n_layers = 1L,
  rec_type = "GRU",
  rec_bidirectional = FALSE,
  rec_dropout = 0.2,
  rec_bias = FALSE,
  rec_parametrizations = "None",
  rec_normalization_type = "LayerNorm",
  rec_residual_type = "ResidualGate",
  tf_act_fct = "ELU",
  tf_dense_dim = 50L,
  tf_n_layers = 1L,
  tf_dropout_rate_1 = 0.1,
  tf_dropout_rate_2 = 0.5,
  tf_attention_type = "MultiHead",
```

```
  tf_positional_type = "absolute",
  tf_num_heads = 1L,
  tf_bias = FALSE,
  tf_parametrizations = "None",
  tf_normalization_type = "LayerNorm",
  tf_residual_type = "ResidualGate",
  merge_attention_type = "multi_head",
  merge_num_heads = 1L,
  merge_normalization_type = "LayerNorm",
  merge_pooling_features = 50L,
  merge_pooling_type = "MinMax"
)
```

Arguments:

`name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`label` `string` Label for the new model. Here you can use free text. Allowed values: any

`text_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`feature_extractor` `TEFeatureExtractor` Object of class TEFeatureExtractor which should be used in order to reduce the number of dimensions of the text embeddings. If no feature extractor should be applied set `NULL`.

`target_levels` `vector` containing the levels (categories or classes) within the target data. Please note that order matters. For ordinal data please ensure that the levels are sorted correctly with later levels indicating a higher category/class. For nominal data the order does not matter.

`shared_feat_layer` `bool` If `TRUE` all streams use the same feature layer. If `FALSE` all streams use their own feature layer.

`feat_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`feat_size` `int` Number of neurons for each dense layer. Allowed values: `2 <= x`

`feat_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`feat_dropout` `double` determining the dropout for the dense projection of the feature layer. Allowed values: `0 <= x <= 0.6`

`feat_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`feat_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`ng_conv_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`ng_conv_n_layers` `int` determining how many times the n-gram layers should be added to the network. Allowed values: `0 <= x`

`ng_conv_ks_min` `int` determining the minimal window size for n-grams. Allowed values: `2 <= x`

ng_conv_ks_max int determining the maximal window size for n-grams. Allowed values: 2 <= x

ng_conv_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

ng_conv_dropout double determining the dropout for n-gram convolution layers. Allowed values: 0 <= x <= 0.6

ng_conv_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

ng_conv_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

ng_conv_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

dense_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

dense_n_layers int Number of dense layers. Allowed values: 0 <= x

dense_dropout double determining the dropout between dense layers. Allowed values: 0 <= x <= 0.6

dense_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

dense_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

dense_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

dense_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

rec_act_fct string Activation function for all layers. Allowed values: 'Tanh'

rec_n_layers int Number of recurrent layers. Allowed values: 0 <= x

rec_type string Type of the recurrent layers. rec_type='GRU' for Gated Recurrent Unit and rec_type='LSTM' for Long Short-Term Memory. Allowed values: 'GRU', 'LSTM'

rec_bidirectional bool If TRUE a bidirectional version of the recurrent layers is used.

rec_dropout double determining the dropout between recurrent layers. Allowed values: 0 <= x <= 0.6

rec_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

rec_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None'

rec_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

rec_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

tf_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

tf_dense_dim int determining the size of the projection layer within a each transformer encoder. Allowed values: 1 <= x

tf_n_layers int determining how many times the encoder should be added to the network. Allowed values: 0 <= x

`tf_dropout_rate_1` `double` determining the dropout after the attention mechanism within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_dropout_rate_2` `double` determining the dropout for the dense projection within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`tf_positional_type` `string` Type of processing positional information. Allowed values: 'None', 'absolute'

`tf_num_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`tf_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`tf_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`tf_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`tf_residual_type` `string` Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

`merge_attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`merge_num_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`merge_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`merge_pooling_features` `int` Number of features to be extracted at the end of the model. Allowed values: `1 <= x`

`merge_pooling_type` `string` Type of extracting intermediate features. Allowed values: 'Max', 'Min', 'MinMax'

Returns: Function does nothing return. It modifies the current object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Classification: `TEClassifierParallelPrototype`, `TEClassifierProtoNet`, `TEClassifierRegular`, `TEClassifierSequential`, `TEClassifierSequentialPrototype`

`TEClassifierParallelPrototype`
Text embedding classifier with a ProtoNet

Description

Classification Type

This object is a metric based classifer and represents in implementation of a prototypical network for few-shot learning as described by Snell, Swersky, and Zemel (2017). The network uses a multi way contrastive loss described by Zhang et al. (2019). The network learns to scale the metric as described by Oreshkin, Rodriguez, and Lacoste (2018).

Parallel Core Architecture

This model is based on a parallel architecture. An input is passed to different types of layers separately. At the end the outputs are combined to create the final output of the whole model.

Transformer Encoder Layers

Description

The transformer encoder layers follow the structure of the encoder layers used in transformer models. A single layer is designed as described by Chollet, Kalinowski, and Allaire (2022, p. 373) with the exception that single components of the layers (such as the activation function, the kind of residual connection, the kind of normalization or the kind of attention) can be customized. All parameters with the prefix *tf_* can be used to configure this layer.

Feature Layer

Description

The feature layer is a dense layer that can be used to increase or decrease the number of features of the input data before passing the data into your model. The aim of this layer is to increase or reduce the complexity of the data for your model. The output size of this layer determines the number of features for all following layers. In the special case that the requested number of features equals the number of features of the text embeddings this layer is reduced to a dropout layer with masking capabilities. All parameters with the prefix *feat_* can be used to configure this layer.

Dense Layers

Description

A fully connected layer. The layer is applied to every step of a sequence. All parameters with the prefix *dense_* can be used to configure this layer.

Multiple N-Gram Layers

Description

This type of layer focuses on sub-sequence and performs an 1d convolutional operation. On a word and token level these sub-sequences can be interpreted as n-grams (Jacovi, Shalom & Goldberg 2018). The convolution is done across all features. The number of filters equals the number of features of the input tensor. Thus, the shape of the tensor is retained (Pham, Kruszewski & Boleda 2016).

The layer is able to consider multiple n-grams at the same time. In this case the convolution of the n-grams is done seprately and the resulting tensors are concatenated along the feature dimension.

The number of filters for every n-gram is set to num_features/num_n-grams. Thus, the resulting tensor has the same shape as the input tensor.

Sub-sequences that are masked in the input are also masked in the output.

The output of this layer can be understand as the results of the n-gram filters. Stacking this layer allows the model to perform n-gram detection of n-grams (meta perspective). All parameters with the prefix *ng_conv_* can be used to configure this layer.

Recurrent Layers

Description

A regular recurrent layer either as Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM) layer. Uses PyTorchs implementation. All parameters with the prefix *rec_* can be used to configure this layer.

Merge Layer

Description

Layer for combining the output of different layers. All inputs must be sequential data of shape (Batch, Times, Features). First, pooling over time is applied extracting the minimal and/or maximal features. Second, the pooled tensors are combined by calculating their weighted sum. Different attention mechanism can be used to dynamically calculate the corresponding weights. This allows the model to decide which part of the data is most usefull. Finally, pooling over features is applied extracting a specific number of maximal and/or minimal features. A normalization of all input at the begining of the layer is possible. All parameters with the prefix *merge_* can be used to configure this layer.

Training and Prediction

For the creation and training of a classifier an object of class EmbeddedText or LargeDataSetForTextEmbeddings on the one hand and a factor on the other hand are necessary.

The object of class EmbeddedText or LargeDataSetForTextEmbeddings contains the numerical text representations (text embeddings) of the raw texts generated by an object of class TextEmbeddingModel. For supporting large data sets it is recommended to use LargeDataSetForTextEmbeddings instead of EmbeddedText.

The `factor` contains the classes/categories for every text. Missing values (unlabeled cases) are supported and can be used for pseudo labeling.

For predictions an object of class EmbeddedText or LargeDataSetForTextEmbeddings has to be used which was created with the same TextEmbeddingModel as for training.

Value

Returns a new object of this class ready for configuration or for loading a saved classifier.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `aifeducation::ClassifiersBasedOnTextEmbeddings` -> `aifeducation::TEClassifiersBasedOnProtoNet` -> `TEClassifierParallelPrototype`

Methods

Public methods:

- TEClassifierParallelPrototype$configure()
- TEClassifierParallelPrototype$clone()

Method `configure()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierParallelPrototype$configure(
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  metric_type = "Euclidean",
  shared_feat_layer = TRUE,
  feat_act_fct = "ELU",
  feat_size = 50L,
  feat_bias = TRUE,
  feat_dropout = 0,
  feat_parametrizations = "None",
  feat_normalization_type = "LayerNorm",
  ng_conv_act_fct = "ELU",
  ng_conv_n_layers = 1L,
  ng_conv_ks_min = 2L,
  ng_conv_ks_max = 4L,
  ng_conv_bias = FALSE,
  ng_conv_dropout = 0.1,
  ng_conv_parametrizations = "None",
  ng_conv_normalization_type = "LayerNorm",
  ng_conv_residual_type = "ResidualGate",
  dense_act_fct = "ELU",
  dense_n_layers = 1L,
  dense_dropout = 0.5,
  dense_bias = FALSE,
  dense_parametrizations = "None",
  dense_normalization_type = "LayerNorm",
  dense_residual_type = "ResidualGate",
  rec_act_fct = "Tanh",
  rec_n_layers = 1L,
  rec_type = "GRU",
  rec_bidirectional = FALSE,
  rec_dropout = 0.2,
  rec_bias = FALSE,
  rec_parametrizations = "None",
  rec_normalization_type = "LayerNorm",
  rec_residual_type = "ResidualGate",
  tf_act_fct = "ELU",
```

```
  tf_dense_dim = 50L,
  tf_n_layers = 1L,
  tf_dropout_rate_1 = 0.1,
  tf_dropout_rate_2 = 0.5,
  tf_attention_type = "MultiHead",
  tf_positional_type = "absolute",
  tf_num_heads = 1L,
  tf_bias = FALSE,
  tf_parametrizations = "None",
  tf_normalization_type = "LayerNorm",
  tf_residual_type = "ResidualGate",
  merge_attention_type = "multi_head",
  merge_num_heads = 1L,
  merge_normalization_type = "LayerNorm",
  merge_pooling_features = 50L,
  merge_pooling_type = "MinMax",
  embedding_dim = 2L
)
```

Arguments:

`name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`label` `string` Label for the new model. Here you can use free text. Allowed values: any

`text_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`feature_extractor` `TEFeatureExtractor` Object of class TEFeatureExtractor which should be used in order to reduce the number of dimensions of the text embeddings. If no feature extractor should be applied set `NULL`.

`target_levels` `vector` containing the levels (categories or classes) within the target data. Please note that order matters. For ordinal data please ensure that the levels are sorted correctly with later levels indicating a higher category/class. For nominal data the order does not matter.

`metric_type` `string` Type of metric used for calculating the distance. Allowed values: 'Euclidean'

`shared_feat_layer` `bool` If `TRUE` all streams use the same feature layer. If `FALSE` all streams use their own feature layer.

