Package 'aifeducation'

July 22, 2025

Type Package

Title Artificial Intelligence for Education

Version 1.0.2

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. Bunkhumpornpat et al. (2012) <doi:10.1007/s10489-011-0287-y>). 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), irrCAC, methods, Rcpp (>= 1.0.10), reshape2, reticulate (>= 1.34.0), rlang, smotefamily, stringi, utils

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

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

SystemRequirements PyTorch (see vignette ``Get started")

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 2025-02-05 13:00:02 UTC

Contents

`.AIFEBaseTransformer` *Base* `R6` *class for creation and definition of* `.AIFE*Transformer-like` *classes*

Description

This base class is used to create and define `.AIFE*Transformer-like` classes. It serves as a skeleton for a future concrete transformer and cannot be used to create an object of itself (an attempt to call `new`-method will produce an error).

See p.1 Base Transformer Class in Transformers for Developers for details.

Create

The `create`-method is a basic algorithm that is used to create a new transformer, but cannot be called directly.

Train

The `train`-method is a basic algorithm that is used to train and tune the transformer but cannot be called directly.

Concrete transformer implementation

There are already implemented concrete (child) transformers (e.g. `BERT`, `DeBERTa-V2`, etc.), to implement a new one see p.4 Implement A Custom Transformer in Transformers for Developers

Public fields

`params` A list containing transformer's parameters ('static', 'dynamic' and 'dependent' parameters) `list()` containing all the transformer parameters. Can be set with `set_model_param()`.

'Static' parameters:
Regardless of the transformer, the following parameters are always included:

- `ml_framework`
- `text_dataset`
- `sustain_track`
- `sustain_iso_code`
- `sustain_region`
- `sustain_interval`
- `trace`
- `pytorch_safetensors`
- `log_dir`
- `log_write_interval`

'Dynamic' parameters:
In the case of **create** it also contains (see `create`-method for details):

- `model_dir`
- `vocab_size`
- `max_position_embeddings`
- `hidden_size`
- `hidden_act`
- `hidden_dropout_prob`

- attention_probs_dropout_prob
- intermediate_size
- num_attention_heads

In the case of **train** it also contains (see train-method for details):

- output_dir
- model_dir_path
- p_mask
- whole_word
- val_size
- n_epoch
- batch_size
- chunk_size
- min_seq_len
- full_sequences_only
- learning_rate
- n_workers
- multi_process
- keras_trace
- pytorch_trace

'Dependent' parameters:

Depending on the transformer and the method used class may contain different parameters:

- vocab_do_lower_case
- num_hidden_layer
- add_prefix_space
- etc.

`temp` A list containing temporary transformer's parameters

list() containing all the temporary local variables that need to be accessed between the step functions. Can be set with set_model_temp().

For example, it can be a variable tok_new that stores the tokenizer from steps_for_creation$create_tokenizer_dra
To train the tokenizer, access the variable tok_new in steps_for_creation$calculate_vocab through the temp list of this class.

Methods

Public methods:

- .AIFEBaseTransformer$new()
- .AIFEBaseTransformer$set_title()
- .AIFEBaseTransformer$set_model_param()
- .AIFEBaseTransformer$set_model_temp()
- .AIFEBaseTransformer$set_SFC_check_max_pos_emb()
- .AIFEBaseTransformer$set_SFC_create_tokenizer_draft()
- .AIFEBaseTransformer$set_SFC_calculate_vocab()
- .AIFEBaseTransformer$set_SFC_save_tokenizer_draft()

- .AIFEBaseTransformer$set_SFC_create_final_tokenizer()
- .AIFEBaseTransformer$set_SFC_create_transformer_model()
- .AIFEBaseTransformer$set_required_SFC()
- .AIFEBaseTransformer$set_SFT_load_existing_model()
- .AIFEBaseTransformer$set_SFT_cuda_empty_cache()
- .AIFEBaseTransformer$set_SFT_create_data_collator()
- .AIFEBaseTransformer$create()
- .AIFEBaseTransformer$train()
- .AIFEBaseTransformer$clone()

Method `new()`**:** An object of this class cannot be created. Thus, method's call will produce an error.

Usage:

```
.AIFEBaseTransformer$new()
```

Returns: This method returns an error.

Method `set_title()`**:** Setter for the title. Sets a new value for the `title` private attribute.

Usage:

```
.AIFEBaseTransformer$set_title(title)
```

Arguments:

`title string` A new title.

Returns: This method returns nothing.

Method `set_model_param()`**:** Setter for the parameters. Adds a new parameter and its value to the `params` list.

Usage:

```
.AIFEBaseTransformer$set_model_param(param_name, param_value)
```

Arguments:

`param_name string` Parameter's name.

`param_value any` Parameter's value.

Returns: This method returns nothing.

Method `set_model_temp()`**:** Setter for the temporary model's parameters. Adds a new temporary parameter and its value to the `temp` list.

Usage:

```
.AIFEBaseTransformer$set_model_temp(temp_name, temp_value)
```

Arguments:

`temp_name string` Parameter's name.

`temp_value any` Parameter's value.

Returns: This method returns nothing.

Method `set_SFC_check_max_pos_emb()`**:** Setter for the `check_max_pos_emb` element of the private `steps_for_creation` list. Sets a new `fun` function as the `check_max_pos_emb` step.

Usage:

```
.AIFEBaseTransformer$set_SFC_check_max_pos_emb(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_SFC_create_tokenizer_draft()`**:** Setter for the `create_tokenizer_draft` element of the private `steps_for_creation` list. Sets a new `fun` function as the `create_tokenizer_draft` step.

Usage:

```
.AIFEBaseTransformer$set_SFC_create_tokenizer_draft(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_SFC_calculate_vocab()`**:** Setter for the `calculate_vocab` element of the private `steps_for_creation` list. Sets a new `fun` function as the `calculate_vocab` step.

Usage:

```
.AIFEBaseTransformer$set_SFC_calculate_vocab(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_SFC_save_tokenizer_draft()`**:** Setter for the `save_tokenizer_draft` element of the private `steps_for_creation` list. Sets a new `fun` function as the `save_tokenizer_draft` step.

Usage:

```
.AIFEBaseTransformer$set_SFC_save_tokenizer_draft(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_SFC_create_final_tokenizer()`**:** Setter for the `create_final_tokenizer` element of the private `steps_for_creation` list. Sets a new `fun` function as the `create_final_tokenizer` step.

Usage:

```
.AIFEBaseTransformer$set_SFC_create_final_tokenizer(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_SFC_create_transformer_model()`**:** Setter for the `create_transformer_model` element of the private `steps_for_creation` list. Sets a new `fun` function as the `create_transformer_model` step.

Usage:

```
.AIFEBaseTransformer$set_SFC_create_transformer_model(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_required_SFC()`**:** Setter for all required elements of the private `steps_for_creation` list. Executes setters for all required creation steps.

Usage:

```
.AIFEBaseTransformer$set_required_SFC(required_SFC)
```

Arguments:

`required_SFC list()` A list of all new required steps.

Returns: This method returns nothing.

Method `set_SFT_load_existing_model()`**:** Setter for the `load_existing_model` element of the private `steps_for_training` list. Sets a new `fun` function as the `load_existing_model` step.

Usage:

```
.AIFEBaseTransformer$set_SFT_load_existing_model(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_SFT_cuda_empty_cache()`**:** Setter for the `cuda_empty_cache` element of the private `steps_for_training` list. Sets a new `fun` function as the `cuda_empty_cache` step.

Usage:

```
.AIFEBaseTransformer$set_SFT_cuda_empty_cache(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `set_SFT_create_data_collator()`**:** Setter for the `create_data_collator` element of the private `steps_for_training` list. Sets a new `fun` function as the `create_data_collator` step. Use this method to make a custom data collator for a transformer.

Usage:

```
.AIFEBaseTransformer$set_SFT_create_data_collator(fun)
```

Arguments:

`fun function()` A new function.

Returns: This method returns nothing.

Method `create()`**:** This method creates a transformer configuration based on the child-transformer architecture and a vocabulary using the python libraries `transformers` and `tokenizers`.
This method **adds** the following parameters to the `temp` list:

- log_file
- raw_text_dataset
- pt_safe_save
- value_top
- total_top
- message_top

This method **uses** the following parameters from the temp list:

- log_file
- raw_text_dataset
- tokenizer

Usage:

```
.AIFEBaseTransformer$create(
  ml_framework,
  model_dir,
  text_dataset,
  vocab_size,
  max_position_embeddings,
  hidden_size,
  num_attention_heads,
  intermediate_size,
  hidden_act,
  hidden_dropout_prob,
  attention_probs_dropout_prob,
  sustain_track,
  sustain_iso_code,
  sustain_region,
  sustain_interval,
  trace,
  pytorch_safetensors,
  log_dir,
  log_write_interval
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`model_dir` `string` Path to the directory where the model should be saved.

`text_dataset` Object of class LargeDataSetForText.

`vocab_size` `int` Size of the vocabulary.

`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.

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding.

`num_attention_heads` `int` Number of attention heads.

`intermediate_size` `int` Number of neurons in the intermediate layer of the attention mechanism.

`hidden_act` `string` Name of the activation function.

`hidden_dropout_prob` `double` Ratio of dropout.

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

Method `train()`**:** This method can be used to train or fine-tune a transformer based on `BERT` architecture with the help of the python libraries `transformers`, `datasets`, and `tokenizers`.

This method **adds** the following parameters to the `temp` list:

- `log_file`
- `loss_file`
- `from_pt`
- `from_tf`
- `load_safe`
- `raw_text_dataset`
- `pt_safe_save`
- `value_top`
- `total_top`
- `message_top`

This method **uses** the following parameters from the `temp` list:

- `log_file`
- `raw_text_dataset`
- `tokenized_dataset`
- `tokenizer`

Usage:

```
.AIFEBaseTransformer$train(
  ml_framework,
  output_dir,
  model_dir_path,
  text_dataset,
  p_mask,
  whole_word,
  val_size,
  n_epoch,
  batch_size,
  chunk_size,
  full_sequences_only,
  min_seq_len,
  learning_rate,
  n_workers,
  multi_process,
  sustain_track,
  sustain_iso_code,
  sustain_region,
  sustain_interval,
  trace,
  keras_trace,
  pytorch_trace,
  pytorch_safetensors,
  log_dir,
  log_write_interval
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`output_dir` `string` Path to the directory where the final model should be saved. If the directory does not exist, it will be created.

`model_dir_path` `string` Path to the directory where the original model is stored.

`text_dataset` Object of class LargeDataSetForText.

`p_mask` `double` Ratio that determines the number of words/tokens used for masking.

`whole_word` `bool`

- `TRUE`: whole word masking should be applied.
- `FALSE`: token masking is used.

`val_size` `double` Ratio that determines the amount of token chunks used for validation.

`n_epoch` `int` Number of epochs for training.

`batch_size` `int` Size of batches.

`chunk_size` `int` Size of every chunk for training.

`full_sequences_only` `bool` `TRUE` for using only chunks with a sequence length equal to `chunk_size`.

`min_seq_len` `int` Only relevant if `full_sequences_only = FALSE`. Value determines the minimal sequence length included in training process.

`learning_rate` `double` Learning rate for adam optimizer.

`n_workers` `int` Number of workers. Only relevant if `ml_framework = "tensorflow"`.

