Package ‘tfdatasets’

September 7, 2021

Type Package
Title Interface to 'TensorFlow' Datasets
Version 2.6.0
Description Interface to 'TensorFlow' Datasets, a high-level library for building complex input pipelines from simple, re-usable pieces. See <https://www.tensorflow.org/guide> for additional details.
License Apache License 2.0
URL https://github.com/rstudio/tfdatasets
BugReports https://github.com/rstudio/tfdatasets/issues
SystemRequirements TensorFlow >= 1.4 (https://www.tensorflow.org/)
Encoding UTF-8
LazyData true
Depends R (>= 3.1)
Imports reticulate (>= 1.10), tensorflow (>= 1.13.1), magrittr, rlang, tidyselect, stats, generics, vctrs
RoxygenNote 7.1.1
Suggests testthat, knitr, keras, rsample, rmarkdown, Metrics, dplyr, tfestimators
VignetteBuilder knitr
NeedsCompilation no
Author Tomasz Kalinowski [ctb, cph, cre],
Daniel Falbel [ctb, cph],
JJ Allaire [aut, cph],
Yuan Tang [aut] (<https://orcid.org/0000-0001-5243-233X>),
Kevin Ushey [aut],
RStudio [cph, fnd],
Google Inc. [cph]
Maintainer Tomasz Kalinowski <tomasz.kalinowski@rstudio.com>
Repository CRAN
Date/Publication 2021-09-07 15:40:02 UTC
\textbf{R topics documented:}

- all_nominal .......................................................... 3
- all_numeric .......................................................... 4
- dataset_batch ....................................................... 4
- dataset_cache ....................................................... 5
- dataset_collect ..................................................... 5
- dataset_concatenate ............................................... 6
- dataset_decode_delim .............................................. 7
- dataset_enumerate ................................................ 7
- dataset_filter ...................................................... 8
- dataset_flat_map .................................................. 9
- dataset_interleave ............................................... 10
- dataset_map ......................................................... 11
- dataset_map_and_batch .......................................... 12
- dataset_options ................................................... 13
- dataset_padded_batch ............................................ 14
- dataset_prefetch .................................................. 15
- dataset_prefetch_to_device .................................... 16
- dataset_prepare .................................................. 17
- dataset_reduce ................................................... 18
- dataset_repeat .................................................... 19
- dataset_scan ....................................................... 19
- dataset_shard ..................................................... 20
- dataset_shuffle ................................................... 21
- dataset_shuffle_and_repeat .................................... 22
- dataset_skip ....................................................... 22
- dataset_snapshot ................................................ 23
- dataset_take ....................................................... 25
- dataset_use_spec ................................................ 25
- dataset_window .................................................... 26
- delim_record_spec ................................................ 27
- dense_features .................................................... 28
- feature_spec ....................................................... 29
- file_list_dataset ................................................ 30
- fit.FeatureSpec ................................................... 31
- fixed_length_record_dataset .................................. 32
- has_type .......................................................... 33
- hearts ............................................................. 33
- input_fn.tf_dataset .............................................. 34
- iterator_get_next ................................................. 35
- iterator_initializer .............................................. 35
- iterator_make_initializer ...................................... 36
- iterator_string_handle ......................................... 36
- layer_input_from_dataset ...................................... 37
- length.tf_dataset ................................................ 38
- make-iterator .................................................... 38
- make_csv_dataset ................................................ 40
**all_nominal**

Find all nominal variables.

**Description**

Currently we only consider "string" type as nominal.

**Usage**

```python
all_nominal()
```

**See Also**

Other Selectors: `all_numeric()`, `has_type()`
all_numeric  

**Description**

Find all the variables with the following types: "float16", "float32", "float64", "int16", "int32", "int64", "half", "double".

**Usage**

```
all_numeric()
```

**See Also**

Other Selectors: `all_nominal()`, `has_type()`

dataset_batch  

**Description**

Combines consecutive elements of this dataset into batches.