`feat_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`feat_size` `int` Number of neurons for each dense layer. Allowed values: `2 <= x`

`feat_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`feat_dropout` `double` determining the dropout for the dense projection of the feature layer. Allowed values: `0 <= x <= 0.6`

`feat_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`feat_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

ng_conv_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

ng_conv_n_layers int determining how many times the n-gram layers should be added to the network. Allowed values: 0 <= x

ng_conv_ks_min int determining the minimal window size for n-grams. Allowed values: 2 <= x

ng_conv_ks_max int determining the maximal window size for n-grams. Allowed values: 2 <= x

ng_conv_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

ng_conv_dropout double determining the dropout for n-gram convolution layers. Allowed values: 0 <= x <= 0.6

ng_conv_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

ng_conv_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

ng_conv_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

dense_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

dense_n_layers int Number of dense layers. Allowed values: 0 <= x

dense_dropout double determining the dropout between dense layers. Allowed values: 0 <= x <= 0.6

dense_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

dense_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

dense_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

dense_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

rec_act_fct string Activation function for all layers. Allowed values: 'Tanh'

rec_n_layers int Number of recurrent layers. Allowed values: 0 <= x

rec_type string Type of the recurrent layers. rec_type='GRU' for Gated Recurrent Unit and rec_type='LSTM' for Long Short-Term Memory. Allowed values: 'GRU', 'LSTM'

rec_bidirectional bool If TRUE a bidirectional version of the recurrent layers is used.

rec_dropout double determining the dropout between recurrent layers. Allowed values: 0 <= x <= 0.6

rec_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

rec_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None'

rec_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

rec_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

`tf_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`tf_dense_dim` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`tf_n_layers` `int` determining how many times the encoder should be added to the network. Allowed values: `0 <= x`

`tf_dropout_rate_1` `double` determining the dropout after the attention mechanism within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_dropout_rate_2` `double` determining the dropout for the dense projection within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`tf_positional_type` `string` Type of processing positional information. Allowed values: 'None', 'absolute'

`tf_num_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`tf_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`tf_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`tf_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`tf_residual_type` `string` Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

`merge_attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`merge_num_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`merge_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`merge_pooling_features` `int` Number of features to be extracted at the end of the model. Allowed values: `1 <= x`

`merge_pooling_type` `string` Type of extracting intermediate features. Allowed values: 'Max', 'Min', 'MinMax'

`embedding_dim` `int` determining the number of dimensions for the embedding. Allowed values: `2 <= x`

Returns: Function does nothing return. It modifies the current object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Oreshkin, B. N., Rodriguez, P. & Lacoste, A. (2018). TADAM: Task dependent adaptive metric for improved few-shot learning. https://doi.org/10.48550/arXiv.1805.10123

Snell, J., Swersky, K. & Zemel, R. S. (2017). Prototypical Networks for Few-shot Learning. https://doi.org/10.48550/arXiv.1703.05175

Zhang, X., Nie, J., Zong, L., Yu, H. & Liang, W. (2019). One Shot Learning with Margin. In Q. Yang, Z.-H. Zhou, Z. Gong, M.-L. Zhang & S.-J. Huang (Eds.), Lecture Notes in Computer Science. Advances in Knowledge Discovery and Data Mining (Vol. 11440, pp. 305–317). Springer International Publishing. https://doi.org/10.1007/978-3-030-16145-3_24

See Also

Other Classification: `TEClassifierParallel`, `TEClassifierProtoNet`, `TEClassifierRegular`, `TEClassifierSequential`, `TEClassifierSequentialPrototype`

`TEClassifierProtoNet` *Text embedding classifier with a ProtoNet*

Description

Abstract class for neural nets with 'pytorch'.

This class is **deprecated**. Please use an Object of class TEClassifierSequentialPrototype instead.

This object represents in implementation of a prototypical network for few-shot learning as described by Snell, Swersky, and Zemel (2017). The network uses a multi way contrastive loss described by Zhang et al. (2019). The network learns to scale the metric as described by Oreshkin, Rodriguez, and Lacoste (2018)

Value

Objects of this class are used for assigning texts to classes/categories. For the creation and training of a classifier an object of class EmbeddedText or LargeDataSetForTextEmbeddings and a `factor` are necessary. The object of class EmbeddedText or LargeDataSetForTextEmbeddings contains the numerical text representations (text embeddings) of the raw texts generated by an object of class TextEmbeddingModel. The `factor` contains the classes/categories for every text. Missing values (unlabeled cases) are supported. For predictions an object of class EmbeddedText or LargeDataSetForTextEmbeddings has to be used which was created with the same TextEmbeddingModel as for training.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `aifeducation::ClassifiersBasedOnTextEmbeddings` -> `aifeducation::TEClassifiersBasedOnProtoNet` -> `TEClassifierProtoNet`

Methods

Public methods:

- TEClassifierProtoNet$new()
- TEClassifierProtoNet$configure()
- TEClassifierProtoNet$embed()
- TEClassifierProtoNet$plot_embeddings()
- TEClassifierProtoNet$clone()

Method `new()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierProtoNet$new()
```

Returns: Returns an object of class TEClassifierProtoNet which is ready for configuration.

Method `configure()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierProtoNet$configure(
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  dense_size = 4L,
  dense_layers = 0L,
  rec_size = 4L,
  rec_layers = 2L,
  rec_type = "GRU",
  rec_bidirectional = FALSE,
  embedding_dim = 2L,
  self_attention_heads = 0L,
  intermediate_size = NULL,
  attention_type = "Fourier",
  add_pos_embedding = TRUE,
  act_fct = "ELU",
  parametrizations = "None",
  rec_dropout = 0.1,
  repeat_encoder = 1L,
  dense_dropout = 0.4,
  encoder_dropout = 0.1
)
```

Arguments:

`name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`label` `string` Label for the new model. Here you can use free text. Allowed values: any

`text_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`feature_extractor` `TEFeatureExtractor` Object of class TEFeatureExtractor which should be used in order to reduce the number of dimensions of the text embeddings. If no feature extractor should be applied set `NULL`.

`target_levels` `vector` containing the levels (categories or classes) within the target data. Please note that order matters. For ordinal data please ensure that the levels are sorted correctly with later levels indicating a higher category/class. For nominal data the order does not matter.

`dense_size` `int` Number of neurons for each dense layer. Allowed values: `1 <= x`

`dense_layers` `int` Number of dense layers. Allowed values: `0 <= x`

`rec_size` `int` Number of neurons for each recurrent layer. Allowed values: `1 <= x`

`rec_layers` `int` Number of recurrent layers. Allowed values: `0 <= x`

`rec_type` `string` Type of the recurrent layers. `rec_type='GRU'` for Gated Recurrent Unit and `rec_type='LSTM'` for Long Short-Term Memory. Allowed values: 'GRU', 'LSTM'

`rec_bidirectional` `bool` If `TRUE` a bidirectional version of the recurrent layers is used.

`embedding_dim` `int` determining the number of dimensions for the embedding. Allowed values: `2 <= x`

`self_attention_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`add_pos_embedding` `bool` `TRUE` if positional embedding should be used.

`act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`rec_dropout` `double` determining the dropout between recurrent layers. Allowed values: `0 <= x <= 0.6`

`repeat_encoder` `int` determining how many times the encoder should be added to the network. Allowed values: `0 <= x`

`dense_dropout` `double` determining the dropout between dense layers. Allowed values: `0 <= x <= 0.6`

`encoder_dropout` `double` determining the dropout for the dense projection within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

Method `embed()`**:** Method for embedding documents. Please do not confuse this type of embeddings with the embeddings of texts created by an object of class TextEmbeddingModel. These embeddings embed documents according to their similarity to specific classes.

Usage:

```
TEClassifierProtoNet$embed(embeddings_q = NULL, batch_size = 32L)
```

Arguments:

`embeddings_q` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all cases which should be embedded into the classification space.

`batch_size` `int` batch size.

Returns: Returns a `list` containing the following elements

- `embeddings_q`: embeddings for the cases (query sample).
- `embeddings_prototypes`: embeddings of the prototypes which were learned during training. They represents the center for the different classes.

Method `plot_embeddings()`**:** Method for creating a plot to visualize embeddings and their corresponding centers (prototypes).

Usage:

```
TEClassifierProtoNet$plot_embeddings(
  embeddings_q,
  classes_q = NULL,
  batch_size = 12L,
  alpha = 0.5,
  size_points = 3L,
  size_points_prototypes = 8L,
  inc_unlabeled = TRUE
)
```

Arguments:

`embeddings_q` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all cases which should be embedded into the classification space.

`classes_q` Named `factor` containg the true classes for every case. Please note that the names must match the names/ids in `embeddings_q`.

`batch_size` `int` batch size.

`alpha` `float` Value indicating how transparent the points should be (important if many points overlap). Does not apply to points representing prototypes.

`size_points` `int` Size of the points excluding the points for prototypes.

`size_points_prototypes` `int` Size of points representing prototypes.

`inc_unlabeled` `bool` If `TRUE` plot includes unlabeled cases as data points.

Returns: Returns a plot of class `ggplot`visualizing embeddings.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

This model requires `pad_value=0`. If this condition is not met the padding value is switched automatically.

References

Oreshkin, B. N., Rodriguez, P. & Lacoste, A. (2018). TADAM: Task dependent adaptive metric for improved few-shot learning. https://doi.org/10.48550/arXiv.1805.10123

Snell, J., Swersky, K. & Zemel, R. S. (2017). Prototypical Networks for Few-shot Learning. https://doi.org/10.48550/arXiv.1703.05175

Zhang, X., Nie, J., Zong, L., Yu, H. & Liang, W. (2019). One Shot Learning with Margin. In Q. Yang, Z.-H. Zhou, Z. Gong, M.-L. Zhang & S.-J. Huang (Eds.), Lecture Notes in Computer Science. Advances in Knowledge Discovery and Data Mining (Vol. 11440, pp. 305–317). Springer International Publishing. https://doi.org/10.1007/978-3-030-16145-3_24

See Also

Other Classification: TEClassifierParallel, TEClassifierParallelPrototype, TEClassifierRegular, TEClassifierSequential, TEClassifierSequentialPrototype

---

TEClassifierRegular *Text embedding classifier with a neural net*

Description

Abstract class for neural nets with 'pytorch'.

This class is **deprecated**. Please use an Object of class TEClassifierSequential instead.

Value

Objects of this class are used for assigning texts to classes/categories. For the creation and training of a classifier an object of class EmbeddedText or LargeDataSetForTextEmbeddings on the one hand and a factor on the other hand are necessary.

The object of class EmbeddedText or LargeDataSetForTextEmbeddings contains the numerical text representations (text embeddings) of the raw texts generated by an object of class TextEmbeddingModel. For supporting large data sets it is recommended to use LargeDataSetForTextEmbeddings instead of EmbeddedText.

The `factor` contains the classes/categories for every text. Missing values (unlabeled cases) are supported and can be used for pseudo labeling.

For predictions an object of class EmbeddedText or LargeDataSetForTextEmbeddings has to be used which was created with the same TextEmbeddingModel as for training.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `aifeducation::ClassifiersBasedOnTextEmbeddings` -> `aifeducation::TEClassifiersBasedOnRegular` -> `TEClassifierRegular`

Methods

Public methods:

- `TEClassifierRegular$new()`
- `TEClassifierRegular$configure()`

- TEClassifierRegular$clone()

Method `new()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierRegular$new()
```

Returns: Returns an object of class TEClassifierRegular which is ready for configuration.

Method `configure()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierRegular$configure(
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  bias = TRUE,
  dense_size = 4L,
  dense_layers = 0L,
  rec_size = 4L,
  rec_layers = 2L,
  rec_type = "GRU",
  rec_bidirectional = FALSE,
  self_attention_heads = 0L,
  intermediate_size = NULL,
  attention_type = "Fourier",
  add_pos_embedding = TRUE,
  act_fct = "ELU",
  parametrizations = "None",
  rec_dropout = 0.1,
  repeat_encoder = 1L,
  dense_dropout = 0.4,
  encoder_dropout = 0.1
)
```

Arguments:

`name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`label` `string` Label for the new model. Here you can use free text. Allowed values: any

`text_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`feature_extractor` `TEFeatureExtractor` Object of class TEFeatureExtractor which should be used in order to reduce the number of dimensions of the text embeddings. If no feature extractor should be applied set `NULL`.