`multi_process` `bool` `TRUE` if multiple processes should be activated. Only relevant if `ml_framework = "tensorflow"`.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`keras_trace` `int`

- `keras_trace = 0`: does not print any information about the training process from keras on the console.
- `keras_trace = 1`: prints a progress bar.
- `keras_trace = 2`: prints one line of information for every epoch. Only relevant if `ml_framework = "tensorflow"`.

`pytorch_trace` `int`

- `pytorch_trace = 0`: does not print any information about the training process from pytorch on the console.
- `pytorch_trace = 1`: prints a progress bar.

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

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

Usage:

```
.AIFEBaseTransformer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hugging Face transformers documantation:

- BERT
- DeBERTa
- Funnel
- Longformer
- RoBERTa
- MPNet

See Also

Other Transformers for developers: `.AIFEBertTransformer`, `.AIFEDebertaTransformer`, `.AIFEFunnelTransformer`, `.AIFELongformerTransformer`, `.AIFEMpnetTransformer`, `.AIFERobertaTransformer`, `.AIFETrObj`

`.AIFEBertTransformer` *Child* `R6` *class for creation and training of* `BERT` *transformers*

Description

This class has the following methods:

- `create`: creates a new transformer based on `BERT`.
- `train`: trains and fine-tunes a `BERT` model.

Create

New models can be created using the `.AIFEBertTransformer$create` method.

Train

To train the model, pass the directory of the model to the method `.AIFEBertTransformer$train`.

Pre-Trained models that can be fine-tuned using this method are available at `https://huggingface.co/`.

The model is trained using dynamic masking, as opposed to the original paper, which used static masking.

Super class

`aifeducation::.AIFEBaseTransformer` -> `.AIFEBertTransformer`

Methods

Public methods:

- `.AIFEBertTransformer$new()`
- `.AIFEBertTransformer$create()`
- `.AIFEBertTransformer$train()`
- `.AIFEBertTransformer$clone()`

Method `new()`**:** Creates a new transformer based on `BERT` and sets the title.

Usage:

```
.AIFEBertTransformer$new()
```

Returns: This method returns nothing.

Method `create()`**:** This method creates a transformer configuration based on the `BERT` base architecture and a vocabulary based on `WordPiece` by using the python libraries `transformers` and `tokenizers`.

This method adds the following *'dependent' parameters* to the base class's inherited `params` list:

- `vocab_do_lower_case`
- `num_hidden_layer`

Usage:

```
.AIFEBertTransformer$create(
  ml_framework = "pytorch",
  model_dir,
  text_dataset,
  vocab_size = 30522,
  vocab_do_lower_case = FALSE,
  max_position_embeddings = 512,
  hidden_size = 768,
  num_hidden_layer = 12,
  num_attention_heads = 12,
  intermediate_size = 3072,
  hidden_act = "gelu",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1,
  sustain_track = FALSE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.

- `ml_framework = "pytorch"`: for 'pytorch'.

`model_dir string` Path to the directory where the model should be saved.

`text_dataset` Object of class LargeDataSetForText.

`vocab_size int` Size of the vocabulary.

`vocab_do_lower_case bool` `TRUE` if all words/tokens should be lower case.

`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.

`hidden_size int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding.

`num_hidden_layer int` Number of hidden layers.

`num_attention_heads int` Number of attention heads.

`intermediate_size int` Number of neurons in the intermediate layer of the attention mechanism.

`hidden_act string` Name of the activation function.

`hidden_dropout_prob double` Ratio of dropout.

`attention_probs_dropout_prob double` Ratio of dropout for attention probabilities.

`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.

`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.

`sustain_interval integer` Interval in seconds for measuring power usage.

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

`pytorch_safetensors bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

Method `train()`**:** This method can be used to train or fine-tune a transformer based on `BERT` architecture with the help of the python libraries `transformers`, `datasets`, and `tokenizers`.

Usage:

```
.AIFEBertTransformer$train(
  ml_framework = "pytorch",
  output_dir,
  model_dir_path,
  text_dataset,
```

```
  p_mask = 0.15,
  whole_word = TRUE,
  val_size = 0.1,
  n_epoch = 1,
  batch_size = 12,
  chunk_size = 250,
  full_sequences_only = FALSE,
  min_seq_len = 50,
  learning_rate = 0.003,
  n_workers = 1,
  multi_process = FALSE,
  sustain_track = FALSE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  keras_trace = 1,
  pytorch_trace = 1,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`output_dir` `string` Path to the directory where the final model should be saved. If the directory does not exist, it will be created.

`model_dir_path` `string` Path to the directory where the original model is stored.

`text_dataset` Object of class LargeDataSetForText.

`p_mask` `double` Ratio that determines the number of words/tokens used for masking.

`whole_word` `bool`

- `TRUE`: whole word masking should be applied.
- `FALSE`: token masking is used.

`val_size` `double` Ratio that determines the amount of token chunks used for validation.

`n_epoch` `int` Number of epochs for training.

`batch_size` `int` Size of batches.

`chunk_size` `int` Size of every chunk for training.

`full_sequences_only` `bool` `TRUE` for using only chunks with a sequence length equal to `chunk_size`.

`min_seq_len` `int` Only relevant if `full_sequences_only = FALSE`. Value determines the minimal sequence length included in training process.

`learning_rate` `double` Learning rate for adam optimizer.

`n_workers` `int` Number of workers. Only relevant if `ml_framework = "tensorflow"`.

`multi_process` `bool` `TRUE` if multiple processes should be activated. Only relevant if `ml_framework = "tensorflow"`.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`keras_trace` `int`

- `keras_trace` = `0`: does not print any information about the training process from keras on the console.
- `keras_trace` = `1`: prints a progress bar.
- `keras_trace` = `2`: prints one line of information for every epoch. Only relevant if `ml_framework` = `"tensorflow"`.

`pytorch_trace` `int`

- `pytorch_trace` = `0`: does not print any information about the training process from pytorch on the console.
- `pytorch_trace` = `1`: prints a progress bar.

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead the trained or fine-tuned model is saved to disk.

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

Usage:

```
.AIFEBertTransformer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

This model uses a `WordPiece` tokenizer like `BERT` and can be trained with whole word masking. The transformer library may display a warning, which can be ignored.

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

Hugging Face documentation

- https://huggingface.co/docs/transformers/model_doc/bert
- https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertForMaskedLM
- https://huggingface.co/docs/transformers/model_doc/bert#transformers.TFBertForMaskedLM

See Also

Other Transformers for developers: .AIFEBaseTransformer, .AIFEDebertaTransformer, .AIFEFunnelTransformer, .AIFELongformerTransformer, .AIFEMpnetTransformer, .AIFERobertaTransformer, .AIFETrObj

---

.AIFEDebertaTransformer

Child R6 *class for creation and training of* DeBERTa-V2 *transformers*

Description

This class has the following methods:

- create: creates a new transformer based on DeBERTa-V2.
- train: trains and fine-tunes a DeBERTa-V2 model.

Create

New models can be created using the .AIFEDebertaTransformer$create method.

Train

To train the model, pass the directory of the model to the method .AIFEDebertaTransformer$train.

Pre-Trained models which can be fine-tuned with this function are available at https://huggingface.co/.

Training of this model makes use of dynamic masking.

Super class

aifeducation::.AIFEBaseTransformer -> .AIFEDebertaTransformer

Methods

Public methods:

- .AIFEDebertaTransformer$new()
- .AIFEDebertaTransformer$create()
- .AIFEDebertaTransformer$train()
- .AIFEDebertaTransformer$clone()

Method `new()`**:** Creates a new transformer based on `DeBERTa-V2` and sets the title.

Usage:

```
.AIFEDebertaTransformer$new()
```

Returns: This method returns nothing.

Method `create()`**:** This method creates a transformer configuration based on the `DeBERTa-V2` base architecture and a vocabulary based on the `SentencePiece` tokenizer using the python `transformers` and `tokenizers` libraries.

This method adds the following *'dependent' parameters* to the base class's inherited `params` list:

- `vocab_do_lower_case`
- `num_hidden_layer`

Usage:

```
.AIFEDebertaTransformer$create(
  ml_framework = "pytorch",
  model_dir,
  text_dataset,
  vocab_size = 128100,
  vocab_do_lower_case = FALSE,
  max_position_embeddings = 512,
  hidden_size = 1536,
  num_hidden_layer = 24,
  num_attention_heads = 24,
  intermediate_size = 6144,
  hidden_act = "gelu",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.

- `ml_framework = "pytorch"`: for 'pytorch'.

`model_dir string` Path to the directory where the model should be saved.

`text_dataset` Object of class LargeDataSetForText.

`vocab_size int` Size of the vocabulary.

`vocab_do_lower_case bool` `TRUE` if all words/tokens should be lower case.

`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.

`hidden_size int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding.

`num_hidden_layer int` Number of hidden layers.

`num_attention_heads int` Number of attention heads.

`intermediate_size int` Number of neurons in the intermediate layer of the attention mechanism.

`hidden_act string` Name of the activation function.

`hidden_dropout_prob double` Ratio of dropout.

`attention_probs_dropout_prob double` Ratio of dropout for attention probabilities.

`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.

`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.

`sustain_interval integer` Interval in seconds for measuring power usage.

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

`pytorch_safetensors bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

Method `train()`**:** This method can be used to train or fine-tune a transformer based on `DeBERTa-V2` architecture with the help of the python libraries `transformers`, `datasets`, and `tokenizers`.

Usage:

```
.AIFEDebertaTransformer$train(
  ml_framework = "pytorch",
  output_dir,
  model_dir_path,
  text_dataset,
```

```
  p_mask = 0.15,
  whole_word = TRUE,
  val_size = 0.1,
  n_epoch = 1,
  batch_size = 12,
  chunk_size = 250,
  full_sequences_only = FALSE,
  min_seq_len = 50,
  learning_rate = 0.03,
  n_workers = 1,
  multi_process = FALSE,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  keras_trace = 1,
  pytorch_trace = 1,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`output_dir` `string` Path to the directory where the final model should be saved. If the directory does not exist, it will be created.

`model_dir_path` `string` Path to the directory where the original model is stored.

`text_dataset` Object of class LargeDataSetForText.

`p_mask` `double` Ratio that determines the number of words/tokens used for masking.

`whole_word` `bool`

- `TRUE`: whole word masking should be applied.
- `FALSE`: token masking is used.

`val_size` `double` Ratio that determines the amount of token chunks used for validation.

`n_epoch` `int` Number of epochs for training.

`batch_size` `int` Size of batches.

`chunk_size` `int` Size of every chunk for training.

`full_sequences_only` `bool` `TRUE` for using only chunks with a sequence length equal to `chunk_size`.

`min_seq_len` `int` Only relevant if `full_sequences_only = FALSE`. Value determines the minimal sequence length included in training process.

`learning_rate` `double` Learning rate for adam optimizer.

`n_workers` `int` Number of workers. Only relevant if `ml_framework = "tensorflow"`.

`multi_process` `bool` `TRUE` if multiple processes should be activated. Only relevant if `ml_framework = "tensorflow"`.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`keras_trace` `int`

- `keras_trace` = `0`: does not print any information about the training process from keras on the console.
- `keras_trace` = `1`: prints a progress bar.
- `keras_trace` = `2`: prints one line of information for every epoch. Only relevant if `ml_framework` = `"tensorflow"`.

