Package: tidybert (via r-universe)

November 20, 2024

Title Tidy BERT-like Models

Version 0.0.0.9900

Description Implements BERT-like NLP models with a consistent interface for fitting and creating predictions. The models are fully compatible with the tidymodels framework.

License Apache License (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports dplyr, glue, hardhat, luz, magrittr, methods, purrr, rlang, stats, tibble, torch, torchtransformers (>= 0.0.0.9500)

Remotes macmillancontentscience/torchtransformers

Suggests knitr, rmarkdown, testthat (>= 3.0.0), dials, dlr, parsnip, recipes, rsample, stringr, tidymodels, tidyr, wordpiece, wordpiece.data

URL `https://github.com/macmillancontentscience/tidybert`

BugReports `https://github.com/macmillancontentscience/tidybert/issues`

Config/testthat/edition 3

VignetteBuilder knitr

Config/pak/sysreqs make libicu-dev

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

RemoteUrl https://github.com/macmillancontentscience/tidybert

RemoteRef HEAD

RemoteSha e25e1912249af094d275cff414598f3ad0cfb6af

Contents

`bert` *Fine-Tune a BERT Model*

Description

`bert()` defines a model that fine-tunes a pre-trained BERT-like model to a classification or regression task.

Usage

```
bert(
  mode = "unknown",
  engine = "tidybert",
  epochs = 10,
  batch_size = 128,
  bert_type = "bert_small_uncased",
  n_tokens = 1
)
```

Arguments

`mode`	A single character string for the prediction outcome mode. Possible values for this model are "unknown", "regression", or "classification".
`engine`	A single character string specifying what computational engine to use for fitting. The only implemented option is "tidybert".
`epochs`	A single integer indicating the maximum number of epochs for training, or a vector of two integers, indicating the minimum and maximum number of epochs for training.
`batch_size`	The number of samples to load in each batch during training.
`bert_type`	Character; which flavor of BERT to use. See `available_berts()` for known models.
`n_tokens`	An integer scalar indicating the number of tokens in the output.

Details

This package (`tidybert`) is currently the only engine for this model. See tidybert_engine for parameters available in this engine.

The defined model is appropriate for use with `parsnip` and the rest of the `tidymodels` framework.

Value

A specification for a model.

---

bert_classification *Fit a BERT-style neural network*

Description

`bert_classification()` fits a classifier neural network in the style of BERT from Google Research.

Usage

```
bert_classification(x, ...)

## Default S3 method:
bert_classification(x, ...)

## S3 method for class 'data.frame'
bert_classification(
  x,
  y,
  valid_x = 0.1,
  valid_y = NULL,
  bert_type = "bert_tiny_uncased",
  n_tokens = torchtransformers::config_bert(bert_type, "max_tokens"),
  loss = torch::nn_cross_entropy_loss(),
  optimizer = torch::optim_adam,
  metrics = list(luz::luz_metric_accuracy()),
  epochs = 10,
  batch_size = 128,
  luz_opt_hparams = list(),
  ...
)

## S3 method for class 'matrix'
bert_classification(
  x,
  y,
  valid_x = 0.1,
  valid_y = NULL,
  bert_type = "bert_tiny_uncased",
  n_tokens = torchtransformers::config_bert(bert_type, "max_tokens"),
  loss = torch::nn_cross_entropy_loss(),
  optimizer = torch::optim_adam,
  metrics = list(luz::luz_metric_accuracy()),
  epochs = 10,
  batch_size = 128,
  luz_opt_hparams = list(),
  ...
)

## S3 method for class 'formula'
bert_classification(
  formula,
  data,
  valid_data = 0.1,
  bert_type = "bert_tiny_uncased",
  n_tokens = torchtransformers::config_bert(bert_type, "max_tokens"),
  loss = torch::nn_cross_entropy_loss(),
  optimizer = torch::optim_adam,
  metrics = list(luz::luz_metric_accuracy()),
  epochs = 10,
  batch_size = 128,
  luz_opt_hparams = list(),
  ...
)
```

Arguments

`x`	Depending on the context: • A **data frame** of character predictors. • A **matrix** of character predictors. • Note that a **recipe** created from `recipes::recipe()` will NOT currently work.
`...`	Additional parameters to pass to methods or to luz for fitting.
`y`	When `x` is a **data frame** or **matrix**, `y` is the outcome specified as: • A **data frame** with 1 factor column. • A **matrix** with 1 factor column. • A factor **vector**.
`valid_x`	Depending on the context: • A number between 0 and 1, representing the fraction of data to use for model validation. • Predictors in the same format as `x`. These predictors will be used for model validation. • `NULL`, in which case no data will be used for model validation.
`valid_y`	When `valid_x` is a set of predictors, `valid_y` should be outcomes in the same format as `y`.
`bert_type`	Character; which flavor of BERT to use. See `available_berts()` for known models.
`n_tokens`	An integer scalar indicating the number of tokens in the output.