`target_levels` `vector` containing the levels (categories or classes) within the target data. Please note that order matters. For ordinal data please ensure that the levels are sorted correctly with later levels indicating a higher category/class. For nominal data the order does not matter.

`bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`dense_size` `int` Number of neurons for each dense layer. Allowed values: `1 <= x`

`dense_layers` `int` Number of dense layers. Allowed values: `0 <= x`

`rec_size` `int` Number of neurons for each recurrent layer. Allowed values: `1 <= x`

`rec_layers` `int` Number of recurrent layers. Allowed values: `0 <= x`

`rec_type` `string` Type of the recurrent layers. `rec_type='GRU'` for Gated Recurrent Unit and `rec_type='LSTM'` for Long Short-Term Memory. Allowed values: 'GRU', 'LSTM'

`rec_bidirectional` `bool` If `TRUE` a bidirectional version of the recurrent layers is used.

`self_attention_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`add_pos_embedding` `bool` `TRUE` if positional embedding should be used.

`act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`rec_dropout` `double` determining the dropout between recurrent layers. Allowed values: `0 <= x <= 0.6`

`repeat_encoder` `int` determining how many times the encoder should be added to the network. Allowed values: `0 <= x`

`dense_dropout` `double` determining the dropout between dense layers. Allowed values: `0 <= x <= 0.6`

`encoder_dropout` `double` determining the dropout for the dense projection within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

Returns: Returns an object of class TEClassifierRegular which is ready for training.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

This model requires `pad_value=0`. If this condition is not met the padding value is switched automatically.

See Also

Other Classification: `TEClassifierParallel`, `TEClassifierParallelPrototype`, `TEClassifierProtoNet`, `TEClassifierSequential`, `TEClassifierSequentialPrototype`

`TEClassifiersBasedOnProtoNet` *Base class for classifiers relying on numerical representations of texts instead of words that use the architecture of Protonets and its corresponding training techniques.*

Description

Base class for classifiers relying on EmbeddedText or LargeDataSetForTextEmbeddings as input which use the architecture of Protonets and its corresponding training techniques.

Objects of this class containing fields and methods used in several other classes in 'AI for Education'.

This class is **not** designed for a direct application and should only be used by developers.

Value

A new object of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `aifeducation::ClassifiersBasedOnTextEmbeddings` -> `TEClassifiersBasedOnProtoNet`

Methods

Public methods:

- `TEClassifiersBasedOnProtoNet$train()`
- `TEClassifiersBasedOnProtoNet$predict_with_samples()`
- `TEClassifiersBasedOnProtoNet$embed()`
- `TEClassifiersBasedOnProtoNet$get_metric_scale_factor()`
- `TEClassifiersBasedOnProtoNet$plot_embeddings()`
- `TEClassifiersBasedOnProtoNet$clone()`

Method `train()`**:** Method for training a neural net.

Training includes a routine for early stopping. In the case that loss<0.0001 and Accuracy=1.00 and Average Iota=1.00 training stops. The history uses the values of the last trained epoch for the remaining epochs.

After training the model with the best values for Average Iota, Accuracy, and Loss on the validation data set is used as the final model.

Usage:

```
TEClassifiersBasedOnProtoNet$train(
  data_embeddings = NULL,
  data_targets = NULL,
  data_folds = 5L,
  data_val_size = 0.25,
```

```
  loss_pt_fct_name = "MultiWayContrastiveLoss",
  use_sc = FALSE,
  sc_method = "knnor",
  sc_min_k = 1L,
  sc_max_k = 10L,
  use_pl = FALSE,
  pl_max_steps = 3L,
  pl_max = 1,
  pl_anchor = 1,
  pl_min = 0,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15L,
  sustain_log_level = "warning",
  epochs = 40L,
  batch_size = 35L,
  Ns = 5L,
  Nq = 3L,
  loss_alpha = 0.5,
  loss_margin = 0.05,
  sampling_separate = FALSE,
  sampling_shuffle = TRUE,
  trace = TRUE,
  ml_trace = 1L,
  log_dir = NULL,
  log_write_interval = 10L,
  n_cores = auto_n_cores(),
  lr_rate = 0.001,
  lr_warm_up_ratio = 0.02,
  optimizer = "AdamW"
)
```

Arguments:

`data_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`data_targets` `factor` containing the labels for cases stored in embeddings. Factor must be named and has to use the same names as used in in the embeddings. .

`data_folds` `int` determining the number of cross-fold samples. Allowed values: `1 <= x`

`data_val_size` `double` between 0 and 1, indicating the proportion of cases which should be used for the validation sample during the estimation of the model. The remaining cases are part of the training data. Allowed values: `0 < x < 1`

`loss_pt_fct_name` `string` Name of the loss function to use during training. Allowed values: 'MultiWayContrastiveLoss'

`use_sc` `bool` `TRUE` if the estimation should integrate synthetic cases. `FALSE` if not.

`sc_method` `string` containing the method for generating synthetic cases. Allowed values: 'knnor'

`sc_min_k` `int` determining the minimal number of k which is used for creating synthetic units. Allowed values: `1 <= x`

sc_max_k int determining the maximal number of k which is used for creating synthetic units. Allowed values: 1 <= x

use_pl bool TRUE if the estimation should integrate pseudo-labeling. FALSE if not.

pl_max_steps int determining the maximum number of steps during pseudo-labeling. Allowed values: 1 <= x

pl_max double setting the maximal level of confidence for considering a case for pseudo-labeling. Allowed values: 0 < x <= 1

pl_anchor double indicating the reference point for sorting the new cases of every label. Allowed values: 0 <= x <= 1

pl_min double setting the mnimal level of confidence for considering a case for pseudo-labeling. Allowed values: 0 <= x < 1

sustain_track bool If TRUE energy consumption is tracked during training via the python library 'codecarbon'.

sustain_iso_code string ISO code (Alpha-3-Code) for the country. This variable must be set if sustainability should be tracked. A list can be found on Wikipedia: https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes. Allowed values: any

sustain_region string Region within a country. Only available for USA and Canada See the documentation of codecarbon for more information. https://mlco2.github.io/codecarbon/parameters.html Allowed values: any

sustain_interval int Interval in seconds for measuring power usage. Allowed values: 1 <= x

sustain_log_level

epochs int Number of training epochs. Allowed values: 1 <= x

batch_size int Size of the batches for training. Allowed values: 1 <= x

Ns int Number of cases for every class in the sample. Allowed values: 1 <= x

Nq int Number of cases for every class in the query. Allowed values: 1 <= x

loss_alpha double Value between 0 and 1 indicating how strong the loss should focus on pulling cases to its corresponding prototypes or pushing cases away from other prototypes. The higher the value the more the loss concentrates on pulling cases to its corresponding prototypes. Allowed values: 0 <= x <= 1

loss_margin double Value greater 0 indicating the minimal distance of every case from prototypes of other classes. Please note that in contrast to the orginal work by Zhang et al. (2019) this implementation reaches better performance if the margin is a magnitude lower (e.g. 0.05 instead of 0.5). Allowed values: 0 <= x <= 1

sampling_separate bool If TRUE the cases for every class are divided into a data set for sample and for query. These are never mixed. If TRUE sample and query cases are drawn from the same data pool. That is, a case can be part of sample in one epoch and in another epoch it can be part of query. It is ensured that a case is never part of sample and query at the same time. In addition, it is ensured that every cases exists only once during a training step.

sampling_shuffle bool if TRUE cases a randomly drawn from the data during every step. If FALSE the cases are not shuffled.

trace bool TRUE if information about the estimation phase should be printed to the console.

ml_trace int ml_trace=0 does not print any information about the training process from pytorch on the console. Allowed values: 0 <= x <= 1

log_dir string Path to the directory where the log files should be saved. If no logging is desired set this argument to NULL. Allowed values: any

`log_write_interval` `int` Time in seconds determining the interval in which the logger should try to update the log files. Only relevant if `log_dir` is not `NULL`. Allowed values: `1 <= x`

`n_cores` `int` Number of cores which should be used during the calculation of synthetic cases. Only relevant if `use_sc=TRUE`. Allowed values: `1 <= x`

`lr_rate` `double` Initial learning rate for the training. Allowed values: `0 < x <= 1`

`lr_warm_up_ratio` `double` Number of epochs used for warm up. Allowed values: `0 < x < 0.5`

`optimizer` `string` determining the optimizer used for training. Allowed values: 'Adam', 'RMSprop', 'AdamW', 'SGD'

`loss_balance_class_weights` `bool` If `TRUE` class weights are generated based on the frequencies of the training data with the method Inverse Class Frequency. If `FALSE` each class has the weight 1.

`loss_balance_sequence_length` `bool` If `TRUE` sample weights are generated for the length of sequences based on the frequencies of the training data with the method Inverse Class Frequency. If `FALSE` each sequences length has the weight 1.

Details:

- `sc_max_k`: All values from sc_min_k up to sc_max_k are successively used. If the number of `sc_max_k` is too high, the value is reduced to a number that allows the calculating of synthetic units.
- `pl_anchor`: With the help of this value, the new cases are sorted. For this aim, the distance from the anchor is calculated and all cases are arranged into an ascending order.

Returns: Function does not return a value. It changes the object into a trained classifier.

Method `predict_with_samples()`**:** Method for predicting the class of given data (query) based on provided examples (sample).

Usage:

```
TEClassifiersBasedOnProtoNet$predict_with_samples(
  newdata,
  batch_size = 32L,
  ml_trace = 1L,
  embeddings_s = NULL,
  classes_s = NULL
)
```

Arguments:

`newdata` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all cases which should be predicted. They form the query set.

`batch_size` `int` batch size.

`ml_trace` `int` `ml_trace=0` does not print any information about the training process from pytorch on the console. Allowed values: `0 <= x <= 1`

`embeddings_s` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all reference examples. They form the sample set.

`classes_s` Named `factor` containing the classes for every case within `embeddings_s`.

Returns: Returns a `data.frame` containing the predictions and the probabilities of the different labels for each case.

Method `embed()`: Method for embedding documents. Please do not confuse this type of embeddings with the embeddings of texts created by an object of class TextEmbeddingModel. These embeddings embed documents according to their similarity to specific classes.

Usage:

```
TEClassifiersBasedOnProtoNet$embed(
  embeddings_q = NULL,
  embeddings_s = NULL,
  classes_s = NULL,
  batch_size = 32L,
  ml_trace = 1L
)
```

Arguments:

`embeddings_q` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all cases which should be embedded into the classification space.

`embeddings_s` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all reference examples. They form the sample set. If set to `NULL` the trained prototypes are used.

`classes_s` Named `factor` containing the classes for every case within `embeddings_s`. If set to `NULL` the trained prototypes are used.

`batch_size` `int` batch size.

`ml_trace` `int` `ml_trace=0` does not print any information about the training process from pytorch on the console. Allowed values: `0 <= x <= 1`

Returns: Returns a `list` containing the following elements

- `embeddings_q`: embeddings for the cases (query sample).
- `distances_q`: `matrix` containing the distance of every query case to every prototype.
- `embeddings_prototypes`: embeddings of the prototypes which were learned during training. They represents the center for the different classes.

Method `get_metric_scale_factor()`: Method returns the scaling factor of the metric.

Usage:

```
TEClassifiersBasedOnProtoNet$get_metric_scale_factor()
```

Returns: Returns the scaling factor of the metric as `float`.

Method `plot_embeddings()`: Method for creating a plot to visualize embeddings and their corresponding centers (prototypes).