**Usage**

```
dataset_batch(dataset, batch_size, drop_remainder = FALSE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>A dataset</td>
</tr>
<tr>
<td>batch_size</td>
<td>An integer, representing the number of consecutive elements of this dataset to combine in a single batch.</td>
</tr>
<tr>
<td>drop_remainder</td>
<td>Ensure that batches have a fixed size by omitting any final smaller batch if it’s present. Note that this is required for use with the Keras tensor inputs to fit/evaluate/etc.</td>
</tr>
</tbody>
</table>

**Value**

A dataset

**See Also**

Other dataset methods: `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`
**dataset_cache**

Caches the elements in this dataset.

**Description**

Caches the elements in this dataset.

**Usage**

```r
dataset_cache(dataset, filename = NULL)
```

**Arguments**

- **dataset**: A dataset
- **filename**: String with the name of a directory on the filesystem to use for caching tensors in this Dataset. If a filename is not provided, the dataset will be cached in memory.

**Value**

A dataset

**See Also**

Other dataset methods: `dataset_batch()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`

**dataset_collect**

Collects a dataset

**Description**

Iterates through the dataset collecting every element into a list. It's useful for looking at the full result of the dataset. Note: You may run out of memory if your dataset is too big.

**Usage**

```r
dataset_collect(dataset, iter_max = Inf)
```

**Arguments**

- **dataset**: A dataset
- **iter_max**: Maximum number of iterations. `Inf` until the end of the dataset
See Also

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`

---

**dataset_concatenate**  
*Creates a dataset by concatenating given dataset with this dataset.*

**Description**

Creates a dataset by concatenating given dataset with this dataset.

**Usage**

```python
dataset_concatenate(dataset, other)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>A dataset</td>
</tr>
<tr>
<td>other</td>
<td>Dataset to be concatenated</td>
</tr>
</tbody>
</table>

**Value**

A dataset

**Note**

Input dataset and dataset to be concatenated should have same nested structures and output types.

**See Also**

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`
dataset_decode_delim

Transform a dataset with delimited text lines into a dataset with named columns

Description
Transform a dataset with delimited text lines into a dataset with named columns

Usage
dataset_decode_delim(dataset, record_spec, parallel_records = NULL)

Arguments
- dataset: Dataset containing delimited text lines (e.g. a CSV)
- record_spec: Specification of column names and types (see delim_record_spec()).
- parallel_records: (Optional) An integer, representing the number of records to decode in parallel. If not specified, records will be processed sequentially.

See Also
Other dataset methods: dataset_batch(), dataset_cache(), dataset_collect(), dataset_concatenate(), dataset_filter(), dataset_interleave(), dataset_map_and_batch(), dataset_map(), dataset_padded_batch(), dataset_prefetch_to_device(), dataset_prefetch(), dataset_reduce(), dataset_repeat(), dataset_shuffle_and_repeat(), dataset_shuffle(), dataset_skip(), dataset_take(), dataset_window()
Details

It is similar to python’s enumerate, this transforms a sequence of elements into a sequence of list(index, element), where index is an integer that indicates the position of the element in the sequence.

Examples

```
## Not run:
dataset <- tensor_slices_dataset(100:103) %>%
  dataset_enumerate()

iterator <- reticulate::as_iterator(dataset)
reticulate::iter_next(iterator) # list(0, 100)
reticulate::iter_next(iterator) # list(1, 101)
reticulate::iter_next(iterator) # list(2, 102)
reticulate::iter_next(iterator) # list(3, 103)
reticulate::iter_next(iterator) # NULL (iterator exhausted)
reticulate::iter_next(iterator) # NULL (iterator exhausted)

## End(Not run)
```

---

**dataset_filter**

*Filter a dataset by a predicate*

Description

Filter a dataset by a predicate

Usage

```r
dataset_filter(dataset, predicate)
```

Arguments

- **dataset**: A dataset
- **predicate**: A function mapping a nested structure of tensors (having shapes and types defined by `output_shapes()` and `output_types()` to a scalar `tf$bool` tensor.

Details

Note that the functions used inside the predicate must be tensor operations (e.g. `tf$not_equal`, `tf$less`, etc.). R generic methods for relational operators (e.g. `<`, `>`, `<=`, etc.) and logical operators (e.g. `!`, `&`, `|`, etc.) are provided so you can use shorthand syntax for most common comparisons (this is illustrated by the example below).