`loss`	(`function`, optional) An optional function with the signature `function(input, target)`. It's only requires if your `nn_module` doesn't implement a method called `loss`.
`optimizer`	(`torch_optimizer`, optional) A function with the signature `function(parameters, ...)` that is used to initialize an optimizer given the model parameters.
`metrics`	(`list`, optional) A list of metrics to be tracked during the training procedure. Sometimes, you want some metrics to be evaluated only during training or validation, in this case you can pass a `luz_metric_set()` object to specify metrics used in each stage.
`epochs`	(int) The maximum number of epochs for training the model. If a single value is provided, this is taken to be the `max_epochs` and `min_epochs` is set to 0. If a vector of two numbers is provided, the first value is `min_epochs` and the second value is `max_epochs`. The minimum and maximum number of epochs are included in the context object as `ctx$min_epochs` and `ctx$max_epochs`, respectively.
`batch_size`	(int, optional): how many samples per batch to load (default: `1`).
`luz_opt_hparams`	List; parameters to pass on to `set_opt_hparams` to initialize the optimizer.
`formula`	A formula specifying the outcome term on the left-hand side, and the predictor terms on the right-hand side.
`data`	When a **formula** is used, `data` is specified as: • A **data frame** containing both the predictors and the outcome. The predictors should be character vectors. The outcome should be a factor.
`valid_data`	When a **formula** is used, `valid_data` can be: • A **data frame** containing both the predictors and the outcome to be used for model validation, in the same format as `data`. • A number between 0 and 1, representing the fraction of data to use for model validation. • `NULL`, in which case no data will be used for model validation.

Details

The generated model is a pretrained BERT model with a final dense linear layer to map the output to the outcome levels, constructed using `model_bert_linear()`. That pretrained model is fine-tuned on the provided training data. Input data (during both fitting and prediction) is automatically tokenized to match the tokenization expected by the BERT model.

Value

A `bert_classification` object.

`bert_regression` *Fit a BERT-style neural network*

Description

`bert_regression()` fits a regression neural network in the style of BERT from Google Research.

Usage

```
bert_regression(x, ...)

## Default S3 method:
bert_regression(x, ...)

## S3 method for class 'data.frame'
bert_regression(
  x,
  y,
  valid_x = 0.1,
  valid_y = NULL,
  bert_type = "bert_tiny_uncased",
  n_tokens = torchtransformers::config_bert(bert_type, "max_tokens"),
  loss = torch::nn_mse_loss(),
  optimizer = torch::optim_adam,
  metrics = list(luz::luz_metric_rmse()),
  epochs = 10,
  batch_size = 128,
  luz_opt_hparams = list(),
  ...
)

## S3 method for class 'matrix'
bert_regression(
  x,
  y,
  valid_x = 0.1,
  valid_y = NULL,
  bert_type = "bert_tiny_uncased",
  n_tokens = torchtransformers::config_bert(bert_type, "max_tokens"),
  loss = torch::nn_mse_loss(),
  optimizer = torch::optim_adam,
  metrics = list(luz::luz_metric_rmse()),
  epochs = 10,
  batch_size = 128,
  luz_opt_hparams = list(),
  ...
)
```

```
## S3 method for class 'formula'
bert_regression(
  formula,
  data,
  valid_data = 0.1,
  bert_type = "bert_tiny_uncased",
  n_tokens = torchtransformers::config_bert(bert_type, "max_tokens"),
  loss = torch::nn_mse_loss(),
  optimizer = torch::optim_adam,
  metrics = list(luz::luz_metric_rmse()),
  epochs = 10,
  batch_size = 128,
  luz_opt_hparams = list(),
  ...
)
```

Arguments

`x`	Depending on the context: • A **data frame** of character predictors. • A **matrix** of character predictors. • Note that a **recipe** created from `recipes::recipe()` will NOT currently work.
`...`	Additional parameters to pass to methods or to luz for fitting.
`y`	When `x` is a **data frame** or **matrix**, `y` is the outcome specified as: • A **data frame** with 1 numerical column. • A **matrix** with 1 numerical column. • A numerical **vector**.
`valid_x`	Depending on the context: • A number between 0 and 1, representing the fraction of data to use for model validation. • Predictors in the same format as `x`. These predictors will be used for model validation. • `NULL`, in which case no data will be used for model validation.
`valid_y`	When `valid_x` is a set of predictors, `valid_y` should be outcomes in the same format as `y`.
`bert_type`	Character; which flavor of BERT to use. See `available_berts()` for known models.
`n_tokens`	An integer scalar indicating the number of tokens in the output.
`loss`	(`function`, optional) An optional function with the signature `function(input, target)`. It's only requires if your `nn_module` doesn't implement a method called `loss`.
`optimizer`	(`torch_optimizer`, optional) A function with the signature `function(parameters, ...)` that is used to initialize an optimizer given the model parameters.