Usage:

```
TEClassifiersBasedOnProtoNet$plot_embeddings(
  embeddings_q,
  classes_q = NULL,
  embeddings_s = NULL,
  classes_s = NULL,
  batch_size = 12L,
  alpha = 0.5,
  size_points = 3L,
  size_points_prototypes = 8L,
```

```
  inc_unlabeled = TRUE,
  inc_margin = TRUE
)
```

Arguments:

`embeddings_q` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all cases which should be embedded into the classification space.

`classes_q` Named `factor` containg the true classes for every case. Please note that the names must match the names/ids in `embeddings_q`.

`embeddings_s` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings for all reference examples. They form the sample set. If set to `NULL` the trained prototypes are used.

`classes_s` Named `factor` containing the classes for every case within `embeddings_s`. If set to `NULL` the trained prototypes are used.

`batch_size` `int` batch size.

`alpha` `float` Value indicating how transparent the points should be (important if many points overlap). Does not apply to points representing prototypes.

`size_points` `int` Size of the points excluding the points for prototypes.

`size_points_prototypes` `int` Size of points representing prototypes.

`inc_unlabeled` `bool` If `TRUE` plot includes unlabeled cases as data points.

`inc_margin` `bool` If `TRUE` plot includes the margin around every prototype. Adding margin requires a trained model. If the model is not trained this argument is treated as set to `FALSE`.

Returns: Returns a plot of class `ggplot`visualizing embeddings.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `BaseModelCore`, `ClassifiersBasedOnTextEmbeddings`, `DataManagerClassifier`, `LargeDataSetBase`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnRegular`, `TokenizerBase`

TEClassifiersBasedOnRegular

Base class for regular classifiers relying on EmbeddedText or LargeDataSetForTextEmbeddings as input

---

Description

Abstract class for all regular classifiers that use numerical representations of texts instead of words.

Objects of this class containing fields and methods used in several other classes in 'AI for Education'.

This class is **not** designed for a direct application and should only be used by developers.

Value

A new object of this class.

Super classes

aifeducation::AIFEMaster -> aifeducation::AIFEBaseModel -> aifeducation::ModelsBasedOnTextEmbeddings -> aifeducation::ClassifiersBasedOnTextEmbeddings -> TEClassifiersBasedOnRegular

Methods

Public methods:

- TEClassifiersBasedOnRegular$train()
- TEClassifiersBasedOnRegular$clone()

Method `train()`**:** Method for training a neural net.

Training includes a routine for early stopping. In the case that loss<0.0001 and Accuracy=1.00 and Average Iota=1.00 training stops. The history uses the values of the last trained epoch for the remaining epochs.

After training the model with the best values for Average Iota, Accuracy, and Loss on the validation data set is used as the final model.

Usage:

```
TEClassifiersBasedOnRegular$train(
  data_embeddings = NULL,
  data_targets = NULL,
  data_folds = 5L,
  data_val_size = 0.25,
  loss_balance_class_weights = TRUE,
  loss_balance_sequence_length = TRUE,
  loss_cls_fct_name = "FocalLoss",
  use_sc = FALSE,
  sc_method = "knnor",
  sc_min_k = 1L,
  sc_max_k = 10L,
  use_pl = FALSE,
  pl_max_steps = 3L,
  pl_max = 1,
  pl_anchor = 1,
  pl_min = 0,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15L,
  sustain_log_level = "warning",
  epochs = 40L,
  batch_size = 32L,
  trace = TRUE,
  ml_trace = 1L,
```

```
  log_dir = NULL,
  log_write_interval = 10L,
  n_cores = auto_n_cores(),
  lr_rate = 0.001,
  lr_warm_up_ratio = 0.02,
  optimizer = "AdamW"
)
```

Arguments:

`data_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`data_targets` `factor` containing the labels for cases stored in embeddings. Factor must be named and has to use the same names as used in in the embeddings. .

`data_folds` `int` determining the number of cross-fold samples. Allowed values: `1 <= x`

`data_val_size` `double` between 0 and 1, indicating the proportion of cases which should be used for the validation sample during the estimation of the model. The remaining cases are part of the training data. Allowed values: `0 < x < 1`

`loss_balance_class_weights` `bool` If `TRUE` class weights are generated based on the frequencies of the training data with the method Inverse Class Frequency. If `FALSE` each class has the weight 1.

`loss_balance_sequence_length` `bool` If `TRUE` sample weights are generated for the length of sequences based on the frequencies of the training data with the method Inverse Class Frequency. If `FALSE` each sequences length has the weight 1.

`loss_cls_fct_name` `string` Name of the loss function to use during training. Allowed values: 'FocalLoss', 'CrossEntropyLoss'

`use_sc` `bool` `TRUE` if the estimation should integrate synthetic cases. `FALSE` if not.

`sc_method` `string` containing the method for generating synthetic cases. Allowed values: 'knnor'

`sc_min_k` `int` determining the minimal number of k which is used for creating synthetic units. Allowed values: `1 <= x`

`sc_max_k` `int` determining the maximal number of k which is used for creating synthetic units. Allowed values: `1 <= x`

`use_pl` `bool` `TRUE` if the estimation should integrate pseudo-labeling. `FALSE` if not.

`pl_max_steps` `int` determining the maximum number of steps during pseudo-labeling. Allowed values: `1 <= x`

`pl_max` `double` setting the maximal level of confidence for considering a case for pseudo-labeling. Allowed values: `0 < x <= 1`

`pl_anchor` `double` indicating the reference point for sorting the new cases of every label. Allowed values: `0 <= x <= 1`

`pl_min` `double` setting the mnimal level of confidence for considering a case for pseudo-labeling. Allowed values: `0 <= x < 1`

`sustain_track` `bool` If `TRUE` energy consumption is tracked during training via the python library 'codecarbon'.

`sustain_iso_code` `string` ISO code (Alpha-3-Code) for the country. This variable must be set if sustainability should be tracked. A list can be found on Wikipedia: https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes. Allowed values: any

`sustain_region` `string` Region within a country. Only available for USA and Canada See the documentation of codecarbon for more information. https://mlco2.github.io/codecarbon/parameters.html Allowed values: any

`sustain_interval` `int` Interval in seconds for measuring power usage. Allowed values: `1 <= x`

`sustain_log_level`

`epochs` `int` Number of training epochs. Allowed values: `1 <= x`

`batch_size` `int` Size of the batches for training. Allowed values: `1 <= x`

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

`ml_trace` `int` `ml_trace=0` does not print any information about the training process from pytorch on the console. Allowed values: `0 <= x <= 1`

`log_dir` `string` Path to the directory where the log files should be saved. If no logging is desired set this argument to `NULL`. Allowed values: any

`log_write_interval` `int` Time in seconds determining the interval in which the logger should try to update the log files. Only relevant if `log_dir` is not `NULL`. Allowed values: `1 <= x`

`n_cores` `int` Number of cores which should be used during the calculation of synthetic cases. Only relevant if `use_sc=TRUE`. Allowed values: `1 <= x`

`lr_rate` `double` Initial learning rate for the training. Allowed values: `0 < x <= 1`

`lr_warm_up_ratio` `double` Number of epochs used for warm up. Allowed values: `0 < x < 0.5`

`optimizer` `string` determining the optimizer used for training. Allowed values: 'Adam', 'RMSprop', 'AdamW', 'SGD'

Details:

- `sc_max_k`: All values from sc_min_k up to sc_max_k are successively used. If the number of sc_max_k is too high, the value is reduced to a number that allows the calculating of synthetic units.
- `pl_anchor`: With the help of this value, the new cases are sorted. For this aim, the distance from the anchor is calculated and all cases are arranged into an ascending order.

Returns: Function does not return a value. It changes the object into a trained classifier.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `BaseModelCore`, `ClassifiersBasedOnTextEmbeddings`, `DataManagerClassifier`, `LargeDataSetBase`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnProtoNet`, `TokenizerBase`

`TEClassifierSequential` *Text embedding classifier with a neural net*

Description

Classification Type

This is a probability classifier that predicts a probability distribution for different classes/categories. This is the standard case most common in literature.

Sequential Core Architecture

This model is based on a sequential architecture. The input is passed to a specific number of layers step by step. All layers are grouped by their kind into stacks.

Transformer Encoder Layers

Description

The transformer encoder layers follow the structure of the encoder layers used in transformer models. A single layer is designed as described by Chollet, Kalinowski, and Allaire (2022, p. 373) with the exception that single components of the layers (such as the activation function, the kind of residual connection, the kind of normalization or the kind of attention) can be customized. All parameters with the prefix *tf_* can be used to configure this layer.

Feature Layer

Description

The feature layer is a dense layer that can be used to increase or decrease the number of features of the input data before passing the data into your model. The aim of this layer is to increase or reduce the complexity of the data for your model. The output size of this layer determines the number of features for all following layers. In the special case that the requested number of features equals the number of features of the text embeddings this layer is reduced to a dropout layer with masking capabilities. All parameters with the prefix *feat_* can be used to configure this layer.

Dense Layers

Description

A fully connected layer. The layer is applied to every step of a sequence. All parameters with the prefix *dense_* can be used to configure this layer.

Multiple N-Gram Layers

Description

This type of layer focuses on sub-sequence and performs an 1d convolutional operation. On a word and token level these sub-sequences can be interpreted as n-grams (Jacovi, Shalom & Goldberg 2018). The convolution is done across all features. The number of filters equals the number of features of the input tensor. Thus, the shape of the tensor is retained (Pham, Kruszewski & Boleda 2016).

The layer is able to consider multiple n-grams at the same time. In this case the convolution of the n-grams is done seprately and the resulting tensors are concatenated along the feature dimension. The number of filters for every n-gram is set to num_features/num_n-grams. Thus, the resulting tensor has the same shape as the input tensor.

Sub-sequences that are masked in the input are also masked in the output.

The output of this layer can be understand as the results of the n-gram filters. Stacking this layer allows the model to perform n-gram detection of n-grams (meta perspective). All parameters with the prefix *ng_conv_* can be used to configure this layer.

Recurrent Layers

Description

A regular recurrent layer either as Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM) layer. Uses PyTorchs implementation. All parameters with the prefix *rec_* can be used to configure this layer.

Classifiction Pooling Layer

Description

Layer transforms sequences into a lower dimensional space that can be passed to dense layers. It performs two types of pooling. First, it extractes features across the time dimension selecting the maximal and/or minimal features. Second, it performs pooling over the remaining features selecting a speficifc number of the heighest and/or lowest features.

In the case of selecting the minmal *and* maximal features at the same time the minmal features are concatenated to the tensor of the maximal features resulting the in the shape $(Batch, Times, 2*Features)$ at the end of the first step. In the second step the number of requested features is halved. The first half is used for the maximal features and the second for the minimal features. All parameters with the prefix *cls_pooling_* can be used to configure this layer.

Training and Prediction

For the creation and training of a classifier an object of class EmbeddedText or LargeDataSetForTextEmbeddings on the one hand and a factor on the other hand are necessary.

The object of class EmbeddedText or LargeDataSetForTextEmbeddings contains the numerical text representations (text embeddings) of the raw texts generated by an object of class TextEmbeddingModel. For supporting large data sets it is recommended to use LargeDataSetForTextEmbeddings instead of EmbeddedText.

The `factor` contains the classes/categories for every text. Missing values (unlabeled cases) are supported and can be used for pseudo labeling.

For predictions an object of class EmbeddedText or LargeDataSetForTextEmbeddings has to be used which was created with the same TextEmbeddingModel as for training.