`pytorch_trace` `int`

- `pytorch_trace` = `0`: does not print any information about the training process from pytorch on the console.
- `pytorch_trace` = `1`: prints a progress bar.

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead the trained or fine-tuned model is saved to disk.

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

Usage:

```
.AIFEDebertaTransformer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

For this model a `WordPiece` tokenizer is created. The standard implementation of `DeBERTa` version 2 from HuggingFace uses a `SentencePiece` tokenizer. Thus, please use `AutoTokenizer` from the `transformers` library to work with this model.

References

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

Hugging Face documentatio

- https://huggingface.co/docs/transformers/model_doc/deberta-v2
- https://huggingface.co/docs/transformers/model_doc/deberta-v2#transformers.DebertaV2ForMaskedLM
- https://huggingface.co/docs/transformers/model_doc/deberta-v2#transformers.TFDebertaV2ForMaskedLM

See Also

Other Transformers for developers: .AIFEBaseTransformer, .AIFEBertTransformer, .AIFEFunnelTransformer, .AIFELongformerTransformer, .AIFEMpnetTransformer, .AIFERobertaTransformer, .AIFETrObj

---

.AIFEFunnelTransformer

Child R6 *class for creation and training of* Funnel *transformers*

Description

This class has the following methods:

- `create`: creates a new transformer based on `Funnel`.
- `train`: trains and fine-tunes a `Funnel` model.

Create

New models can be created using the `.AIFEFunnelTransformer$create` method.

Model is created with `separete_cls` = `TRUE`, `truncate_seq` = `TRUE`, and `pool_q_only` = `TRUE`.

Train

To train the model, pass the directory of the model to the method `.AIFEFunnelTransformer$train`.

Pre-Trained models which can be fine-tuned with this function are available at https://huggingface.co/.

Training of the model makes use of dynamic masking.

Super class

aifeducation::.AIFEBaseTransformer -> .AIFEFunnelTransformer

Methods

Public methods:

- `.AIFEFunnelTransformer$new()`
- `.AIFEFunnelTransformer$create()`
- `.AIFEFunnelTransformer$train()`
- `.AIFEFunnelTransformer$clone()`

Method `new()`**:** Creates a new transformer based on `Funnel` and sets the title.

Usage:

```
.AIFEFunnelTransformer$new()
```

Returns: This method returns nothing.

Method `create()`**:** This method creates a transformer configuration based on the `Funnel` transformer base architecture and a vocabulary based on `WordPiece` using the python `transformers` and `tokenizers` libraries.

This method adds the following *'dependent' parameters* to the base class's inherited `params` list:

- `vocab_do_lower_case`
- `target_hidden_size`
- `block_sizes`
- `num_decoder_layers`
- `pooling_type`
- `activation_dropout`

Usage:

```
.AIFEFunnelTransformer$create(
  ml_framework = "pytorch",
  model_dir,
  text_dataset,
  vocab_size = 30522,
  vocab_do_lower_case = FALSE,
  max_position_embeddings = 512,
  hidden_size = 768,
  target_hidden_size = 64,
  block_sizes = c(4, 4, 4),
  num_attention_heads = 12,
  intermediate_size = 3072,
  num_decoder_layers = 2,
  pooling_type = "mean",
  hidden_act = "gelu",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1,
  activation_dropout = 0,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
```

```
  trace = TRUE,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`model_dir` `string` Path to the directory where the model should be saved.

`text_dataset` Object of class LargeDataSetForText.

`vocab_size` `int` Size of the vocabulary.

`vocab_do_lower_case` `bool` `TRUE` if all words/tokens should be lower case.

`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.

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding.

`target_hidden_size` `int` Number of neurons in the final layer. This parameter determines the dimensionality of the resulting text embedding.

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

`num_attention_heads` `int` Number of attention heads.

`intermediate_size` `int` Number of neurons in the intermediate layer of the attention mechanism.

`num_decoder_layers` `int` Number of decoding layers.

`pooling_type` `string` Type of pooling.

- `"mean"` for pooling with mean.
- `"max"` for pooling with maximum values.

`hidden_act` `string` Name of the activation function.

`hidden_dropout_prob` `double` Ratio of dropout.

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities.

`activation_dropout` `float` Dropout probability between the layers of the feed-forward blocks.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.

- FALSE (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

log_dir Path to the directory where the log files should be saved.

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.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

Method train(): This method can be used to train or fine-tune a transformer based on Funnel Transformer architecture with the help of the python libraries transformers, datasets, and tokenizers.

Usage:

```
.AIFEFunnelTransformer$train(
  ml_framework = "pytorch",
  output_dir,
  model_dir_path,
  text_dataset,
  p_mask = 0.15,
  whole_word = TRUE,
  val_size = 0.1,
  n_epoch = 1,
  batch_size = 12,
  chunk_size = 250,
  full_sequences_only = FALSE,
  min_seq_len = 50,
  learning_rate = 0.003,
  n_workers = 1,
  multi_process = FALSE,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  keras_trace = 1,
  pytorch_trace = 1,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

ml_framework string Framework to use for training and inference.

- ml_framework = "tensorflow": for 'tensorflow'.
- ml_framework = "pytorch": for 'pytorch'.

output_dir string Path to the directory where the final model should be saved. If the directory does not exist, it will be created.

model_dir_path string Path to the directory where the original model is stored.

`text_dataset` Object of class LargeDataSetForText.

`p_mask` `double` Ratio that determines the number of words/tokens used for masking.

`whole_word` `bool`

- `TRUE`: whole word masking should be applied.
- `FALSE`: token masking is used.

`val_size` `double` Ratio that determines the amount of token chunks used for validation.

`n_epoch` `int` Number of epochs for training.

`batch_size` `int` Size of batches.

`chunk_size` `int` Size of every chunk for training.

`full_sequences_only` `bool` `TRUE` for using only chunks with a sequence length equal to `chunk_size`.

`min_seq_len` `int` Only relevant if `full_sequences_only` = `FALSE`. Value determines the minimal sequence length included in training process.

`learning_rate` `double` Learning rate for adam optimizer.

`n_workers` `int` Number of workers. Only relevant if `ml_framework` = `"tensorflow"`.

`multi_process` `bool` `TRUE` if multiple processes should be activated. Only relevant if `ml_framework` = `"tensorflow"`.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`keras_trace` `int`

- `keras_trace` = `0`: does not print any information about the training process from keras on the console.
- `keras_trace` = `1`: prints a progress bar.
- `keras_trace` = `2`: prints one line of information for every epoch. Only relevant if `ml_framework` = `"tensorflow"`.

`pytorch_trace` `int`

- `pytorch_trace` = `0`: does not print any information about the training process from pytorch on the console.
- `pytorch_trace` = `1`: prints a progress bar.

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead the trained or fine-tuned model is saved to disk.

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

Usage:

```
.AIFEFunnelTransformer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

The model uses a configuration with `truncate_seq = TRUE` to avoid implementation problems with tensorflow.

This model uses a `WordPiece` tokenizer like `BERT` and can be trained with whole word masking. The transformer library may display a warning, which can be ignored.

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

Hugging Face documentation

- https://huggingface.co/docs/transformers/model_doc/funnel#funnel-transformer
- https://huggingface.co/docs/transformers/model_doc/funnel#transformers.FunnelModel
- https://huggingface.co/docs/transformers/model_doc/funnel#transformers.TFFunnelModel

See Also

Other Transformers for developers: `.AIFEBaseTransformer`, `.AIFEBertTransformer`, `.AIFEDebertaTransformer`, `.AIFELongformerTransformer`, `.AIFEMpnetTransformer`, `.AIFERobertaTransformer`, `.AIFETrObj`

.AIFELongformerTransformer

Child `R6` *class for creation and training of* `Longformer` *transformers*

Description

This class has the following methods:

- `create`: creates a new transformer based on `Longformer`.
- `train`: trains and fine-tunes a `Longformer` model.

Create

New models can be created using the `.AIFELongformerTransformer$create` method.

Train

To train the model, pass the directory of the model to the method `.AIFELongformerTransformer$train`.

Pre-Trained models which can be fine-tuned with this function are available at https://huggingface.co/.

Training of this model makes use of dynamic masking.

Super class

`aifeducation::.AIFEBaseTransformer` -> `.AIFELongformerTransformer`

Methods

Public methods:

- `.AIFELongformerTransformer$new()`
- `.AIFELongformerTransformer$create()`
- `.AIFELongformerTransformer$train()`
- `.AIFELongformerTransformer$clone()`

Method `new()`**:** Creates a new transformer based on `Longformer` and sets the title.

Usage:

```
.AIFELongformerTransformer$new()
```

Returns: This method returns nothing

Method `create()`**:** This method creates a transformer configuration based on the `Longformer` base architecture and a vocabulary based on `Byte-Pair Encoding` (BPE) tokenizer using the python `transformers` and `tokenizers` libraries.

This method adds the following *'dependent' parameters* to the base class's inherited `params` list:

- `add_prefix_space`
- `trim_offsets`
- `num_hidden_layer`
- `attention_window`

Usage:

```
.AIFELongformerTransformer$create(
  ml_framework = "pytorch",
  model_dir,
  text_dataset,
  vocab_size = 30522,
  add_prefix_space = FALSE,
  trim_offsets = TRUE,
  max_position_embeddings = 512,
  hidden_size = 768,
  num_hidden_layer = 12,
  num_attention_heads = 12,
  intermediate_size = 3072,
  hidden_act = "gelu",
```

```
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1,
  attention_window = 512,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`model_dir` `string` Path to the directory where the model should be saved.

`text_dataset` Object of class LargeDataSetForText.

`vocab_size` `int` Size of the vocabulary.

`add_prefix_space` `bool` `TRUE` if an additional space should be inserted to the leading words.

`trim_offsets` `bool` `TRUE` trims the whitespaces from the produced offsets.

`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.

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding.

`num_hidden_layer` `int` Number of hidden layers.

`num_attention_heads` `int` Number of attention heads.

`intermediate_size` `int` Number of neurons in the intermediate layer of the attention mechanism.

`hidden_act` `string` Name of the activation function.

`hidden_dropout_prob` `double` Ratio of dropout.

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities.

`attention_window` `int` Size of the window around each token for attention mechanism in every layer.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- TRUE: a 'pytorch' model is saved in safetensors format.
- FALSE (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

log_dir Path to the directory where the log files should be saved.

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.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

Method train()**:** This method can be used to train or fine-tune a transformer based on Longformer Transformer architecture with the help of the python libraries transformers, datasets, and tokenizers.

Usage:

```
.AIFELongformerTransformer$train(
  ml_framework = "pytorch",
  output_dir,
  model_dir_path,
  text_dataset,
  p_mask = 0.15,
  val_size = 0.1,
  n_epoch = 1,
  batch_size = 12,
  chunk_size = 250,
  full_sequences_only = FALSE,
  min_seq_len = 50,
  learning_rate = 0.03,
  n_workers = 1,
  multi_process = FALSE,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  keras_trace = 1,
  pytorch_trace = 1,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

ml_framework string Framework to use for training and inference.

- ml_framework = "tensorflow": for 'tensorflow'.
- ml_framework = "pytorch": for 'pytorch'.

output_dir string Path to the directory where the final model should be saved. If the directory does not exist, it will be created.

model_dir_path string Path to the directory where the original model is stored.