`metrics`	(`list`, optional) A list of metrics to be tracked during the training procedure. Sometimes, you want some metrics to be evaluated only during training or validation, in this case you can pass a `luz_metric_set()` object to specify metrics used in each stage.
`epochs`	(int) The maximum number of epochs for training the model. If a single value is provided, this is taken to be the `max_epochs` and `min_epochs` is set to 0. If a vector of two numbers is provided, the first value is `min_epochs` and the second value is `max_epochs`. The minimum and maximum number of epochs are included in the context object as `ctx$min_epochs` and `ctx$max_epochs`, respectively.
`batch_size`	(int, optional): how many samples per batch to load (default: `1`).
`luz_opt_hparams`	List; parameters to pass on to `set_opt_hparams` to initialize the optimizer.
`formula`	A formula specifying the outcome term on the left-hand side, and the predictor terms on the right-hand side.
`data`	When a **formula** is used, `data` is specified as: • A **data frame** containing both the predictors and the outcome. The predictors should be character vectors. The outcome should be numerical.
`valid_data`	When a **formula** is used, `valid_data` can be: • A **data frame** containing both the predictors and the outcome to be used for model validation, in the same format as `data`. • A number between 0 and 1, representing the fraction of data to use for model validation. • `NULL`, in which case no data will be used for model validation.

Details

The generated model is a pretrained BERT model with a final dense linear layer to map the output to a numerical value, constructed using `model_bert_linear()`. That pretrained model is fine-tuned on the provided training data. Input data (during both fitting and prediction) is automatically tokenized to match the tokenization expected by the BERT model.

Value

A `bert_regression` object.

`bert_type` *Pre-Trained BERT Model*

Description

The pre-trained BERT model that will be fine-tuned for a model.

Usage

```
bert_type(
  values = c("bert_tiny_uncased", "bert_mini_uncased", "bert_small_uncased",
   "bert_medium_uncased", "bert_base_uncased", "bert_base_cased", "bert_large_uncased")
)
```

Arguments

`values`	A character vector indicating the names of available models. The default uses the 7 named pre-trained BERT models. We recommend that you select specific models that are likely to work on your hardware. See `torchtransformers::available_berts()` for possible values.

Value

A parameter that can be tuned with the `tune` package.

Examples

```
if (rlang::is_installed("dials")) {
  bert_type()
}
```

`model_bert_linear` *Pretrained BERT Model with Linear Output*

Description

Construct a BERT model with pretrained weights, and add a final dense linear layer to transform to a desired number of dimensions. Note that we only use the CLS token output from the final layer of the BERT model. It is possible to attach a classification or regression head to BERT using other techniques, but here we use this simple technique.

Usage

```
model_bert_linear(bert_type = "bert_tiny_uncased", output_dim = 1L)
```

Arguments

`bert_type`	Character; which flavor of BERT to use. See `available_berts()` for known models.
`output_dim`	Integer; the target number of output dimensions.

Value

A torch neural net model with pretrained BERT weights and a final dense layer.

`n_tokens` *Number of Tokens*

Description

The number of tokens to use for tokenization of predictors.

Usage

```
n_tokens(range = c(1, 9), trans = scales::log2_trans())
```

Arguments

`range`	A two-element integer vector with the smallest and largest possible values. By default these values should be the powers of two to try.
`trans`	An optional transformation to apply. By default, `scales::log2_trans()` (meaning take the log2 of the original values, resulting in small-number range values).

Value

A parameter that can be tuned with the `tune` package.

Examples

```
if (rlang::is_installed("dials")) {
  n_tokens()
}
```

`predict.bert_classification` *Predict from a* `bert_classification` *model.*

Description

Predict from a `bert_classification` model.

Usage

```
## S3 method for class 'bert_classification'
predict(object, new_data, type = c("class", "prob"), ...)
```

Arguments

`object`	A `bert_classification` object.
`new_data`	A data frame or matrix of new character predictors. This data is automatically tokenized to match the tokenization expected by the BERT model.
`type`	A single character. The type of predictions to generate. Valid options are: • `"class"` for "hard" class predictions. • `"prob"` for class probabilities.
`...`	Not used, but required for extensibility.

Value

A tibble of predictions. The number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`.

`predict.bert_regression` *Predict from a* `bert_regression` *model.*

Description

Predict from a `bert_regression` model.

Usage

```
## S3 method for class 'bert_regression'
predict(object, new_data, ...)
```

Arguments

`object`	A `bert_regression` object.
`new_data`	A data frame or matrix of new character predictors. This data is automatically tokenized to match the tokenization expected by the BERT model.
`...`	Not used, but required for extensibility.

Value

A tibble of predictions. The number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`.

`tidy_bert_output` *Tidy the BERT Output*

Description

Given the output from a transformer model, construct tidy data frames for the layer outputs and the attention weights.

Usage

```
tidy_bert_output(bert_model_output, tokenized)
```

Arguments

`bert_model_output`	The output from a BERT model.
`tokenized`	The raw output from `torchtransformers::tokenize_bert`.

Value

A list of data frames, one for the layer output embeddings and one for the attention weights.

Index