Value

Returns a new object of this class ready for configuration or for loading a saved classifier.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `aifeducation::ClassifiersBasedOnTextEmbeddings` -> `aifeducation::TEClassifiersBasedOnRegular` -> `TEClassifierSequential`

Methods

Public methods:

- TEClassifierSequential$configure()
- TEClassifierSequential$clone()

Method `configure()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierSequential$configure(
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  skip_connection_type = "ResidualGate",
  cls_pooling_features = NULL,
  cls_pooling_type = "MinMax",
  feat_act_fct = "ELU",
  feat_size = 50L,
  feat_bias = TRUE,
  feat_dropout = 0,
  feat_parametrizations = "None",
  feat_normalization_type = "LayerNorm",
  ng_conv_act_fct = "ELU",
  ng_conv_n_layers = 1L,
  ng_conv_ks_min = 2L,
  ng_conv_ks_max = 4L,
  ng_conv_bias = FALSE,
  ng_conv_dropout = 0.1,
  ng_conv_parametrizations = "None",
  ng_conv_normalization_type = "LayerNorm",
  ng_conv_residual_type = "ResidualGate",
  dense_act_fct = "ELU",
  dense_n_layers = 1,
  dense_dropout = 0.5,
  dense_bias = FALSE,
  dense_parametrizations = "None",
  dense_normalization_type = "LayerNorm",
  dense_residual_type = "ResidualGate",
  rec_act_fct = "Tanh",
  rec_n_layers = 1L,
  rec_type = "GRU",
  rec_bidirectional = FALSE,
  rec_dropout = 0.2,
  rec_bias = FALSE,
  rec_parametrizations = "None",
  rec_normalization_type = "LayerNorm",
  rec_residual_type = "ResidualGate",
```

```
  tf_act_fct = "ELU",
  tf_dense_dim = 50L,
  tf_n_layers = 1L,
  tf_dropout_rate_1 = 0.1,
  tf_dropout_rate_2 = 0.5,
  tf_attention_type = "MultiHead",
  tf_positional_type = "absolute",
  tf_num_heads = 1,
  tf_bias = FALSE,
  tf_parametrizations = "None",
  tf_normalization_type = "LayerNorm",
  tf_residual_type = "ResidualGate"
)
```

Arguments:

`name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`label` `string` Label for the new model. Here you can use free text. Allowed values: any

`text_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`feature_extractor` `TEFeatureExtractor` Object of class TEFeatureExtractor which should be used in order to reduce the number of dimensions of the text embeddings. If no feature extractor should be applied set `NULL`.

`target_levels` `vector` containing the levels (categories or classes) within the target data. Please note that order matters. For ordinal data please ensure that the levels are sorted correctly with later levels indicating a higher category/class. For nominal data the order does not matter.

`skip_connection_type` `string` Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

`cls_pooling_features` `int` Number of features to be extracted at the end of the model. Allowed values: `1 <= x`

`cls_pooling_type` `string` Type of extracting intermediate features. Allowed values: 'Max', 'Min', 'MinMax'

`feat_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`feat_size` `int` Number of neurons for each dense layer. Allowed values: `2 <= x`

`feat_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`feat_dropout` `double` determining the dropout for the dense projection of the feature layer. Allowed values: `0 <= x <= 0.6`

`feat_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`feat_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`ng_conv_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

ng_conv_n_layers int determining how many times the n-gram layers should be added to the network. Allowed values: 0 <= x

ng_conv_ks_min int determining the minimal window size for n-grams. Allowed values: 2 <= x

ng_conv_ks_max int determining the maximal window size for n-grams. Allowed values: 2 <= x

ng_conv_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

ng_conv_dropout double determining the dropout for n-gram convolution layers. Allowed values: 0 <= x <= 0.6

ng_conv_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

ng_conv_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

ng_conv_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

dense_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

dense_n_layers int Number of dense layers. Allowed values: 0 <= x

dense_dropout double determining the dropout between dense layers. Allowed values: 0 <= x <= 0.6

dense_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

dense_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

dense_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

dense_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

rec_act_fct string Activation function for all layers. Allowed values: 'Tanh'

rec_n_layers int Number of recurrent layers. Allowed values: 0 <= x

rec_type string Type of the recurrent layers. rec_type='GRU' for Gated Recurrent Unit and rec_type='LSTM' for Long Short-Term Memory. Allowed values: 'GRU', 'LSTM'

rec_bidirectional bool If TRUE a bidirectional version of the recurrent layers is used.

rec_dropout double determining the dropout between recurrent layers. Allowed values: 0 <= x <= 0.6

rec_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

rec_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None'

rec_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

rec_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

tf_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`tf_dense_dim` `int` determining the size of the projection layer within a each transformer encoder. Allowed values: `1 <= x`

`tf_n_layers` `int` determining how many times the encoder should be added to the network. Allowed values: `0 <= x`

`tf_dropout_rate_1` `double` determining the dropout after the attention mechanism within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_dropout_rate_2` `double` determining the dropout for the dense projection within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`tf_positional_type` `string` Type of processing positional information. Allowed values: 'None', 'absolute'

`tf_num_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`tf_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`tf_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`tf_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`tf_residual_type` `string` Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

Returns: Function does nothing return. It modifies the current object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Classification: `TEClassifierParallel`, `TEClassifierParallelPrototype`, `TEClassifierProtoNet`, `TEClassifierRegular`, `TEClassifierSequentialPrototype`

`TEClassifierSequentialPrototype`

Text embedding classifier with a ProtoNet

Description

Classification Type

This object is a metric based classifer and represents in implementation of a prototypical network for few-shot learning as described by Snell, Swersky, and Zemel (2017). The network uses a multi way contrastive loss described by Zhang et al. (2019). The network learns to scale the metric as described by Oreshkin, Rodriguez, and Lacoste (2018).

Sequential Core Architecture

This model is based on a sequential architecture. The input is passed to a specific number of layers step by step. All layers are grouped by their kind into stacks.

Transformer Encoder Layers

Description

The transformer encoder layers follow the structure of the encoder layers used in transformer models. A single layer is designed as described by Chollet, Kalinowski, and Allaire (2022, p. 373) with the exception that single components of the layers (such as the activation function, the kind of residual connection, the kind of normalization or the kind of attention) can be customized. All parameters with the prefix *tf_* can be used to configure this layer.

Feature Layer

Description

The feature layer is a dense layer that can be used to increase or decrease the number of features of the input data before passing the data into your model. The aim of this layer is to increase or reduce the complexity of the data for your model. The output size of this layer determines the number of features for all following layers. In the special case that the requested number of features equals the number of features of the text embeddings this layer is reduced to a dropout layer with masking capabilities. All parameters with the prefix *feat_* can be used to configure this layer.

Dense Layers

Description

A fully connected layer. The layer is applied to every step of a sequence. All parameters with the prefix *dense_* can be used to configure this layer.

Multiple N-Gram Layers

Description

This type of layer focuses on sub-sequence and performs an 1d convolutional operation. On a word and token level these sub-sequences can be interpreted as n-grams (Jacovi, Shalom & Goldberg 2018). The convolution is done across all features. The number of filters equals the number of features of the input tensor. Thus, the shape of the tensor is retained (Pham, Kruszewski & Boleda 2016).

The layer is able to consider multiple n-grams at the same time. In this case the convolution of the n-grams is done seprately and the resulting tensors are concatenated along the feature dimension. The number of filters for every n-gram is set to num_features/num_n-grams. Thus, the resulting tensor has the same shape as the input tensor.

Sub-sequences that are masked in the input are also masked in the output.

The output of this layer can be understand as the results of the n-gram filters. Stacking this layer allows the model to perform n-gram detection of n-grams (meta perspective). All parameters with the prefix *ng_conv_* can be used to configure this layer.

Recurrent Layers

Description

A regular recurrent layer either as Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM) layer. Uses PyTorchs implementation. All parameters with the prefix *rec_* can be used to configure this layer.

Classifiction Pooling Layer

Description

Layer transforms sequences into a lower dimensional space that can be passed to dense layers. It performs two types of pooling. First, it extractes features across the time dimension selecting the maximal and/or minimal features. Second, it performs pooling over the remaining features selecting a speficifc number of the heighest and/or lowest features.

In the case of selecting the minmal *and* maximal features at the same time the minmal features are concatenated to the tensor of the maximal features resulting the in the shape $(Batch, Times, 2*Features)$ at the end of the first step. In the second step the number of requested features is halved. The first half is used for the maximal features and the second for the minimal features. All parameters with the prefix *cls_pooling_* can be used to configure this layer.

Training and Prediction

For the creation and training of a classifier an object of class EmbeddedText or LargeDataSetForTextEmbeddings on the one hand and a factor on the other hand are necessary.

The object of class EmbeddedText or LargeDataSetForTextEmbeddings contains the numerical text representations (text embeddings) of the raw texts generated by an object of class TextEmbeddingModel. For supporting large data sets it is recommended to use LargeDataSetForTextEmbeddings instead of EmbeddedText.

The `factor` contains the classes/categories for every text. Missing values (unlabeled cases) are supported and can be used for pseudo labeling.

For predictions an object of class EmbeddedText or LargeDataSetForTextEmbeddings has to be used which was created with the same TextEmbeddingModel as for training..

Value

Returns a new object of this class ready for configuration or for loading a saved classifier.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `aifeducation::ClassifiersBasedOnTextEmbeddings` -> `aifeducation::TEClassifiersBasedOnProtoNet` -> `TEClassifierSequentialPrototype`

Methods

Public methods:

- `TEClassifierSequentialPrototype$configure()`
- `TEClassifierSequentialPrototype$clone()`

Method `configure()`**:** Creating a new instance of this class.

Usage:

```
TEClassifierSequentialPrototype$configure(
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  skip_connection_type = "ResidualGate",
  cls_pooling_features = 50L,
  cls_pooling_type = "MinMax",
  metric_type = "Euclidean",
  feat_act_fct = "ELU",
  feat_size = 50L,
  feat_bias = TRUE,
  feat_dropout = 0,
  feat_parametrizations = "None",
  feat_normalization_type = "LayerNorm",
  ng_conv_act_fct = "ELU",
  ng_conv_n_layers = 1L,
  ng_conv_ks_min = 2L,
  ng_conv_ks_max = 4,
  ng_conv_bias = FALSE,
  ng_conv_dropout = 0.1,
  ng_conv_parametrizations = "None",
  ng_conv_normalization_type = "LayerNorm",
  ng_conv_residual_type = "ResidualGate",
  dense_act_fct = "ELU",
  dense_n_layers = 1L,
  dense_dropout = 0.5,
  dense_bias = FALSE,
  dense_parametrizations = "None",
  dense_normalization_type = "LayerNorm",
  dense_residual_type = "ResidualGate",
  rec_act_fct = "Tanh",
  rec_n_layers = 1,
  rec_type = "GRU",
  rec_bidirectional = FALSE,
  rec_dropout = 0.2,
  rec_bias = FALSE,
  rec_parametrizations = "None",
  rec_normalization_type = "LayerNorm",
  rec_residual_type = "ResidualGate",
  tf_act_fct = "ELU",
  tf_dense_dim = 50L,
  tf_n_layers = 1L,
  tf_dropout_rate_1 = 0.1,
  tf_dropout_rate_2 = 0.5,
  tf_attention_type = "MultiHead",
```