`text_dataset` Object of class LargeDataSetForText.

`p_mask` `double` Ratio that determines the number of words/tokens used for masking.

`val_size` `double` Ratio that determines the amount of token chunks used for validation.

`n_epoch` `int` Number of epochs for training.

`batch_size` `int` Size of batches.

`chunk_size` `int` Size of every chunk for training.

`full_sequences_only` `bool` `TRUE` for using only chunks with a sequence length equal to `chunk_size`.

`min_seq_len` `int` Only relevant if `full_sequences_only` = `FALSE`. Value determines the minimal sequence length included in training process.

`learning_rate` `double` Learning rate for adam optimizer.

`n_workers` `int` Number of workers. Only relevant if `ml_framework` = `"tensorflow"`.

`multi_process` `bool` `TRUE` if multiple processes should be activated. Only relevant if `ml_framework` = `"tensorflow"`.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`keras_trace` `int`

- `keras_trace` = `0`: does not print any information about the training process from keras on the console.
- `keras_trace` = `1`: prints a progress bar.
- `keras_trace` = `2`: prints one line of information for every epoch. Only relevant if `ml_framework` = `"tensorflow"`.

`pytorch_trace` `int`

- `pytorch_trace` = `0`: does not print any information about the training process from pytorch on the console.
- `pytorch_trace` = `1`: prints a progress bar.

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead the trained or fine-tuned model is saved to disk.

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

Usage:

```
.AIFELongformerTransformer$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The Long-Document Transformer. doi:10.48550/arXiv.2004.05150

Hugging Face Documentation

- https://huggingface.co/docs/transformers/model_doc/longformer
- https://huggingface.co/docs/transformers/model_doc/longformer#transformers.LongformerModel
- https://huggingface.co/docs/transformers/model_doc/longformer#transformers.TFLongformerModel

See Also

Other Transformers for developers: .AIFEBaseTransformer, .AIFEBertTransformer, .AIFEDebertaTransformer, .AIFEFunnelTransformer, .AIFEMpnetTransformer, .AIFERobertaTransformer, .AIFETrObj

---

.AIFEMpnetTransformer *Child* R6 *class for creation and training of* MPNet *transformers*

Description

This class has the following methods:

- create: creates a new transformer based on MPNet.
- train: trains and fine-tunes a MPNet model.

Create

New models can be created using the .AIFEMpnetTransformer$create method.

Train

To train the model, pass the directory of the model to the method .AIFEMpnetTransformer$train.

Super class

aifeducation::.AIFEBaseTransformer -> .AIFEMpnetTransformer

Public fields

`special_tokens_list list` List for special tokens with the following elements:

- `cls` - CLS token representation (<s>)
- `pad` - pad token representation (<pad>)
- `sep` - sep token representation (</s>)
- `unk` - unk token representation (<unk>)
- `mask` - mask token representation (<mask>)

Methods

Public methods:

- `.AIFEMpnetTransformer$new()`
- `.AIFEMpnetTransformer$create()`
- `.AIFEMpnetTransformer$train()`
- `.AIFEMpnetTransformer$clone()`

Method `new()`**:** Creates a new transformer based on `MPNet` and sets the title.

Usage:

```
.AIFEMpnetTransformer$new()
```

Returns: This method returns nothing.

Method `create()`**:** This method creates a transformer configuration based on the `MPNet` base architecture.

This method adds the following *'dependent' parameters* to the base class's inherited `params` list:

- `vocab_do_lower_case`
- `num_hidden_layer`

Usage:

```
.AIFEMpnetTransformer$create(
  ml_framework = "pytorch",
  model_dir,
  text_dataset,
  vocab_size = 30522,
  vocab_do_lower_case = FALSE,
  max_position_embeddings = 512,
  hidden_size = 768,
  num_hidden_layer = 12,
  num_attention_heads = 12,
  intermediate_size = 3072,
  hidden_act = "gelu",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1,
  sustain_track = FALSE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
```

```
  trace = TRUE,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`model_dir` `string` Path to the directory where the model should be saved.

`text_dataset` Object of class LargeDataSetForText.

`vocab_size` `int` Size of the vocabulary.

`vocab_do_lower_case` `bool` `TRUE` if all words/tokens should be lower case.

`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.

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding.

`num_hidden_layer` `int` Number of hidden layers.

`num_attention_heads` `int` Number of attention heads.

`intermediate_size` `int` Number of neurons in the intermediate layer of the attention mechanism.

`hidden_act` `string` Name of the activation function.

`hidden_dropout_prob` `double` Ratio of dropout.

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

Method `train()`**:** This method can be used to train or fine-tune a transformer based on `MPNet` architecture with the help of the python libraries `transformers`, `datasets`, and `tokenizers`. This method adds the following *'dependent' parameter* to the base class's inherited `params` list:

- `p_perm`

Usage:

```
.AIFEMpnetTransformer$train(
  ml_framework = "pytorch",
  output_dir,
  model_dir_path,
  text_dataset,
  p_mask = 0.15,
  p_perm = 0.15,
  whole_word = TRUE,
  val_size = 0.1,
  n_epoch = 1,
  batch_size = 12,
  chunk_size = 250,
  full_sequences_only = FALSE,
  min_seq_len = 50,
  learning_rate = 0.003,
  n_workers = 1,
  multi_process = FALSE,
  sustain_track = FALSE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  keras_trace = 1,
  pytorch_trace = 1,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`output_dir` `string` Path to the directory where the final model should be saved. If the directory does not exist, it will be created.

`model_dir_path` `string` Path to the directory where the original model is stored.

`text_dataset` Object of class LargeDataSetForText.

`p_mask` `double` Ratio that determines the number of words/tokens used for masking.

`p_perm` `double` Ratio that determines the number of words/tokens used for permutation.

`whole_word` `bool`

- `TRUE`: whole word masking should be applied.
- `FALSE`: token masking is used.

`val_size` `double` Ratio that determines the amount of token chunks used for validation.

`n_epoch` `int` Number of epochs for training.

`batch_size` `int` Size of batches.

`chunk_size` `int` Size of every chunk for training.

`full_sequences_only` `bool` `TRUE` for using only chunks with a sequence length equal to `chunk_size`.

`min_seq_len` `int` Only relevant if `full_sequences_only` = `FALSE`. Value determines the minimal sequence length included in training process.

`learning_rate` `double` Learning rate for adam optimizer.

`n_workers` `int` Number of workers. Only relevant if `ml_framework` = `"tensorflow"`.

`multi_process` `bool` `TRUE` if multiple processes should be activated. Only relevant if `ml_framework` = `"tensorflow"`.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`keras_trace` `int`

- `keras_trace` = `0`: does not print any information about the training process from keras on the console.
- `keras_trace` = `1`: prints a progress bar.
- `keras_trace` = `2`: prints one line of information for every epoch. Only relevant if `ml_framework` = `"tensorflow"`.

`pytorch_trace` `int`

- `pytorch_trace` = `0`: does not print any information about the training process from pytorch on the console.
- `pytorch_trace` = `1`: prints a progress bar.

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead the trained or fine-tuned model is saved to disk.

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

Usage:

```
.AIFEMpnetTransformer$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Note

Using this class with `tensorflow` is not supported. Supported framework is `pytorch`.

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

Hugging Face documentation

- https://huggingface.co/docs/transformers/model_doc/mpnet
- https://huggingface.co/docs/transformers/model_doc/mpnet#transformers.MPNetForMaskedLM
- https://huggingface.co/docs/transformers/model_doc/mpnet#transformers.TFMPNetForMaskedLM

See Also

Other Transformers for developers: `.AIFEBaseTransformer`, `.AIFEBertTransformer`, `.AIFEDebertaTransformer`, `.AIFEFunnelTransformer`, `.AIFELongformerTransformer`, `.AIFERobertaTransformer`, `.AIFETrObj`

.AIFERobertaTransformer

Child `R6` *class for creation and training of* `RoBERTa` *transformers*

Description

This class has the following methods:

- `create`: creates a new transformer based on `RoBERTa`.
- `train`: trains and fine-tunes a `RoBERTa` model.

Create

New models can be created using the `.AIFERobertaTransformer$create` method.

Train

To train the model, pass the directory of the model to the method `.AIFERobertaTransformer$train`.

Pre-Trained models which can be fine-tuned with this function are available at https://huggingface.co/.

Training of this model makes use of dynamic masking.

Super class

aifeducation::.AIFEBaseTransformer -> .AIFERobertaTransformer

Methods

Public methods:

- .AIFERobertaTransformer$new()
- .AIFERobertaTransformer$create()
- .AIFERobertaTransformer$train()
- .AIFERobertaTransformer$clone()

Method new(): Creates a new transformer based on RoBERTa and sets the title.

Usage:

```
.AIFERobertaTransformer$new()
```

Returns: This method returns nothing.

Method create(): This method creates a transformer configuration based on the RoBERTa base architecture and a vocabulary based on Byte-Pair Encoding (BPE) tokenizer using the python transformers and tokenizers libraries.

This method adds the following *'dependent' parameters* to the base class' inherited params list:

- add_prefix_space
- trim_offsets
- num_hidden_layer

Usage:

```
.AIFERobertaTransformer$create(
  ml_framework = "pytorch",
  model_dir,
  text_dataset,
  vocab_size = 30522,
  add_prefix_space = FALSE,
  trim_offsets = TRUE,
  max_position_embeddings = 512,
  hidden_size = 768,
  num_hidden_layer = 12,
  num_attention_heads = 12,
  intermediate_size = 3072,
  hidden_act = "gelu",
  hidden_dropout_prob = 0.1,
  attention_probs_dropout_prob = 0.1,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
```

```
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`model_dir` `string` Path to the directory where the model should be saved.

`text_dataset` Object of class LargeDataSetForText.

`vocab_size` `int` Size of the vocabulary.

`add_prefix_space` `bool` `TRUE` if an additional space should be inserted to the leading words.

`trim_offsets` `bool` `TRUE` trims the whitespaces from the produced offsets.

`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.

`hidden_size` `int` Number of neurons in each layer. This parameter determines the dimensionality of the resulting text embedding.

`num_hidden_layer` `int` Number of hidden layers.

`num_attention_heads` `int` Number of attention heads.

`intermediate_size` `int` Number of neurons in the intermediate layer of the attention mechanism.

`hidden_act` `string` Name of the activation function.

`hidden_dropout_prob` `double` Ratio of dropout.

`attention_probs_dropout_prob` `double` Ratio of dropout for attention probabilities.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead, it saves the configuration and vocabulary of the new model to disk.

Method `train()`**:** This method can be used to train or fine-tune a transformer based on `RoBERTa` Transformer architecture with the help of the python libraries `transformers`, `datasets`, and `tokenizers`.

Usage:

```
.AIFERobertaTransformer$train(
  ml_framework = "pytorch",
  output_dir,
  model_dir_path,
  text_dataset,
  p_mask = 0.15,
  val_size = 0.1,
  n_epoch = 1,
  batch_size = 12,
  chunk_size = 250,
  full_sequences_only = FALSE,
  min_seq_len = 50,
  learning_rate = 0.03,
  n_workers = 1,
  multi_process = FALSE,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  trace = TRUE,
  keras_trace = 1,
  pytorch_trace = 1,
  pytorch_safetensors = TRUE,
  log_dir = NULL,
  log_write_interval = 2
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference.