Value

A dataset composed of records that matched the predicate.
dataset_flat_map

See Also

Other dataset methods: dataset_batch(), dataset_cache(), dataset_collect(), dataset_concatenate(), dataset_decode_delim(), dataset_interleave(), dataset_map_and_batch(), dataset_map(), dataset_padded_batch(), dataset_prefetch_to_device(), dataset_prefetch(), dataset_reduce(), dataset_repeat(), dataset_shuffle_and_repeat(), dataset_shuffle(), dataset_skip(), dataset_take(), dataset_window()

Examples

```r
## Not run:

dataset <- text_line_dataset("mtcars.csv", record_spec = mtcars_spec) %>%
  dataset_filter(function(record) {
    record$mpg >= 20
  })

dataset <- text_line_dataset("mtcars.csv", record_spec = mtcars_spec) %>%
  dataset_filter(function(record) {
    record$mpg >= 20 & record$cyl >= 6L
  })

## End(Not run)
```

---

dataset_flat_map  

Maps map_func across this dataset and flattens the result.

Description

Maps map_func across this dataset and flattens the result.

Usage

dataset_flat_map(dataset, map_func)

Arguments

dataset  

A dataset

map_func  

A function mapping a nested structure of tensors (having shapes and types defined by output_shapes() and output_types()) to a dataset.

Value

A dataset
dataset_interleave  

Maps `map_func` across this dataset, and interleaves the results

Description

Maps `map_func` across this dataset, and interleaves the results

Usage

```r
dataset_interleave(dataset, map_func, cycle_length, block_length = 1)
```

Arguments

- `dataset`: A dataset
- `map_func`: A function mapping a nested structure of tensors (having shapes and types defined by `output_shapes()` and `output_types()` to a dataset.
- `cycle_length`: The number of elements from this dataset that will be processed concurrently.
- `block_length`: The number of consecutive elements to produce from each input element before cycling to another input element.

Details

The `cycle_length` and `block_length` arguments control the order in which elements are produced. `cycle_length` controls the number of input elements that are processed concurrently. In general, this transformation will apply `map_func` to `cycle_length` input elements, open iterators on the returned dataset objects, and cycle through them producing `block_length` consecutive elements from each iterator, and consuming the next input element each time it reaches the end of an iterator.

See Also

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`

Examples

```r
## Not run:
dataset <- tensor_slices_dataset(c(1,2,3,4,5)) %>%
dataset_interleave(cycle_length = 2, block_length = 4, function(x) {
  tensors_dataset(x) %>%
  dataset_repeat(6)
})
# resulting dataset (newlines indicate "block" boundaries):
```

dataset_map

Map a function across a dataset.

Description
Map a function across a dataset.

Usage
dataset_map(dataset, map_func, num_parallel_calls = NULL)

Arguments
dataset A dataset
map_func A function mapping a nested structure of tensors (having shapes and types defined by output_shapes() and output_types() to another nested structure of tensors. It also supports purrr style lambda functions powered by rlang::as_function().
num_parallel_calls (Optional) An integer, representing the number of elements to process in parallel. If not specified, elements will be processed sequentially.

Value
A dataset

See Also
Other dataset methods: dataset_batch(), dataset_cache(), dataset_collect(), dataset_concatenate(), dataset_decode_delim(), dataset_filter(), dataset_interleave(), dataset_map_and_batch(), dataset_padded_batch(), dataset_prefetch_to_device(), dataset_prefetch(), dataset_reduce(), dataset_repeat(), dataset_shuffle_and_repeat(), dataset_shuffle(), dataset_skip(), dataset_take(), dataset_window()
**Description**

Maps `map_func` across `batch_size` consecutive elements of this dataset and then combines them into a batch. Functionally, it is equivalent to `map` followed by `batch`. However, by fusing the two transformations together, the implementation can be more efficient.

**Usage**

```
dataset_map_and_batch(
  dataset,
  map_func,
  batch_size,
  num_parallel_batches = NULL,
  drop_remainder = FALSE,
  num_parallel_calls = NULL
)
```

**Arguments**

- **dataset**
  A dataset

- **map_func**
  A function mapping a nested structure of tensors (having shapes and types defined by `output_shapes()` and `output_types()`) to another nested structure of tensors. It also supports `purrr` style lambda functions powered by `rlang::as_function()`.

- **batch_size**
  An integer, representing the number of consecutive elements of this dataset to combine in a single batch.

- **num_parallel_batches**
  (Optional) An integer, representing the number of batches to create in parallel. On one hand, higher values can help mitigate the effect of stragglers. On the other hand, higher values can increase contention if CPU is scarce.

- **drop_remainder**
  Ensure that batches have a fixed size by omitting any final smaller batch if it's present. Note that this is required for use with the Keras tensor inputs to `fit/evaluate/etc.`

- **num_parallel_calls**
  (Optional) An integer, representing the number of elements to process in parallel. If not specified, elements will be processed sequentially.