```
  tf_positional_type = "absolute",
  tf_num_heads = 1L,
  tf_bias = FALSE,
  tf_parametrizations = "None",
  tf_normalization_type = "LayerNorm",
  tf_residual_type = "ResidualGate",
  embedding_dim = 2L
)
```

Arguments:

`name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`label` `string` Label for the new model. Here you can use free text. Allowed values: any

`text_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`feature_extractor` `TEFeatureExtractor` Object of class TEFeatureExtractor which should be used in order to reduce the number of dimensions of the text embeddings. If no feature extractor should be applied set `NULL`.

`target_levels` `vector` containing the levels (categories or classes) within the target data. Please note that order matters. For ordinal data please ensure that the levels are sorted correctly with later levels indicating a higher category/class. For nominal data the order does not matter.

`skip_connection_type` `string` Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

`cls_pooling_features` `int` Number of features to be extracted at the end of the model. Allowed values: `1 <= x`

`cls_pooling_type` `string` Type of extracting intermediate features. Allowed values: 'Max', 'Min', 'MinMax'

`metric_type` `string` Type of metric used for calculating the distance. Allowed values: 'Euclidean'

`feat_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`feat_size` `int` Number of neurons for each dense layer. Allowed values: `2 <= x`

`feat_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`feat_dropout` `double` determining the dropout for the dense projection of the feature layer. Allowed values: `0 <= x <= 0.6`

`feat_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`feat_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`ng_conv_act_fct` `string` Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

`ng_conv_n_layers` `int` determining how many times the n-gram layers should be added to the network. Allowed values: `0 <= x`

ng_conv_ks_min int determining the minimal window size for n-grams. Allowed values: 2 <= x

ng_conv_ks_max int determining the maximal window size for n-grams. Allowed values: 2 <= x

ng_conv_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

ng_conv_dropout double determining the dropout for n-gram convolution layers. Allowed values: 0 <= x <= 0.6

ng_conv_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

ng_conv_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

ng_conv_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

dense_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

dense_n_layers int Number of dense layers. Allowed values: 0 <= x

dense_dropout double determining the dropout between dense layers. Allowed values: 0 <= x <= 0.6

dense_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

dense_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

dense_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

dense_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

rec_act_fct string Activation function for all layers. Allowed values: 'Tanh'

rec_n_layers int Number of recurrent layers. Allowed values: 0 <= x

rec_type string Type of the recurrent layers. rec_type='GRU' for Gated Recurrent Unit and rec_type='LSTM' for Long Short-Term Memory. Allowed values: 'GRU', 'LSTM'

rec_bidirectional bool If TRUE a bidirectional version of the recurrent layers is used.

rec_dropout double determining the dropout between recurrent layers. Allowed values: 0 <= x <= 0.6

rec_bias bool If TRUE a bias term is added to all layers. If FALSE no bias term is added to the layers.

rec_parametrizations string Re-Parametrizations of the weights of layers. Allowed values: 'None'

rec_normalization_type string Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

rec_residual_type string Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

tf_act_fct string Activation function for all layers. Allowed values: 'ELU', 'LeakyReLU', 'ReLU', 'GELU', 'Sigmoid', 'Tanh', 'PReLU'

tf_dense_dim int determining the size of the projection layer within a each transformer encoder. Allowed values: 1 <= x

`tf_n_layers` `int` determining how many times the encoder should be added to the network. Allowed values: `0 <= x`

`tf_dropout_rate_1` `double` determining the dropout after the attention mechanism within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_dropout_rate_2` `double` determining the dropout for the dense projection within the transformer encoder layers. Allowed values: `0 <= x <= 0.6`

`tf_attention_type` `string` Choose the attention type. Allowed values: 'Fourier', 'MultiHead'

`tf_positional_type` `string` Type of processing positional information. Allowed values: 'None', 'absolute'

`tf_num_heads` `int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type='multihead'` Allowed values: `0 <= x`

`tf_bias` `bool` If `TRUE` a bias term is added to all layers. If `FALSE` no bias term is added to the layers.

`tf_parametrizations` `string` Re-Parametrizations of the weights of layers. Allowed values: 'None', 'OrthogonalWeights', 'WeightNorm', 'SpectralNorm'

`tf_normalization_type` `string` Type of normalization applied to all layers and stack layers. Allowed values: 'LayerNorm', 'None'

`tf_residual_type` `string` Type of residual connenction for all layers and stack of layers. Allowed values: 'ResidualGate', 'Addition', 'None'

`embedding_dim` `int` determining the number of dimensions for the embedding. Allowed values: `2 <= x`

Returns: Function does nothing return. It modifies the current object.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Oreshkin, B. N., Rodriguez, P. & Lacoste, A. (2018). TADAM: Task dependent adaptive metric for improved few-shot learning. https://doi.org/10.48550/arXiv.1805.10123

Snell, J., Swersky, K. & Zemel, R. S. (2017). Prototypical Networks for Few-shot Learning. https://doi.org/10.48550/arXiv.1703.05175

Zhang, X., Nie, J., Zong, L., Yu, H. & Liang, W. (2019). One Shot Learning with Margin. In Q. Yang, Z.-H. Zhou, Z. Gong, M.-L. Zhang & S.-J. Huang (Eds.), Lecture Notes in Computer Science. Advances in Knowledge Discovery and Data Mining (Vol. 11440, pp. 305–317). Springer International Publishing. https://doi.org/10.1007/978-3-030-16145-3_24

See Also

Other Classification: `TEClassifierParallel`, `TEClassifierParallelPrototype`, `TEClassifierProtoNet`, `TEClassifierRegular`, `TEClassifierSequential`

`TEFeatureExtractor` *Feature extractor for reducing the number for dimensions of text embeddings.*

Description

Abstract class for auto encoders with 'pytorch'.

Objects of this class are used for reducing the number of dimensions of text embeddings created by an object of class TextEmbeddingModel.

For training an object of class EmbeddedText or LargeDataSetForTextEmbeddings generated by an object of class TextEmbeddingModel is necessary. Passing raw texts is not supported.

For prediction an ob object class EmbeddedText or LargeDataSetForTextEmbeddings is necessary that was generated with the same TextEmbeddingModel as during training. Prediction outputs a new object of class EmbeddedText or LargeDataSetForTextEmbeddings which contains a text embedding with a lower number of dimensions.

All models use tied weights for the encoder and decoder layers (except `method="LSTM"`) and apply the estimation of orthogonal weights. In addition, training tries to train the model to achieve uncorrelated features.

Objects of class TEFeatureExtractor are designed to be used with classifiers such as TEClassifierRegular and TEClassifierProtoNet.

Value

A new instances of this class.

Super classes

`aifeducation::AIFEMaster` -> `aifeducation::AIFEBaseModel` -> `aifeducation::ModelsBasedOnTextEmbeddings` -> `TEFeatureExtractor`

Methods

Public methods:

- `TEFeatureExtractor$configure()`
- `TEFeatureExtractor$train()`
- `TEFeatureExtractor$extract_features()`
- `TEFeatureExtractor$extract_features_large()`
- `TEFeatureExtractor$plot_training_history()`
- `TEFeatureExtractor$clone()`

Method `configure()`**:** Creating a new instance of this class.

Usage:

```
TEFeatureExtractor$configure(
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  features = 128L,
  method = "dense",
  orthogonal_method = "matrix_exp",
  noise_factor = 0.2
)
```

Arguments:

`name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`label` `string` Label for the new model. Here you can use free text. Allowed values: any

`text_embeddings` `EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`features` `int` Number of features the model should use. Allowed values: `1 <= x`

`method` `string` Method to use for the feature extraction. `'lstm'` for an extractor based on LSTM-layers or `'Dense'` for dense layers. Allowed values: 'Dense', 'LSTM'

`orthogonal_method` `string` Method to use for the feature extraction. `'lstm'` for an extractor based on LSTM-layers or `'Dense'` for dense layers. Allowed values: 'Dense', 'LSTM'

`noise_factor` `double` Value between 0 and a value lower 1 indicating how much noise should be added to the input during training. Allowed values: `0 <= x <= 1`

Returns: Returns an object of class TEFeatureExtractor which is ready for training.

Method `train()`**:** Method for training a neural net.

Usage:

```
TEFeatureExtractor$train(
  data_embeddings = NULL,
  data_val_size = 0.25,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15L,
  sustain_log_level = "warning",
  epochs = 40L,
  batch_size = 32L,
  trace = TRUE,
  ml_trace = 1L,
  log_dir = NULL,
  log_write_interval = 10L,
  lr_rate = 0.001,
  lr_warm_up_ratio = 0.02,
  optimizer = "AdamW"
)
```

Arguments:

`data_embeddings EmbeddedText, LargeDataSetForTextEmbeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`data_val_size double` between 0 and 1, indicating the proportion of cases which should be used for the validation sample during the estimation of the model. The remaining cases are part of the training data. Allowed values: `0 < x < 1`

`sustain_track bool` If `TRUE` energy consumption is tracked during training via the python library 'codecarbon'.

`sustain_iso_code string` ISO code (Alpha-3-Code) for the country. This variable must be set if sustainability should be tracked. A list can be found on Wikipedia: https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes. Allowed values: any

`sustain_region string` Region within a country. Only available for USA and Canada See the documentation of codecarbon for more information. https://mlco2.github.io/codecarbon/parameters.html Allowed values: any

`sustain_interval int` Interval in seconds for measuring power usage. Allowed values: `1 <= x`

`sustain_log_level string` Level for printing information to the console. Allowed values: 'debug', 'info', 'warning', 'error', 'critical'

`epochs int` Number of training epochs. Allowed values: `1 <= x`

`batch_size int` Size of the batches for training. Allowed values: `1 <= x`

`trace bool` `TRUE` if information about the estimation phase should be printed to the console.

`ml_trace int` `ml_trace=0` does not print any information about the training process from pytorch on the console. Allowed values: `0 <= x <= 1`

`log_dir string` Path to the directory where the log files should be saved. If no logging is desired set this argument to `NULL`. Allowed values: any

`log_write_interval int` Time in seconds determining the interval in which the logger should try to update the log files. Only relevant if `log_dir` is not `NULL`. Allowed values: `1 <= x`

`lr_rate double` Initial learning rate for the training. Allowed values: `0 < x <= 1`

`lr_warm_up_ratio double` Number of epochs used for warm up. Allowed values: `0 < x < 0.5`

`optimizer string` determining the optimizer used for training. Allowed values: 'Adam', 'RMSprop', 'AdamW', 'SGD'

Returns: Function does not return a value. It changes the object into a trained classifier.

Method `extract_features()`**:** Method for extracting features. Applying this method reduces the number of dimensions of the text embeddings. Please note that this method should only be used if a small number of cases should be compressed since the data is loaded completely into memory. For a high number of cases please use the method `extract_features_large`.

Usage:

```
TEFeatureExtractor$extract_features(data_embeddings, batch_size)
```

Arguments:

`data_embeddings` Object of class EmbeddedText,LargeDataSetForTextEmbeddings, `datasets.arrow_dataset.Datas` or `array` containing the text embeddings which should be reduced in their dimensions.

`batch_size int` batch size.

Returns: Returns an object of class EmbeddedText containing the compressed embeddings.

Method `extract_features_large()`**:** Method for extracting features from a large number of cases. Applying this method reduces the number of dimensions of the text embeddings.

Usage:

```
TEFeatureExtractor$extract_features_large(
  data_embeddings,
  batch_size,
  trace = FALSE
)
```

Arguments:

`data_embeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings containing the text embeddings which should be reduced in their dimensions.

`batch_size` `int` batch size.

`trace` `bool` If `TRUE` information about the progress is printed to the console.

Returns: Returns an object of class LargeDataSetForTextEmbeddings containing the compressed embeddings.

Method `plot_training_history()`**:** Method for requesting a plot of the training history. This method requires the *R* package 'ggplot2' to work.

Usage:

```
TEFeatureExtractor$plot_training_history(
  y_min = NULL,
  y_max = NULL,
  text_size = 10L
)
```

Arguments:

`y_min` Minimal value for the y-axis. Set to `NULL` for an automatic adjustment.

`y_max` Maximal value for the y-axis. Set to `NULL` for an automatic adjustment.

`text_size` Size of the text.

Returns: Returns a plot of class `ggplot` visualizing the training process.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

`features` refers to the number of features for the compressed text embeddings.

This model requires `pad_value=0`. If this condition is not met the padding value is switched automatically.

This model requires that the underlying TextEmbeddingModel uses `pad_value=0`. If this condition is not met the pad value is switched before training.

See Also

Other Text Embedding: `TextEmbeddingModel`

tensor_list_to_numpy *Convert list of tensors into numpy arrays*

Description

Function converts tensors within a `list` into numpy arrays in order to allow further operations in *R*.

Usage

```
tensor_list_to_numpy(tensor_list)
```

Arguments

`tensor_list`	`list` of objects.

Value

Returns the same `list` with the exception that objects of class `torch.Tensor` are transformed into numpy arrays. If the tensor requires a gradient and/or is on gpu it is detached and converted. If the object in a list is not of this class the original object is returned.

See Also

Other Utils Python Data Management Developers: `class_vector_to_py_dataset()`, `data.frame_to_py_dataset()`, `get_batches_index()`, `prepare_r_array_for_dataset()`, `py_dataset_to_embeddings()`, `reduce_to_unique()`, `tensor_to_numpy()`

tensor_to_matrix_c *Transform tensor to matrix*

Description

Function written in C++ for transformation the tensor (with size batch x times x features) to the matrix (with size batch x times*features)

Usage

```
tensor_to_matrix_c(tensor, times, features)
```

Arguments

`tensor`	`3-D array (cube)` data as tensor (with size batch x times x features)
`times`	`unsigned integer` times number
`features`	`unsigned integer` features number

Value

Returns matrix (with size batch x times*features)

See Also

Other Utils Developers: auto_n_cores(), create_object(), create_synthetic_units_from_matrix(), generate_id(), get_n_chunks(), get_synthetic_cases_from_matrix(), get_time_stamp(), matrix_to_array_c(), to_categorical_c()

---

tensor_to_numpy *Tensor_to_numpy*

Description

Function converts a tensor into a numpy array in order to allow further operations in *R*.

Usage

```
tensor_to_numpy(object)
```

Arguments

object	Object of any class.

Value

In the case the object is of class `torch.Tensor` it returns a numpy error. If the tensor requires a gradient and/or is on gpu it is detached and converted. If the object is not of class `torch.Tensor` the original object is returned.

See Also

Other Utils Python Data Management Developers: class_vector_to_py_dataset(), data.frame_to_py_dataset(), get_batches_index(), prepare_r_array_for_dataset(), py_dataset_to_embeddings(), reduce_to_unique(), tensor_list_to_numpy()

TextEmbeddingModel *Text embedding model*

Description

This R6 class stores a text embedding model which can be used to tokenize, encode, decode, and embed raw texts. The object provides a unique interface for different text processing methods.

Value

Objects of class TextEmbeddingModel transform raw texts into numerical representations which can be used for downstream tasks. For this aim objects of this class allow to tokenize raw texts, to encode tokens to sequences of integers, and to decode sequences of integers back to tokens.

Super classes

aifeducation::AIFEMaster -> aifeducation::AIFEBaseModel -> TextEmbeddingModel

Public fields

BaseModel ('BaseModelCore')
Object of class BaseModelCore.

Methods

Public methods:

- TextEmbeddingModel$configure()
- TextEmbeddingModel$load_from_disk()
- TextEmbeddingModel$save()
- TextEmbeddingModel$encode()
- TextEmbeddingModel$decode()
- TextEmbeddingModel$embed()
- TextEmbeddingModel$embed_large()
- TextEmbeddingModel$get_n_features()
- TextEmbeddingModel$get_pad_value()
- TextEmbeddingModel$set_publication_info()
- TextEmbeddingModel$get_sustainability_data()
- TextEmbeddingModel$estimate_sustainability_inference_embed()
- TextEmbeddingModel$clone()

Method configure(): Method for creating a new text embedding model

Usage:

```
TextEmbeddingModel$configure(
  model_name = NULL,
  model_label = NULL,
  model_language = NULL,
  max_length = 0L,
  chunks = 2L,
  overlap = 0L,
  emb_layer_min = 1L,
  emb_layer_max = 2L,
  emb_pool_type = "Average",
  pad_value = -100L,
  base_model = NULL
)
```

Arguments:

`model_name` `string` Name of the new model. Please refer to common name conventions. Free text can be used with parameter `label`. If set to `NULL` a unique ID is generated automatically. Allowed values: any

`model_label` `string` Label for the new model. Here you can use free text. Allowed values: any

`model_language` `string` Languages that the models can work with. Allowed values: any

`max_length` `int` Maximal number of token per chunks. Must be equal or lower as the maximal postional embeddings for the model. Allowed values: `20 <= x`

`chunks` `int` Maximal number chunks. Allowed values: `2 <= x`

`overlap` `int` Number of tokens from the previous chunk that should be added at the beginng of the next chunk. Allowed values: `0 <= x`

`emb_layer_min` `int` Minimal layer from which the embeddings should be calculated. Allowed values: `1 <= x`

`emb_layer_max` `int` Maximal layer from which the embeddings should be calculated. Allowed values: `1 <= x`

`emb_pool_type` `string` Method to summarize the embedding of single tokens into a text embedding. In the case of `'CLS'` all cls-tokens between `emb_layer_min` and `emb_layer_max` are averaged. In the case of `'Average'` the embeddings of all tokens are averaged. Please note that BaseModelFunnel allows only 'CLS'. Allowed values: 'CLS', 'Average'

`pad_value` `int` Value indicating padding. This value should no be in the range of regluar values for computations. Thus it is not recommended to chance this value. Default is `-100`. Allowed values: `x <= -100`

`base_model` `BaseModelCore` BaseModels for processing raw texts.

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

Returns: Does nothing return.

Method `load_from_disk()`**:** Loads an object from disk and updates the object to the current version of the package.

Usage:

```
TextEmbeddingModel$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the object set is stored.

Returns: Function does nothin return. It loads an object from disk.

Method `save()`**:** Method for saving a model on disk.

Usage:

```
TextEmbeddingModel$save(dir_path, folder_name)
```

Arguments:

`dir_path` Path to the directory where to save the object.

`folder_name string` Name of the folder where the model should be saved. Allowed values: any

Returns: Function does nothing return. It is used to save an object on disk.

Method `encode()`**:** Method for encoding words of raw texts into integers.

Usage:

```
TextEmbeddingModel$encode(
  raw_text,
  token_encodings_only = FALSE,
  token_to_int = TRUE,
  trace = FALSE
)
```

Arguments:

`raw_text vector` Raw text.

`token_encodings_only bool`

- `TRUE`: Returns a `list` containg only the tokens.
- `FALSE`: Returns a `list` containg a list for the tokens, the number of chunks, and the number potential number of chunks for each document/text.

`token_to_int bool`

- `TRUE`: Returns the tokens as `int` index.
- `FALSE`: Returns the tokens as `strings`.

`trace bool` `TRUE` if information about the estimation phase should be printed to the console.

Returns: `list` containing the integer or token sequences of the raw texts with special tokens.

Method `decode()`**:** Method for decoding a sequence of integers into tokens

Usage:

```
TextEmbeddingModel$decode(int_seqence, to_token = FALSE)
```

Arguments:

`int_seqence list` `list` of integer sequence that should be converted to tokens.

`to_token bool`

- `FALSE`: Transforms the integers to plain text.
- `TRUE`: Transforms the integers to a sequence of tokens.

Returns: `list` of token sequences

Method `embed()`**:** Method for creating text embeddings from raw texts. This method should only be used if a small number of texts should be transformed into text embeddings. For a large number of texts please use the method `embed_large`.

Usage:

```
TextEmbeddingModel$embed(
  raw_text = NULL,
  doc_id = NULL,
  batch_size = 8L,
  trace = FALSE,
  return_large_dataset = FALSE
)
```

Arguments:

`raw_text` `vector` Raw text.

`doc_id` `vector` Id for every text.

`batch_size` `int` Size of the batches for training. Allowed values: `1 <= x`

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

`return_large_dataset` `bool` If `TRUE` a LargeDataSetForTextEmbeddings is returned. If `FALSE` an object if class EmbeddedText is returned.

Returns: Method returns an object of class EmbeddedText or LargeDataSetForTextEmbeddings. This object contains the embeddings as a data.frame and information about the model creating the embeddings.

Method `embed_large()`**:** Method for creating text embeddings from raw texts.

Usage:

```
TextEmbeddingModel$embed_large(
  text_dataset,
  batch_size = 32L,
  trace = FALSE,
  log_file = NULL,
  log_write_interval = 2L
)
```

Arguments:

`text_dataset` `LargeDataSetForText` LargeDataSetForText Object storing textual data.

`batch_size` `int` Size of the batches for training. Allowed values: `1 <= x`

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

`log_file` `string` Path to the file where the log files should be saved. If no logging is desired set this argument to `NULL`. Allowed values: any

`log_write_interval` `int` Time in seconds determining the interval in which the logger should try to update the log files. Only relevant if `log_dir` is not `NULL`. Allowed values: `1 <= x`

Returns: Method returns an object of class LargeDataSetForTextEmbeddings.

Method `get_n_features()`**:** Method for requesting the number of features.

Usage:

```
TextEmbeddingModel$get_n_features()
```

Returns: Returns a `double` which represents the number of features. This number represents the hidden size of the embeddings for every chunk or time.

Method `get_pad_value()`**:** Value for indicating padding.

Usage:

```
TextEmbeddingModel$get_pad_value()
```

Returns: Returns an `int` describing the value used for padding.

Method `set_publication_info()`**:** Method for setting the bibliographic information of the model.

Usage:

```
TextEmbeddingModel$set_publication_info(type, authors, citation, url = NULL)
```

Arguments:

`type` `string` Type of information which should be changed/added. `developer`, and `modifier` are possible.

`authors` List of people.

`citation` `string` Citation in free text.

`url` `string` Corresponding URL if applicable.

Returns: Function does not return a value. It is used to set the private members for publication information of the model.

Method `get_sustainability_data()`**:** Method for requesting a summary of tracked energy consumption during training and an estimate of the resulting CO2 equivalents in kg.

Usage:

```
TextEmbeddingModel$get_sustainability_data(track_mode = "training")
```

Arguments:

`track_mode` `string` Determines the stept to which the data refer. Allowed values: 'training', 'inference'

Returns: Returns a `list` containing the tracked energy consumption, CO2 equivalents in kg, information on the tracker used, and technical information on the training infrastructure.

Method `estimate_sustainability_inference_embed()`**:** Calculates the energy consumption for inference of the given task.

Usage:

```
TextEmbeddingModel$estimate_sustainability_inference_embed(
  text_dataset = NULL,
  batch_size = 32L,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 10L,
  sustain_log_level = "warning",
  trace = TRUE
)
```

Arguments:

`text_dataset` `LargeDataSetForText` LargeDataSetForText Object storing textual data.

`batch_size` `int` Size of the batches for training. Allowed values: `1 <= x`

`sustain_iso_code` `string` ISO code (Alpha-3-Code) for the country. This variable must be set if sustainability should be tracked. A list can be found on Wikipedia: https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes. Allowed values: any

`sustain_region` `string` Region within a country. Only available for USA and Canada See the documentation of codecarbon for more information. https://mlco2.github.io/codecarbon/parameters.html Allowed values: any

`sustain_interval` `int` Interval in seconds for measuring power usage. Allowed values: `1 <= x`

`sustain_log_level`

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

Returns: Returns nothing. Method saves the statistics internally. The statistics can be accessed with the method `get_sustainability_data("inference")`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Text Embedding: `TEFeatureExtractor`

`TokenizerBase` *Base class for tokenizers*

Description

Base class for tokenizers containing all methods shared by the sub-classes.