- `ml_framework = "tensorflow"`: for 'tensorflow'.
- `ml_framework = "pytorch"`: for 'pytorch'.

`output_dir` `string` Path to the directory where the final model should be saved. If the directory does not exist, it will be created.

`model_dir_path` `string` Path to the directory where the original model is stored.

`text_dataset` Object of class LargeDataSetForText.

`p_mask` `double` Ratio that determines the number of words/tokens used for masking.

`val_size` `double` Ratio that determines the amount of token chunks used for validation.

`n_epoch` `int` Number of epochs for training.

`batch_size` `int` Size of batches.

`chunk_size` `int` Size of every chunk for training.

`full_sequences_only` `bool` `TRUE` for using only chunks with a sequence length equal to `chunk_size`.

`min_seq_len` `int` Only relevant if `full_sequences_only = FALSE`. Value determines the minimal sequence length included in training process.

`learning_rate` `double` Learning rate for adam optimizer.

`n_workers` `int` Number of workers. Only relevant if `ml_framework = "tensorflow"`.

`multi_process` `bool` `TRUE` if multiple processes should be activated. Only relevant if `ml_framework = "tensorflow"`.

`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.

`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.

`sustain_interval` `integer` Interval in seconds for measuring power usage.

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

`keras_trace` `int`

- `keras_trace = 0`: does not print any information about the training process from keras on the console.
- `keras_trace = 1`: prints a progress bar.
- `keras_trace = 2`: prints one line of information for every epoch. Only relevant if `ml_framework = "tensorflow"`.

`pytorch_trace` `int`

- `pytorch_trace = 0`: does not print any information about the training process from pytorch on the console.
- `pytorch_trace = 1`: prints a progress bar.

`pytorch_safetensors` `bool` Only relevant for pytorch models.

- `TRUE`: a 'pytorch' model is saved in safetensors format.
- `FALSE` (or 'safetensors' is not available): model is saved in the standard pytorch format (.bin).

`log_dir` Path to the directory where the log files should be saved.

`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`.

Returns: This method does not return an object. Instead the trained or fine-tuned model is saved to disk.

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

Usage:

```
.AIFERobertaTransformer$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

Hugging Face Documentation

- https://huggingface.co/docs/transformers/model_doc/roberta
- https://huggingface.co/docs/transformers/model_doc/roberta#transformers.RobertaModel
- https://huggingface.co/docs/transformers/model_doc/roberta#transformers.TFRobertaModel

See Also

Other Transformers for developers: .AIFEBaseTransformer, .AIFEBertTransformer, .AIFEDebertaTransformer, .AIFEFunnelTransformer, .AIFELongformerTransformer, .AIFEMpnetTransformer, .AIFETrObj

---

AIFEBaseModel *Base class for models using neural nets*

Description

Abstract class for all models that do not rely on the python library 'transformers'.

Value

Objects of this 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.

Public fields

`model` ('tensorflow_model' or 'pytorch_model')
: Field for storing the 'tensorflow' or 'pytorch' model after loading.

`model_config` ('list()')
: List for storing information about the configuration of the model.

`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:

- AIFEBaseModel$get_model_info()
- AIFEBaseModel$get_text_embedding_model()
- AIFEBaseModel$set_publication_info()

- AIFEBaseModel$get_publication_info()
- AIFEBaseModel$set_model_license()
- AIFEBaseModel$get_model_license()
- AIFEBaseModel$set_documentation_license()
- AIFEBaseModel$get_documentation_license()
- AIFEBaseModel$set_model_description()
- AIFEBaseModel$get_model_description()
- AIFEBaseModel$save()
- AIFEBaseModel$load()
- AIFEBaseModel$get_package_versions()
- AIFEBaseModel$get_sustainability_data()
- AIFEBaseModel$get_ml_framework()
- AIFEBaseModel$get_text_embedding_model_name()
- AIFEBaseModel$check_embedding_model()
- AIFEBaseModel$count_parameter()
- AIFEBaseModel$is_configured()
- AIFEBaseModel$get_private()
- AIFEBaseModel$get_all_fields()
- AIFEBaseModel$clone()

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

Usage:

```
AIFEBaseModel$get_model_info()
```

Returns: `list` of all relevant model information.

Method `get_text_embedding_model()`**:** Method for requesting the text embedding model information.

Usage:

```
AIFEBaseModel$get_text_embedding_model()
```

Returns: `list` of all relevant model information on the text embedding model underlying the model.

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

Usage:

```
AIFEBaseModel$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:

```
AIFEBaseModel$get_publication_info()
```

Returns: `list` with all saved bibliographic information.

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

Usage:

```
AIFEBaseModel$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:

```
AIFEBaseModel$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:

```
AIFEBaseModel$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:

```
AIFEBaseModel$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:

```
AIFEBaseModel$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:

```
AIFEBaseModel$get_model_description()
```

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

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

Usage:

```
AIFEBaseModel$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()`**:** Method for importing a model.

Usage:

```
AIFEBaseModel$load(dir_path)
```

Arguments:

`dir_path string` Path of the directory where the model is saved.

Returns: Function does not return a value. It is used to load the weights of a model.

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

Usage:

```
AIFEBaseModel$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:

```
AIFEBaseModel$get_sustainability_data()
```

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:

```
AIFEBaseModel$get_ml_framework()
```

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

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

Usage:

```
AIFEBaseModel$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:

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

Usage:

```
AIFEBaseModel$is_configured()
```

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

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

Usage:

```
AIFEBaseModel$get_private()
```

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

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

Usage:

```
AIFEBaseModel$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:

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

Arguments:

`deep` Whether to make a deep clone.

`AIFETransformerMaker` `R6` *class for transformer creation*

Description

This class was developed to make the creation of transformers easier for users. Pass the transformer's type to the `make` method and get desired transformer. Now run the `create` or/and `train` methods of the new transformer.

The already created aife_transformer_maker object of this class can be used.

See p.3 Transformer Maker in Transformers for Developers for details.

See .AIFEBaseTransformer class for details.

Methods

Public methods:

- `AIFETransformerMaker$make()`
- `AIFETransformerMaker$clone()`

Method `make()`**:** Creates a new transformer with the passed type.

Usage:

```
AIFETransformerMaker$make(type)
```

Arguments:

`type` `string` A type of the new transformer. Allowed types are bert, roberta, deberta_v2, funnel, longformer, mpnet. See AIFETrType list.

Returns: If success - a new transformer, otherwise - an error (passed type is invalid).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Transformer: `AIFETrType`, `aife_transformer_maker`

Examples

```
# Create transformer maker
tr_maker <- AIFETransformerMaker$new()

# Use 'make' method of the 'tr_maker' object
# Pass string with the type of transformers
# Allowed types are "bert", "deberta_v2", "funnel", etc. See aifeducation::AIFETrType list
my_bert <- tr_maker$make("bert")

# Or use elements of the 'aifeducation::AIFETrType' list
my_longformer <- tr_maker$make(AIFETrType$longformer)

# Run 'create' or 'train' methods of the transformer in order to create a
# new transformer or train the newly created one, respectively
# my_bert$create(...)
# my_bert$train(...)

# my_longformer$create(...)
# my_longformer$train(...)
```

AIFETrType *Transformer types*

Description

This list contains transformer types. Elements of the list can be used in the public `make` of the AIFETransformerMaker `R6` class as input parameter `type`.

It has the following elements:

- `bert` = 'bert'
- `roberta` = 'roberta'
- `deberta_v2` = 'deberta_v2'
- `funnel` = 'funnel'
- `longformer` = 'longformer'
- `mpnet` = 'mpnet'

  Elements can be used like `AIFETrType$bert`, `AIFETrType$deberta_v2`, `AIFETrType$funnel`, etc.

Usage

```
AIFETrType
```

Format

An object of class `list` of length 6.

See Also

Other Transformer: AIFETransformerMaker, aife_transformer_maker

---

`aife_transformer_maker` R6 *object of the* `AIFETransformerMaker` *class*

Description

Object for creating the transformers with different types. See AIFETransformerMaker class for details.

Usage

```
aife_transformer_maker
```

Format

An object of class `AIFETransformerMaker` (inherits from `R6`) of length 3.

See Also

Other Transformer: AIFETrType, AIFETransformerMaker

Examples

```
# Use 'make' method of the 'aifeducation::aife_transformer_maker' object
# Pass string with the type of transformers
# Allowed types are "bert", "deberta_v2", "funnel", etc. See aifeducation::AIFETrType list
my_bert <- aife_transformer_maker$make("bert")

# Or use elements of the 'aifeducation::AIFETrType' list
my_longformer <- aife_transformer_maker$make(AIFETrType$longformer)

# Run 'create' or 'train' methods of the transformer in order to create a
# new transformer or train the newly created one, respectively
# my_bert$create(...)
# my_bert$train(...)

# my_longformer$create(...)
# my_longformer$train(...)
```

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: `clean_pytorch_log_transformers()`, `create_config_state()`, `create_dir()`, `generate_id()`, `get_file_extension()`, `get_py_package_versions()`, `is.null_or_na()`, `output_message()`, `print_message()`, `run_py_file()`

calc_standard_classification_measures *Calculate standard classification measures*

Description

Function for calculating recall, precision, and f1.

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 classifier_utils: `get_coder_metrics()`

`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, check = "pytorch")
```

Arguments

`trace`	`bool` `TRUE` if a list with all modules and their availability should be printed to the console.
`check`	`string` determining the machine learning framework to check for. • `check = "pytorch"`: for 'pytorch'. • `check = "tensorflow"`: for 'tensorflow'. • `check = "all"`: for both frameworks.

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: `install_aifeducation()`, `install_py_modules()`, `set_transformers_logger()`

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: auto_n_cores(), create_config_state(), create_dir(), generate_id(), get_file_extension(), get_py_package_versions(), is.null_or_na(), output_message(), print_message(), run_py_file()

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: fleiss_kappa(), 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: auto_n_cores(), clean_pytorch_log_transformers(), create_config_state(), generate_id(), get_file_extension(), get_py_package_versions(), is.null_or_na(), output_message(), print_message(), run_py_file()

`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 "smote","dbsmote", and "adas" from the package smotefamily are 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 data_management_utils: `get_n_chunks()`, `get_synthetic_cases_from_matrix()`

`DataManagerClassifier` *Data manager for classification tasks*

Description

Abstract class for managing the data and samples during training a classifier. DataManagerClassifier is used with TEClassifierRegular and TEClassifierProtoNet.

Value

Objects of this class are used for ensuring the correct data management for training different types of classifiers. Objects of this class 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:

- `DataManagerClassifier$new()`
- `DataManagerClassifier$get_config()`
- `DataManagerClassifier$get_labeled_data()`
- `DataManagerClassifier$get_unlabeled_data()`

- DataManagerClassifier$get_samples()
- DataManagerClassifier$set_state()
- DataManagerClassifier$get_n_folds()
- DataManagerClassifier$get_n_classes()
- DataManagerClassifier$get_statistics()
- DataManagerClassifier$get_dataset()
- DataManagerClassifier$get_val_dataset()
- DataManagerClassifier$get_test_dataset()
- DataManagerClassifier$create_synthetic()
- DataManagerClassifier$add_replace_pseudo_data()
- DataManagerClassifier$clone()

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

Usage:

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

Arguments:

`data_embeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings from which the DataManagerClassifier should be created.