**See Also**

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`
**dataset_options**

*Get or Set Dataset Options*

**Description**

Get or Set Dataset Options

**Usage**

```r
dataset_options(dataset, ...)
```

**Arguments**

- `dataset`: a tensorflow dataset
- `...`: Valid values include:
  - A set of named arguments setting options. Names of nested attributes can be separated with a "." (see examples). The set of named arguments can be supplied individually to `...`, or as a single named list.
  - a `tf$data$Options()` instance.

**Details**

The options are "global" in the sense they apply to the entire dataset. If options are set multiple times, they are merged as long as different options do not use different non-default values.

**Value**

- If values are supplied to `...`, returns a `tf.data.Dataset` with the given options set/updated. Otherwise, returns the currently set options for the dataset.

**Examples**

```r
## Not run:
# pass options directly:
range_dataset(0, 10) %>%
  dataset_options(
    experimental_deterministic = FALSE,
    threading.private_threadpool_size = 10
  )

# pass options as a named list:
opts <- list(
  experimental_deterministic = FALSE,
  threading.private_threadpool_size = 10
)
range_dataset(0, 10) %>%
  dataset_options(opts)
```
# pass a tf.data.Options() instance
opts <- tf$data$Options()
 opts$experimental_deterministic <- FALSE
 opts$threading$private_threadpool_size <- 10L
 range_dataset(0, 10) %>%
  dataset_options(opts)

# get currently set options
range_dataset(0, 10) %>% dataset_options()

## End(Not run)

dataset_padded_batch  Combines consecutive elements of this dataset into padded batches

Description

This method combines multiple consecutive elements of this dataset, which might have different shapes, into a single element. The tensors in the resulting element have an additional outer dimension, and are padded to the respective shape in padded_shapes.

Usage

dataset_padded_batch(
  dataset,
  batch_size,
  padded_shapes,
  padding_values = NULL,
  drop_remainder = FALSE
)

Arguments

dataset  A dataset

batch_size  An integer, representing the number of consecutive elements of this dataset to combine in a single batch.

padded_shapes  A nested structure of tf$TensorShape or integer vector tensor-like objects representing the shape to which the respective component of each input element should be padded prior to batching. Any unknown dimensions (e.g. tf$Dimension(NULL) in a tf$TensorShape or -1 in a tensor-like object) will be padded to the maximum size of that dimension in each batch.

padding_values  (Optional) A nested structure of scalar-shaped tf$Tensor, representing the padding values to use for the respective components. Defaults are 0 for numeric types and the empty string for string types.

drop_remainder  Ensure that batches have a fixed size by omitting any final smaller batch if it’s present. Note that this is required for use with the Keras tensor inputs to fit/evaluate/etc.
Value  
A dataset

See Also  
Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`
dataset_prefetch_to_device

*Description*

A transformation that prefetches dataset values to the given device

*Usage*

```python
dataset_prefetch_to_device(dataset, device, buffer_size = NULL)
```

*Arguments*

- **dataset**: A dataset
- **device**: A string. The name of a device to which elements will be prefetched (e.g. "/gpu:0").
- **buffer_size**: (Optional.) The number of elements to buffer on device. Defaults to an automatically chosen value.

*Value*

A dataset

*Note*

Although the transformation creates a dataset, the transformation must be the final dataset in the input pipeline.

*See Also*

Other dataset methods: dataset_batch(), dataset_cache(), dataset_collect(), dataset_concatenate(), dataset_decode_delim(), dataset_filter(), dataset_interleave(), dataset_map_and_batch(), dataset_map(), dataset_padded_batch(), dataset_prefetch(), dataset_reduce(), dataset_repeat(), dataset_shuffle_and_repeat(), dataset_shuffle(), dataset_skip(), dataset_take(), dataset_window()
**dataset_prepare**

Prepare a dataset for analysis

**Description**

Transform a dataset with named columns into a list with features \((x)\) and response \((y)\) elements.