Value

Does return a new object of this class.

Returns a `data.frame` containing the estimates.

Super class

`aifeducation::AIFEMaster` -> `TokenizerBase`

Methods

Public methods:

- TokenizerBase$save()
- TokenizerBase$load_from_disk()
- TokenizerBase$get_tokenizer_statistics()
- TokenizerBase$get_tokenizer()
- TokenizerBase$encode()
- TokenizerBase$decode()
- TokenizerBase$get_special_tokens()
- TokenizerBase$n_special_tokens()
- TokenizerBase$calculate_statistics()
- TokenizerBase$clone()

Method `save()`**:** Method for saving a model on disk.

Usage:

```
TokenizerBase$save(dir_path, folder_name)
```

Arguments:

`dir_path` Path to the directory where to save the object.

`folder_name` `string` Name of the folder where the model should be saved. Allowed values: any

Returns: Function does nothing return. It is used to save an object on disk.

Method `load_from_disk()`**:** Loads an object from disk and updates the object to the current version of the package.

Usage:

```
TokenizerBase$load_from_disk(dir_path)
```

Arguments:

`dir_path` Path where the object set is stored.

Returns: Function does nothin return. It loads an object from disk.

Method `get_tokenizer_statistics()`**:** Tokenizer statistics

Usage:

```
TokenizerBase$get_tokenizer_statistics()
```

Returns: Returns a `data.frame` containing the tokenizer's statistics.

Method `get_tokenizer()`**:** Python tokenizer

Usage:

```
TokenizerBase$get_tokenizer()
```

Returns: Returns the python tokenizer within the model.

Method `encode()`**:** Method for encoding words of raw texts into integers.

Usage:

```
TokenizerBase$encode(
  raw_text,
  token_overlap = 0L,
  max_token_sequence_length = 512L,
  n_chunks = 1L,
  token_encodings_only = FALSE,
  token_to_int = TRUE,
  return_token_type_ids = TRUE,
  trace = FALSE
)
```

Arguments:

`raw_text` `vector` Raw text.

`token_overlap` `int` Number of tokens from the previous chunk that should be added at the beginng of the next chunk. Allowed values: `0 <= x`

`max_token_sequence_length` `int` Maximal number of tokens per chunk. Allowed values: `20 <= x`

`n_chunks` `int` Maximal number chunks. Allowed values: `1 <= x`

`token_encodings_only` `bool`

- `TRUE`: Returns a `list` containg only the tokens.
- `FALSE`: Returns a `list` containg a list for the tokens, the number of chunks, and the number potential number of chunks for each document/text.

`token_to_int` `bool`

- `TRUE`: Returns the tokens as `int` index.
- `FALSE`: Returns the tokens as `strings`.

`return_token_type_ids` `bool` If `TRUE` additionally returns the return_token_type_ids.

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

Returns: `list` containing the integer or token sequences of the raw texts with special tokens.

Method `decode()`**:** Method for decoding a sequence of integers into tokens

Usage:

```
TokenizerBase$decode(int_seqence, to_token = FALSE)
```

Arguments:

`int_seqence` `list` `list` of integer sequence that should be converted to tokens.

`to_token` `bool`

- `FALSE`: Transforms the integers to plain text.
- `TRUE`: Transforms the integers to a sequence of tokens.

Returns: `list` of token sequences

Method `get_special_tokens()`**:** Method for receiving the special tokens of the model

Usage:

```
TokenizerBase$get_special_tokens()
```

Returns: Returns a `matrix` containing the special tokens in the rows and their type, token, and id in the columns.

Method `n_special_tokens()`**:** Method for receiving the special tokens of the model

Usage:

```
TokenizerBase$n_special_tokens()
```

Returns: Returns an 'int' counting the number of special tokens.

Method `calculate_statistics()`**:** Method for calculating tokenizer statistics as suggested by Kaya and Tantuğ (2024).

Kaya, Y. B., & Tantuğ, A. C. (2024). Effect of tokenization granularity for Turkish large language models. Intelligent Systems with Applications, 21, 200335. <https://doi.org/10.1016/j.iswa.2024.200335>

Usage:

```
TokenizerBase$calculate_statistics(
  text_dataset,
  statistics_max_tokens_length,
  step = "creation"
)
```

Arguments:

`text_dataset` `LargeDataSetForText` LargeDataSetForText Object storing textual data.

`statistics_max_tokens_length` `int` Maximum sequence length for calculating the statistics. Allowed values: `20 <= x <= 8192`

`step` `string` describing the context of the estimation.

Returns: Returns an 'int' counting the number of special tokens.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other R6 Classes for Developers: `AIFEBaseModel`, `AIFEMaster`, `BaseModelCore`, `ClassifiersBasedOnTextEmbeddings`, `DataManagerClassifier`, `LargeDataSetBase`, `ModelsBasedOnTextEmbeddings`, `TEClassifiersBasedOnProtoNet`, `TEClassifiersBasedOnRegular`

`TokenizerIndex` *List of all available Tokenizers*

Description

Named `list` containing all tokenizers as a string.

Usage

```
TokenizerIndex
```

Format

An object of class `list` of length 2.

See Also

Other Parameter Dictionary: `BaseModelsIndex`, `DataSetsIndex`, `get_TEClassifiers_class_names()`, `get_called_args()`, `get_depr_obj_names()`, `get_magnitude_values()`, `get_param_def()`, `get_param_dict()`, `get_param_doc_desc()`

`to_categorical_c` *Transforming classes to one-hot encoding*

Description

Function transforming a vector of classes (int) into a binary class matrix.

Usage

```
to_categorical_c(class_vector, n_classes)
```

Arguments

`class_vector`	`vector` containing integers for every class. The integers must range from 0 to n_classes-1.
`n_classes`	`int` Total number of classes.

Value

Returns a `matrix` containing the binary representation for every class.

See Also

Other Utils Developers: `auto_n_cores()`, `create_object()`, `create_synthetic_units_from_matrix()`, `generate_id()`, `get_n_chunks()`, `get_synthetic_cases_from_matrix()`, `get_time_stamp()`, `matrix_to_array_c()`, `tensor_to_matrix_c()`

update_aifeducation *Updates an existing installation of 'aifeducation' on a machine*

Description

Function for updating 'aifeducation' on a machine.

The function tries to find an existing environment on the machine, removes the environment and installs the environment with the new python modules.

In the case `env_type = "auto"` the function tries to update an existing virtual environment. If no virtual environment exits it tries to update a conda environment.

Usage

```
update_aifeducation(
  update_aifeducation_studio = TRUE,
  env_type = "auto",
  cuda_version = "12.4",
  envname = "aifeducation"
)
```

Arguments

update_aifeducation_studio	`bool` If `TRUE` all necessary R packages are installed for using AI for Education Studio.
env_type	`string` If set to `"venv"` virtual environment is requested. If set to `"conda"` a 'conda' environment is requested. If set to `"auto"` the function tries to use a virtual environment with the given name. If this environment does not exist it tries to activate a conda environment with the given name. If this fails the default virtual environment is used.'
cuda_version	`string` determining the requested version of cuda.
envname	`string` Name of the environment where the packages should be installed.

Value

Function does nothing return. It installs python, optional R packages, and necessary 'python' packages on a machine.

Note

On MAC OS torch will be installed without support for cuda.

See Also

Other Installation and Configuration: `check_aif_py_modules()`, `get_recommended_py_versions()`, `install_aifeducation()`, `install_aifeducation_studio()`, `install_py_modules()`, `prepare_session()`, `set_transformers_logger()`

WordPieceTokenizer *WordPieceTokenizer*

Description

Tokenizer based on the WordPiece model (Wu et al. 2016).

Value

Does return a new object of this class.

Super classes

aifeducation::AIFEMaster -> aifeducation::TokenizerBase -> WordPieceTokenizer

Methods

Public methods:

- WordPieceTokenizer$configure()
- WordPieceTokenizer$train()
- WordPieceTokenizer$clone()

Method `configure()`**:** Configures a new object of this class.

Usage:

```
WordPieceTokenizer$configure(vocab_size = 10000L, vocab_do_lower_case = FALSE)
```

Arguments:

`vocab_size` `int` Size of the vocabulary. Allowed values: `1000 <= x <= 500000`

`vocab_do_lower_case` `bool` `TRUE` if all tokens should be lower case.

Returns: Does nothing return.

Method `train()`**:** Trains a new object of this class

Usage:

```
WordPieceTokenizer$train(
  text_dataset,
  statistics_max_tokens_length = 512L,
  sustain_track = FALSE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15L,
  sustain_log_level = "warning",
  trace = FALSE
)
```

Arguments:

`text_dataset` `LargeDataSetForText` LargeDataSetForText Object storing textual data.

`statistics_max_tokens_length` `int` Maximum sequence length for calculating the statistics. Allowed values: `20 <= x <= 8192`

`sustain_track` `bool` If `TRUE` energy consumption is tracked during training via the python library 'codecarbon'.

`sustain_iso_code` `string` ISO code (Alpha-3-Code) for the country. This variable must be set if sustainability should be tracked. A list can be found on Wikipedia: https://en.wikipedia.org/wiki/List_of_ISO_3166_country_codes. Allowed values: any

`sustain_region` `string` Region within a country. Only available for USA and Canada See the documentation of codecarbon for more information. https://mlco2.github.io/codecarbon/parameters.html Allowed values: any

`sustain_interval` `int` Interval in seconds for measuring power usage. Allowed values: `1 <= x`

`sustain_log_level` `string` Level for printing information to the console. Allowed values: 'debug', 'info', 'warning', 'error', 'critical'

`trace` `bool` `TRUE` if information about the estimation phase should be printed to the console.

Returns: Does nothing return.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., . . . Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. <https://doi.org/10.48550/arXiv.1609.08144>

See Also

Other Tokenizer: HuggingFaceTokenizer

`write_log` *Write log*

Description

Function for writing a log file from R containing three rows and three columns. The log file can report the current status of maximal three processes. The first row describes the top process. The second row describes the status of the process within the top process. The third row can be used to describe the status of a process within the middle process.

The log can be read with read_log.

Usage

```
write_log(
  log_file,
  value_top = 0L,
  total_top = 1L,
  message_top = NA,
  value_middle = 0L,
  total_middle = 1L,
  message_middle = NA,
  value_bottom = 0L,
  total_bottom = 1L,
  message_bottom = NA,
  last_log = NULL,
  write_interval = 2L
)
```

Arguments

log_file	string Path to the file where the log should be saved and updated.
value_top	double Current value for the top process.
total_top	double Maximal value for the top process.
message_top	string Message describing the current state of the top process.
value_middle	double Current value for the middle process.
total_middle	double Maximal value for the middle process.
message_middle	string Message describing the current state of the middle process.
value_bottom	double Current value for the bottom process.
total_bottom	double Maximal value for the bottom process.
message_bottom	string Message describing the current state of the bottom process.
last_log	POSIXct Time when the last log was created. If there is no log file set this value to NULL.
write_interval	int Time in seconds. This time must be past before a new log is created.

Value

This function writes a log file to the given location. If log_file is NULL the function will not try to write a log file.

If log_file is a valid path to a file the function will write a log if the time specified by write_interval has passed. In addition the function will return an object of class POSIXct describing the time when the log file was successfully updated. If the initial attempt for writing log fails the function returns the value of last_log which is NULL by default.

See Also

Other Utils Log Developers: cat_message(), clean_pytorch_log_transformers(), output_message(), print_message(), read_log(), read_loss_log(), reset_log(), reset_loss_log()

Index