`data_targets` `factor` containing the labels for cases stored in `data_embeddings`. Factor must be named and has to use the same names used in `data_embeddings`. Missing values are supported and should be supplied (e.g., for pseudo labeling).

`folds` `int` determining the number of cross-fold samples. Value must be at least 2.

`val_size` `double` between 0 and 1, indicating the proportion of cases of each class which should be used for the validation sample. The remaining cases are part of the training data.

`class_levels` `vector` containing the possible levels of the labels.

`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` determining the technique used for creating synthetic cases.

`sc_min_k` `int` determining the minimal number of neighbors during the creating of synthetic cases.

`sc_max_k` `int` determining the minimal number of neighbors during the creating of synthetic cases.

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

`n_cores` `int` Number of cores which should be used during the calculation of synthetic cases.

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 `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 Data Management: `EmbeddedText`, `LargeDataSetForText`, `LargeDataSetForTextEmbeddings`

EmbeddedText *Embedded text*

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$get_original_features()`
- `EmbeddedText$is_compressed()`
- `EmbeddedText$add_feature_extractor_info()`
- `EmbeddedText$get_feature_extractor_info()`
- `EmbeddedText$convert_to_LargeDataSetForTextEmbeddings()`
- `EmbeddedText$n_rows()`

- EmbeddedText$get_all_fields()
- EmbeddedText$clone()

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

Usage:

```
EmbeddedText$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_aggregation = NULL,
  embeddings
)
```

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

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

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 `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 `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: `DataManagerClassifier`, `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

Retuns 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: cohens_kappa(), kendalls_w(), kripp_alpha()

---

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 = 16)
```

Arguments

length	int determining the length of the id suffix.

Value

Returns a string of the requested length.

See Also

Other Utils: auto_n_cores(), clean_pytorch_log_transformers(), create_config_state(), create_dir(), get_file_extension(), get_py_package_versions(), is.null_or_na(), output_message(), print_message(), run_py_file()

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 Auxiliary Functions: `matrix_to_array_c()`, `summarize_tracked_sustainability()`, `to_categorical_c()`

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_ac**: Gwet's AC1/AC2 agreement coefficient.

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

See Also

Other classifier_utils: `calc_standard_classification_measures()`

`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: auto_n_cores(), clean_pytorch_log_transformers(), create_config_state(), create_dir(), generate_id(), get_py_package_versions(), is.null_or_na(), output_message(), print_message(), run_py_file()

---

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)
```

Arguments

text_embeddings	data.frame or array containing the text embeddings.
features	int Number of features within each sequence.
times	int Number of sequences.

Value

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

See Also

Other data_management_utils: create_synthetic_units_from_matrix(), get_synthetic_cases_from_matrix()

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: auto_n_cores(), clean_pytorch_log_transformers(), create_config_state(), create_dir(), generate_id(), get_file_extension(), is.null_or_na(), output_message(), print_message(), run_py_file()

get_synthetic_cases_from_matrix

Create synthetic cases for balancing training data

Description

This function creates synthetic cases for balancing the training with an object of the class TEClassifierRegular or TEClassifierProtoNet.

Usage

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

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 "smote", "dbsmote", and "adas" from the package smotefamily are 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 data_management_utils: create_synthetic_units_from_matrix(), get_n_chunks()

---

install_aifeducation *Install aifeducation on a machine*

Description

Function for installing 'aifeducation' on a machine. This functions assumes that not 'python' and no 'miniconda' is installed. Only'pytorch' is installed.

Usage

```
install_aifeducation(install_aifeducation_studio = TRUE)
```

Arguments

install_aifeducation_studio	bool If TRUE all necessary R packages are installed for using AI for Education Studio.

Value

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

See Also

Other Installation and Configuration: `check_aif_py_modules()`, `install_py_modules()`, `set_transformers_logger()`

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

Description

Function for installing the necessary python modules.

Usage

```
install_py_modules(
  envname = "aifeducation",
  install = "pytorch",
  transformer_version = "<=4.46",
  tokenizers_version = "<=0.20.4",
  pandas_version = "<=2.2.3",
  datasets_version = "<=3.1.0",
  codecarbon_version = "<=2.8.2",
  safetensors_version = "<=0.4.5",
  torcheval_version = "<=0.0.7",
  accelerate_version = "<=1.1.1",
  pytorch_cuda_version = "12.1",
  python_version = "3.9",
  remove_first = FALSE
)
```

Arguments

`envname`	`string` Name of the environment where the packages should be installed.
`install`	`character` determining which machine learning frameworks should be installed. • `install = "all"`: for 'pytorch' and 'tensorflow'. • `install = "pytorch"`: for 'pytorch'. • `install = "tensorflow"`: for 'tensorflow'.
`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'.

pytorch_cuda_version
: string determining the desired version of 'cuda' for ' PyTorch'.

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.

Value

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

See Also

Other Installation and Configuration: check_aif_py_modules(), install_aifeducation(), set_transformers_logger(

---

is.null_or_na *Check if NULL or NA*

Description

Function for checking if an object is NULL or .

Usage

```
is.null_or_na(object)
```

Arguments

object
: An object to test.

Value

Returns FALSE if the object is not NULL and not NA. Returns TRUE in all other cases.

See Also

Other Utils: auto_n_cores(), clean_pytorch_log_transformers(), create_config_state(), create_dir(), generate_id(), get_file_extension(), get_py_package_versions(), output_message(), print_message(), run_py_file()

`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: cohens_kappa(), fleiss_kappa(), kripp_alpha()

`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.

References

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

See Also

Other performance measures: cohens_kappa(), fleiss_kappa(), 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. LargeDataSetForTextEmbeddings or LargeDataSetForText.

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$get_all_fields()
- LargeDataSetBase$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 `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.

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 = 500,
  log_file = NULL,
  log_write_interval = 2,
  log_top_value = 0,
  log_top_total = 1,
  log_top_message = NA,
  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.

`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 = 500,
  log_file = NULL,
  log_write_interval = 2,
```

```
  log_top_value = 0,
  log_top_total = 1,
  log_top_message = NA,
  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.

`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 = 2,
  log_top_value = 0,
  log_top_total = 1,
  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: `DataManagerClassifier`, `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 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.

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$add_embeddings_from_array()`
- `LargeDataSetForTextEmbeddings$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_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_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 `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: `DataManagerClassifier`, `EmbeddedText`, `LargeDataSetForText`

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()

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 studio_utils: `create_data_embeddings_description()`

`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 Auxiliary Functions: `get_alpha_3_codes()`, `summarize_tracked_sustainability()`, `to_categorical_c()`

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: `auto_n_cores()`, `clean_pytorch_log_transformers()`, `create_config_state()`, `create_dir()`, `generate_id()`, `get_file_extension()`, `get_py_package_versions()`, `is.null_or_na()`, `print_message()`, `run_py_file()`

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: auto_n_cores(), clean_pytorch_log_transformers(), create_config_state(), create_dir(), generate_id(), get_file_extension(), get_py_package_versions(), is.null_or_na(), output_message(), run_py_file()

`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: auto_n_cores(), clean_pytorch_log_transformers(), create_config_state(), create_dir(), generate_id(), get_file_extension(), get_py_package_versions(), is.null_or_na(), output_message(), print_message()

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_config_cpu_only *Setting cpu only for 'tensorflow'*

Description

This functions configurates 'tensorflow' to use only cpus.

Usage

```
set_config_cpu_only()
```

Value

This function does not return anything. It is used for its side effects.

Note

os$environ$setdefault("CUDA_VISIBLE_DEVICES","-1")

See Also

Other Installation and Configuration Tensorflow: `set_config_gpu_low_memory()`, `set_config_os_environ_logger()`, `set_config_tf_logger()`

`set_config_gpu_low_memory`

Setting gpus' memory usage

Description

This function changes the memory usage of the gpus to allow computations on machines with small memory. With this function, some computations of large models may be possible but the speed of computation decreases.

Usage

```
set_config_gpu_low_memory()
```

Value

This function does not return anything. It is used for its side effects.

Note

This function sets TF_GPU_ALLOCATOR to `"cuda_malloc_async"` and sets memory growth to `TRUE`.

See Also

Other Installation and Configuration Tensorflow: `set_config_cpu_only()`, `set_config_os_environ_logger()`, `set_config_tf_logger()`

`set_config_os_environ_logger`

Sets the level for logging information in tensorflow

Description

This function changes the level for logging information with 'tensorflow' via the os environment. This function must be called before importing 'tensorflow'.

Usage

```
set_config_os_environ_logger(level = "ERROR")
```

Arguments

level	string Minimal level that should be printed to console. Four levels are available: INFO, WARNING, ERROR and NONE.

Value

This function does not return anything. It is used for its side effects.

See Also

Other Installation and Configuration Tensorflow: set_config_cpu_only(), set_config_gpu_low_memory(), set_config_tf_logger()

---

set_config_tf_logger *Sets the level for logging information in tensorflow*

Description

This function changes the level for logging information with 'tensorflow'.

Usage

```
set_config_tf_logger(level = "ERROR")
```

Arguments

level	string Minimal level that should be printed to console. Five levels are available: FATAL, ERROR, WARN, INFO, and DEBUG.

Value

This function does not return anything. It is used for its side effects.

See Also

Other Installation and Configuration Tensorflow: set_config_cpu_only(), set_config_gpu_low_memory(), set_config_os_environ_logger()

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()`, `install_aifeducation()`, `install_py_modules()`

start_aifeducation_studio

Aifeducation Studio

Description

Functions starts a shiny app that represents Aifeducation Studio.

Usage

```
start_aifeducation_studio()
```

Value

This function does nothing return. It is used to start a shiny app.

`TEClassifierProtoNet` *Text embedding classifier with a ProtoNet*

Description

Abstract class for neural nets with 'keras'/'tensorflow' and 'pytorch'.

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::AIFEBaseModel` -> `aifeducation::TEClassifierRegular` -> `TEClassifierProtoNet`

Methods

Public methods:

- `TEClassifierProtoNet$configure()`
- `TEClassifierProtoNet$train()`
- `TEClassifierProtoNet$embed()`
- `TEClassifierProtoNet$plot_embeddings()`
- `TEClassifierProtoNet$clone()`

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

Usage:

```
TEClassifierProtoNet$configure(
  ml_framework = "pytorch",
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  dense_size = 4,
  dense_layers = 0,
```

```
  rec_size = 4,
  rec_layers = 2,
  rec_type = "gru",
  rec_bidirectional = FALSE,
  embedding_dim = 2,
  self_attention_heads = 0,
  intermediate_size = NULL,
  attention_type = "fourier",
  add_pos_embedding = TRUE,
  rec_dropout = 0.1,
  repeat_encoder = 1,
  dense_dropout = 0.4,
  recurrent_dropout = 0.4,
  encoder_dropout = 0.1,
  optimizer = "adam"
)
```

Arguments:

`ml_framework string` Currently only pytorch is supported (`ml_framework="pytorch"`).

`name string` Name of the new classifier. Please refer to common name conventions. Free text can be used with parameter `label`.

`label string` Label for the new classifier. Here you can use free text.

`text_embeddings` An object of class TextEmbeddingModel or LargeDataSetForTextEmbeddings.

`feature_extractor` 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 not 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.

`dense_layers int` Number of dense layers.

`rec_size int` Number of neurons for each recurrent layer.

`rec_layers int` Number of recurrent layers.

`rec_type string` Type of the recurrent layers.`rec_type="gru"` for Gated Recurrent Unit and `rec_type="lstm"` for Long Short-Term Memory.

`rec_bidirectional bool` If `TRUE` a bidirectional version of the recurrent layers is used.

`embedding_dim int` determining the number of dimensions for the text embedding.

`self_attention_heads int` determining the number of attention heads for a self-attention layer. Only relevant if `attention_type="multihead"`.

`intermediate_size int` determining the size of the projection layer within a each transformer encoder.

`attention_type string` Choose the relevant attention type. Possible values are `"fourier"` and `"multihead"`. Please note that you may see different values for a case for different input orders if you choose `fourier` on linux.

`add_pos_embedding bool` `TRUE` if positional embedding should be used.

`rec_dropout` `double` ranging between 0 and lower 1, determining the dropout between bidirectional recurrent layers.

`repeat_encoder` `int` determining how many times the encoder should be added to the network.

`dense_dropout` `double` ranging between 0 and lower 1, determining the dropout between dense layers.

`recurrent_dropout` `double` ranging between 0 and lower 1, determining the recurrent dropout for each recurrent layer. Only relevant for keras models.

`encoder_dropout` `double` ranging between 0 and lower 1, determining the dropout for the dense projection within the encoder layers.

`optimizer` `string` `"adam"` or `"rmsprop"` .

Returns: Returns an object of class TEClassifierProtoNet which is ready for training.

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:

```
TEClassifierProtoNet$train(
  data_embeddings,
  data_targets,
  data_folds = 5,
  data_val_size = 0.25,
  use_sc = TRUE,
  sc_method = "dbsmote",
  sc_min_k = 1,
  sc_max_k = 10,
  use_pl = TRUE,
  pl_max_steps = 3,
  pl_max = 1,
  pl_anchor = 1,
  pl_min = 0,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  epochs = 40,
  batch_size = 35,
  Ns = 5,
  Nq = 3,
  loss_alpha = 0.5,
  loss_margin = 0.5,
  sampling_separate = FALSE,
  sampling_shuffle = TRUE,
  dir_checkpoint,
```

```
  trace = TRUE,
  ml_trace = 1,
  log_dir = NULL,
  log_write_interval = 10,
  n_cores = auto_n_cores()
)
```

Arguments:

`data_embeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

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

`data_folds` `int` determining the number of cross-fold samples.

`data_val_size` `double` between 0 and 1, indicating the proportion of cases of each class which should be used for the validation sample during the estimation of the model. The remaining cases are part of the training data.

`use_sc` `bool` `TRUE` if the estimation should integrate synthetic cases. `FALSE` if not.

`sc_method` `vector` containing the method for generating synthetic cases. Possible are `sc_method="adas"`, `sc_method="smote"`, and `sc_method="dbsmote"`.

`sc_min_k` `int` determining the minimal number of k which is used for creating synthetic units.

`sc_max_k` `int` determining the maximal number of k which is used for creating synthetic units.

`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.

`pl_max` `double` between 0 and 1, setting the maximal level of confidence for considering a case for pseudo-labeling.

`pl_anchor` `double` between 0 and 1 indicating the reference point for sorting the new cases of every label. See notes for more details.

`pl_min` `double` between 0 and 1, setting the minimal level of confidence for considering a case for pseudo-labeling.

`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.

`sustain_region` 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

`sustain_interval` `int` Interval in seconds for measuring power usage.

`epochs` `int` Number of training epochs.

`batch_size` `int` Size of the batches for training.

`Ns` `int` Number of cases for every class in the sample.

`Nq` `int` Number of cases for every class in the query.

`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.

`loss_margin` `double` Value greater 0 indicating the minimal distance of every case from prototypes of other classes

`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.

`dir_checkpoint` `string` Path to the directory where the checkpoint during training should be saved. If the directory does not exist, it is created.

`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.

`log_dir` `string` Path to the directory where the log files 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_dir` is not `NULL`.

`n_cores` `int` Number of cores which should be used during the calculation of synthetic cases. Only relevant if `use_sc=TRUE`.

`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.

`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 `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 = 32)
```

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 = 12,
  alpha = 0.5,
  size_points = 3,
  size_points_prototypes = 8,
  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.

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: TEClassifierRegular

TEClassifierRegular *Text embedding classifier with a neural net*

Description

Abstract class for neural nets with 'keras'/'tensorflow' and ' pytorch'.

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 class

`aifeducation::AIFEBaseModel` -> `TEClassifierRegular`

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:

- `TEClassifierRegular$configure()`
- `TEClassifierRegular$train()`
- `TEClassifierRegular$predict()`
- `TEClassifierRegular$check_embedding_model()`
- `TEClassifierRegular$check_feature_extractor_object_type()`
- `TEClassifierRegular$requires_compression()`
- `TEClassifierRegular$save()`
- `TEClassifierRegular$load_from_disk()`
- `TEClassifierRegular$clone()`

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

Usage:

```
TEClassifierRegular$configure(
  ml_framework = "pytorch",
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  feature_extractor = NULL,
  target_levels = NULL,
  dense_size = 4,
  dense_layers = 0,
  rec_size = 4,
  rec_layers = 2,
  rec_type = "gru",
  rec_bidirectional = FALSE,
  self_attention_heads = 0,
  intermediate_size = NULL,
  attention_type = "fourier",
```

```
  add_pos_embedding = TRUE,
  rec_dropout = 0.1,
  repeat_encoder = 1,
  dense_dropout = 0.4,
  recurrent_dropout = 0.4,
  encoder_dropout = 0.1,
  optimizer = "adam"
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference. `ml_framework="tensorflow"` for 'tensorflow' and `ml_framework="pytorch"` for 'pytorch'

`name` `string` Name of the new classifier. Please refer to common name conventions. Free text can be used with parameter `label`.

`label` `string` Label for the new classifier. Here you can use free text.

`text_embeddings` An object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`feature_extractor` 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 not 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.

`dense_layers` `int` Number of dense layers.

`rec_size` `int` Number of neurons for each recurrent layer.

`rec_layers` `int` Number of recurrent layers.

`rec_type` `string` Type of the recurrent layers. `rec_type="gru"` for Gated Recurrent Unit and `rec_type="lstm"` for Long Short-Term Memory.

`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"`

`intermediate_size` `int` determining the size of the projection layer within a each transformer encoder.

`attention_type` `string` Choose the relevant attention type. Possible values are `fourier` and `multihead`. Please note that you may see different values for a case for different input orders if you choose `fourier` on linux.

`add_pos_embedding` `bool` `TRUE` if positional embedding should be used.

`rec_dropout` `int` ranging between 0 and lower 1, determining the dropout between bidirectional recurrent layers.

`repeat_encoder` `int` determining how many times the encoder should be added to the network.

`dense_dropout` `int` ranging between 0 and lower 1, determining the dropout between dense layers.

`recurrent_dropout` `int` ranging between 0 and lower 1, determining the recurrent dropout for each recurrent layer. Only relevant for keras models.

`encoder_dropout` `int` ranging between 0 and lower 1, determining the dropout for the dense projection within the encoder layers.

`optimizer` `string` `"adam"` or `"rmsprop"` .

Returns: Returns an object of class TEClassifierRegular which is ready for training.

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:

```
TEClassifierRegular$train(
  data_embeddings,
  data_targets,
  data_folds = 5,
  data_val_size = 0.25,
  balance_class_weights = TRUE,
  balance_sequence_length = TRUE,
  use_sc = TRUE,
  sc_method = "dbsmote",
  sc_min_k = 1,
  sc_max_k = 10,
  use_pl = TRUE,
  pl_max_steps = 3,
  pl_max = 1,
  pl_anchor = 1,
  pl_min = 0,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  epochs = 40,
  batch_size = 32,
  dir_checkpoint,
  trace = TRUE,
  ml_trace = 1,
  log_dir = NULL,
  log_write_interval = 10,
  n_cores = auto_n_cores()
)
```

Arguments:

`data_embeddings` Object of class EmbeddedText or LargeDataSetForTextEmbeddings.

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

`data_folds` `int` determining the number of cross-fold samples.

`data_val_size` `double` between 0 and 1, indicating the proportion of cases of each class which should be used for the validation sample during the estimation of the model. The remaining cases are part of the training data.

`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.

`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.

`use_sc` `bool` `TRUE` if the estimation should integrate synthetic cases. `FALSE` if not.

`sc_method` `vector` containing the method for generating synthetic cases. Possible are `sc_method="adas"`, `sc_method="smote"`, and `sc_method="dbsmote"`.

`sc_min_k` `int` determining the minimal number of k which is used for creating synthetic units.

`sc_max_k` `int` determining the maximal number of k which is used for creating synthetic units.

`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.

`pl_max` `double` between 0 and 1, setting the maximal level of confidence for considering a case for pseudo-labeling.

`pl_anchor` `double` between 0 and 1 indicating the reference point for sorting the new cases of every label. See notes for more details.

`pl_min` `double` between 0 and 1, setting the minimal level of confidence for considering a case for pseudo-labeling.

`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.

`sustain_region` 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

`sustain_interval` `int` Interval in seconds for measuring power usage.

`epochs` `int` Number of training epochs.

`batch_size` `int` Size of the batches for training.

`dir_checkpoint` `string` Path to the directory where the checkpoint during training should be saved. If the directory does not exist, it is created.

`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.

`log_dir` `string` Path to the directory where the log files 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_dir` is not `NULL`.

`n_cores` `int` Number of cores which should be used during the calculation of synthetic cases. Only relevant if `use_sc=TRUE`.

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

Usage:

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

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:

```
TEClassifierRegular$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:

```
TEClassifierRegular$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:

```
TEClassifierRegular$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:

```
TEClassifierRegular$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:

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Classification: `TEClassifierProtoNet`

`TEFeatureExtractor` *Feature extractor for reducing the number for dimensions of text embeddings.*

Description

Abstract class for auto encoders with 'pytorch'.

Value

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.

Super class

`aifeducation::AIFEBaseModel` -> `TEFeatureExtractor`

Methods

Public methods:

- `TEFeatureExtractor$configure()`
- `TEFeatureExtractor$train()`
- `TEFeatureExtractor$load_from_disk()`
- `TEFeatureExtractor$extract_features()`
- `TEFeatureExtractor$extract_features_large()`
- `TEFeatureExtractor$is_trained()`
- `TEFeatureExtractor$clone()`

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

Usage:

```
TEFeatureExtractor$configure(
  ml_framework = "pytorch",
  name = NULL,
  label = NULL,
  text_embeddings = NULL,
  features = 128,
  method = "lstm",
  noise_factor = 0.2,
  optimizer = "adam"
)
```

Arguments:

`ml_framework` `string` Framework to use for training and inference. Currently only `ml_framework="pytorch"` is supported.

`name` `string` Name of the new classifier. Please refer to common name conventions. Free text can be used with parameter `label`.

`label` `string` Label for the new classifier. Here you can use free text.

`text_embeddings` An object of class EmbeddedText or LargeDataSetForTextEmbeddings.

`features` `int` determining the number of dimensions to which the dimension of the text embedding should be reduced.

`method` `string` Method to use for the feature extraction. `"lstm"` for an extractor based on LSTM-layers or `"dense"` for dense layers.

`noise_factor` `double` between 0 and a value lower 1 indicating how much noise should be added for the training of the feature extractor.

`optimizer` `string` `"adam"` or `"rmsprop"` .

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,
  data_val_size = 0.25,
  sustain_track = TRUE,
  sustain_iso_code = NULL,
  sustain_region = NULL,
  sustain_interval = 15,
  epochs = 40,
  batch_size = 32,
  dir_checkpoint,
  trace = TRUE,
  ml_trace = 1,
  log_dir = NULL,
  log_write_interval = 10
)
```

Arguments:

`data_embeddings` 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.

`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.

`sustain_region` 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

`sustain_interval` `int` Interval in seconds for measuring power usage.

`epochs` `int` Number of training epochs.

`batch_size` `int` Size of batches.

`dir_checkpoint` `string` Path to the directory where the checkpoint during training should be saved. If the directory does not exist, it is created.

`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. `ml_trace=1` prints a progress bar.

`log_dir` `string` Path to the directory where the log files 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_dir` is not `NULL`.

Returns: Function does not return a value. It changes the object into a trained classifier.

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

Usage:

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

Usage:

```
TEFeatureExtractor$is_trained()
```

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

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.

See Also

Other Text Embedding: `TextEmbeddingModel`

`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.

Public fields

`last_training` ('list()')
: List for storing the history and the results of the last training. This information will be overwritten if a new training is started.

`tokenizer_statistics` ('matrix()')
: Matrix containing the tokenizer statistics for the creation of the tokenizer and all training runs according to Kaya & 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

Methods

Public methods:

- `TextEmbeddingModel$configure()`
- `TextEmbeddingModel$load_from_disk()`
- `TextEmbeddingModel$load()`
- `TextEmbeddingModel$save()`
- `TextEmbeddingModel$encode()`
- `TextEmbeddingModel$decode()`
- `TextEmbeddingModel$get_special_tokens()`
- `TextEmbeddingModel$embed()`
- `TextEmbeddingModel$embed_large()`
- `TextEmbeddingModel$fill_mask()`
- `TextEmbeddingModel$set_publication_info()`
- `TextEmbeddingModel$get_publication_info()`
- `TextEmbeddingModel$set_model_license()`
- `TextEmbeddingModel$get_model_license()`
- `TextEmbeddingModel$set_documentation_license()`
- `TextEmbeddingModel$get_documentation_license()`
- `TextEmbeddingModel$set_model_description()`
- `TextEmbeddingModel$get_model_description()`
- `TextEmbeddingModel$get_model_info()`
- `TextEmbeddingModel$get_package_versions()`
- `TextEmbeddingModel$get_basic_components()`
- `TextEmbeddingModel$get_transformer_components()`
- `TextEmbeddingModel$get_sustainability_data()`
- `TextEmbeddingModel$get_ml_framework()`
- `TextEmbeddingModel$count_parameter()`
- `TextEmbeddingModel$is_configured()`
- `TextEmbeddingModel$get_private()`
- `TextEmbeddingModel$get_all_fields()`
- `TextEmbeddingModel$clone()`

Method `configure()`**:** Method for creating a new text embedding model

Usage:

```
TextEmbeddingModel$configure(
  model_name = NULL,
  model_label = NULL,
  model_language = NULL,
  method = NULL,
  ml_framework = "pytorch",
  max_length = 0,
  chunks = 2,
  overlap = 0,
  emb_layer_min = "middle",
  emb_layer_max = "2_3_layer",
  emb_pool_type = "average",
  model_dir = NULL,
  trace = FALSE
)
```

Arguments:

`model_name` `string` containing the name of the new model.

`model_label` `string` containing the label/title of the new model.

`model_language` `string` containing the language which the model represents (e.g., English).

`method` `string` determining the kind of embedding model. Currently the following models are supported: `method="bert"` for Bidirectional Encoder Representations from Transformers (BERT), `method="roberta"` for A Robustly Optimized BERT Pretraining Approach (RoBERTa), `method="longformer"` for Long-Document Transformer, `method="funnel"` for Funnel-Transformer, `method="deberta_v2"` for Decoding-enhanced BERT with Disentangled Attention (DeBERTa V2), `method="glove"`` for `GlobalVector Clusters`, and method="lda"‘ for topic modeling. See details for more information.

`ml_framework` `string` Framework to use for the model. `ml_framework="tensorflow"` for 'tensorflow' and `ml_framework="pytorch"` for 'pytorch'. Only relevant for transformer models. To request bag-of-words model set `ml_framework=NULL`.

`max_length` `int` determining the maximum length of token sequences used in transformer models. Not relevant for the other methods.

`chunks` `int` Maximum number of chunks. Must be at least 2.

`overlap` `int` determining the number of tokens which should be added at the beginning of the next chunk. Only relevant for transformer models.

`emb_layer_min` `int` or `string` determining the first layer to be included in the creation of embeddings. An integer correspondents to the layer number. The first layer has the number 1. Instead of an integer the following strings are possible: `"start"` for the first layer, `"middle"` for the middle layer, `"2_3_layer"` for the layer two-third layer, and `"last"` for the last layer.

`emb_layer_max` `int` or `string` determining the last layer to be included in the creation of embeddings. An integer correspondents to the layer number. The first layer has the number 1. Instead of an integer the following strings are possible: `"start"` for the first layer, `"middle"` for the middle layer, `"2_3_layer"` for the layer two-third layer, and `"last"` for the last layer.

`emb_pool_type` `string` determining the method for pooling the token embeddings within each layer. If `"cls"` only the embedding of the CLS token is used. If `"average"` the token

embedding of all tokens are averaged (excluding padding tokens). "cls is not supported for method="funnel".

model_dir string path to the directory where the BERT model is stored.

trace bool TRUE prints information about the progress. FALSE does not.

Details: In the case of any transformer (e.g.method="bert", method="roberta", and method="longformer"), a pretrained transformer model must be supplied via model_dir.

Returns: Returns an object of class TextEmbeddingModel.

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

Method load(): Method for loading a transformers model into R.

Usage:

```
TextEmbeddingModel$load(dir_path)
```

Arguments:

dir_path string containing the path to the relevant model directory.

Returns: Function does not return a value. It is used for loading a saved transformer model into the R interface.

Method save(): Method for saving a transformer model on disk.Relevant only for transformer models.

Usage:

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

Arguments:

dir_path string containing the path to the relevant model directory.

folder_name string Name for the folder created within the directory. This folder contains all model files.

Returns: Function does not return a value. It is used for saving a transformer model to disk.

Method encode(): Method for encoding words of raw texts into integers.

Usage:

```
TextEmbeddingModel$encode(
  raw_text,
  token_encodings_only = FALSE,
  to_int = TRUE,
  trace = FALSE
)
```

Arguments:

`raw_text` `vector`containing the raw texts.

`token_encodings_only` `bool` If `TRUE`, only the token encodings are returned. If `FALSE`, the complete encoding is returned which is important for some transformer models.

`to_int` `bool` If `TRUE` the integer ids of the tokens are returned. If `FALSE` the tokens are returned. Argument only applies for transformer models and if `token_encodings_only=TRUE`.

`trace` `bool` If `TRUE`, information of the progress is printed. `FALSE` if not requested.

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` containing the integer sequences which should be transformed to tokens or plain text.

`to_token` `bool` If `FALSE` plain text is returned. If `TRUE` a sequence of tokens is returned. Argument only relevant if the model is based on a transformer.

Returns: `list` of token sequences

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

Usage:

```
TextEmbeddingModel$get_special_tokens()
```

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

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`. In the case of using a GPU and running out of memory while using 'tensorflow' reduce the batch size or restart R and switch to use cpu only via `set_config_cpu_only`. In general, not relevant for 'pytorch'.

Usage:

```
TextEmbeddingModel$embed(
  raw_text = NULL,
  doc_id = NULL,
  batch_size = 8,
  trace = FALSE,
  return_large_dataset = FALSE
)
```

Arguments:

`raw_text` `vector` containing the raw texts.

`doc_id` `vector` containing the corresponding IDs for every text.

`batch_size` `int` determining the maximal size of every batch.

`trace` `bool` `TRUE`, if information about the progression should be printed on console.

`return_large_dataset` 'bool' If `TRUE` the retuned object is of class LargeDataSetForTextEmbeddings. If `FALSE` it is of class EmbeddedText

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(
  large_datas_set,
  batch_size = 32,
  trace = FALSE,
  log_file = NULL,
  log_write_interval = 2
)
```

Arguments:

`large_datas_set` Object of class LargeDataSetForText containing the raw texts.

`batch_size` `int` determining the maximal size of every batch.

`trace` `bool` `TRUE`, if information about the progression should be printed on console.

`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`.

Returns: Method returns an object of class LargeDataSetForTextEmbeddings.

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

Usage:

```
TextEmbeddingModel$fill_mask(text, n_solutions = 5)
```

Arguments:

`text` `string` Text containing mask tokens.

`n_solutions` `int` Number estimated tokens for every mask.

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 `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_publication_info()`**:** Method for getting the bibliographic information of the model.

Usage:

```
TextEmbeddingModel$get_publication_info()
```

Returns: `list` of bibliographic information.

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

Usage:

```
TextEmbeddingModel$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 requesting the license of the model

Usage:

```
TextEmbeddingModel$get_model_license()
```

Returns: `string` License of the model

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

Usage:

```
TextEmbeddingModel$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 to set the private member for the documentation license of the model.

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

Usage:

```
TextEmbeddingModel$get_documentation_license()
```

Arguments:

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

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

Usage:

```
TextEmbeddingModel$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 of the classifier, its theoretical and empirical background, and the different output labels in English.

`native string` A text describing the training of the classifier, its theoretical and empirical background, and the different output labels 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 classifier.

`keywords_eng vector`of keywords in English.

`keywords_native vector`of keywords in the native language of the classifier.

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

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

Usage:

```
TextEmbeddingModel$get_model_description()
```

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

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

Usage:

```
TextEmbeddingModel$get_model_info()
```

Returns: `list` of all relevant model information

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

Usage:

```
TextEmbeddingModel$get_package_versions()
```

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

Method `get_basic_components()`**:** Method for requesting the part of interface's configuration that is necessary for all models.

Usage:

```
TextEmbeddingModel$get_basic_components()
```

Returns: Returns a `list`.

Method `get_transformer_components()`**:** Method for requesting the part of interface's configuration that is necessary for transformer models.

Usage:

```
TextEmbeddingModel$get_transformer_components()
```

Returns: Returns a `list`.

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

Usage:

```
TextEmbeddingModel$get_sustainability_data()
```

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

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

Usage:

```
TextEmbeddingModel$get_ml_framework()
```

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

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

Usage:

```
TextEmbeddingModel$count_parameter(with_head = FALSE)
```

Arguments:

`with_head` `bool` If `TRUE` the number of parameters is returned including the language modeling head of the model. If `FALSE` only the number of parameters of the core model is returned.

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

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

Usage:

```
TextEmbeddingModel$is_configured()
```

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

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

Usage:

```
TextEmbeddingModel$get_private()
```

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

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

Usage:

```
TextEmbeddingModel$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:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Text Embedding: `TEFeatureExtractor`

`to_categorical_c` *Transforming classes to one-hot encoding*

Description

Function written in C++ 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 Auxiliary Functions: `get_alpha_3_codes()`, `matrix_to_array_c()`, `summarize_tracked_sustainability()`

Index