**Usage**

```r
dataset_prepare(
  dataset,
  x,
  y = NULL,
  named = TRUE,
  named_features = FALSE,
  parallel_records = NULL,
  batch_size = NULL,
  num_parallel_batches = NULL,
  drop_remainder = FALSE
)
```

**Arguments**

- **dataset**: A dataset
- **x**: Features to include. When `named_features` is `FALSE` all features will be stacked into a single tensor so must have an identical data type.
- **y**: (Optional). Response variable.
- **named**: `TRUE` to name the dataset elements "x" and "y", `FALSE` to not name the dataset elements.
- **named_features**: `TRUE` to yield features as a named list; `FALSE` to stack features into a single array. Note that in the case of `FALSE` (the default) all features will be stacked into a single 2D tensor so need to have the same underlying data type.
- **parallel_records**: (Optional) An integer, representing the number of records to decode in parallel. If not specified, records will be processed sequentially.
- **batch_size**: (Optional). Batch size if you would like to fuse the `dataset_prepare()` operation together with a `dataset_batch()` (fusing generally improves overall training performance).
- **num_parallel_batches**: (Optional) An integer, representing the number of batches to create in parallel. On one hand, higher values can help mitigate the effect of stragglers. On the other hand, higher values can increase contention if CPU is scarce.
- **drop_remainder**: Ensure that batches have a fixed size by omitting any final smaller batch if it’s present. Note that this is required for use with the Keras tensor inputs to fit/evaluate/etc.
dataset_reduce

Value

A dataset. The dataset will have a structure of either:

- When `named_features` is `TRUE`: `list(x = list(feature_name = feature_values,...), y = response_values)`
- When `named_features` is `FALSE`: `list(x = features_array, y = response_values)`, where `features_array` is a Rank 2 array of `(batch_size, num_features)`

Note that the `y` element will be omitted when `y` is `NULL`.

See Also

`input_fn()` for use with `tfestimators`.

---

dataset_reduce  

Reduces the input dataset to a single element.

Description

The transformation calls `reduce_func` successively on every element of the input dataset until the dataset is exhausted, aggregating information in its internal state. The `initial_state` argument is used for the initial state and the final state is returned as the result.

Usage

```python
dataset_reduce(dataset, initial_state, reduce_func)
```

Arguments

dataset  
initial_state  
reduce_func

- **dataset**: A dataset
- **initial_state**: An element representing the initial state of the transformation.
- **reduce_func**: A function that maps `(old_state, input_element)` to `new_state`. It must take two arguments and return a new element. The structure of `new_state` must match the structure of `initial_state`.

Value

A dataset element.

See Also

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`
**dataset_repeat**  
*Repeats a dataset count times.*

**Description**  
Repeats a dataset count times.

**Usage**  
```
dataset_repeat(dataset, count = NULL)
```

**Arguments**

- **dataset**  
  A dataset

- **count**  
  (Optional.) An integer value representing the number of times the elements of this dataset should be repeated. The default behavior (if `count` is `NULL` or `-1`) is for the elements to be repeated indefinitely.

**Value**  
A dataset

**See Also**

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`

---

**dataset_scan**  
*A transformation that scans a function across an input dataset*

**Description**  
A transformation that scans a function across an input dataset

**Usage**  
```
dataset_scan(dataset, initial_state, scan_func)
```
Arguments

dataset A tensorflow dataset
initial_state A nested structure of tensors, representing the initial state of the accumulator.
scan_func A function that maps (old_state, input_element) to (new_state, output_element).
It must take two arguments and return a pair of nested structures of tensors. The new_state must match the structure of initial_state.

Details

This transformation is a stateful relative of dataset_map(). In addition to mapping scan_func across the elements of the input dataset, scan() accumulates one or more state tensors, whose initial values are initial_state.

Examples

## Not run:
initial_state <- as_tensor(0, dtype="int64")
scan_func <- function(state, i) list(state + i, state + i)
dataset <- range_dataset(0, 10) %>%
  dataset_scan(initial_state, scan_func)

reticulate::iterate(dataset, as.array) %>%
  unlist()
# 0 1 3 6 10 15 21 28 36 45

## End(Not run)

---

dataset_shard Creates a dataset that includes only 1 / num_shards of this dataset.

Description

This dataset operator is very useful when running distributed training, as it allows each worker to read a unique subset.

Usage

dataset_shard(dataset, num_shards, index)

Arguments

dataset A dataset
num_shards A integer representing the number of shards operating in parallel.
index A integer, representing the worker index.

Value

A dataset
dataset_shuffle

Randomly shuffles the elements of this dataset.

Description

Randomly shuffles the elements of this dataset.

Usage

dataset_shuffle(
  dataset,
  buffer_size,
  seed = NULL,
  reshuffle_each_iteration = NULL
)

Arguments

dataset
  A dataset

buffer_size
  An integer, representing the number of elements from this dataset from which
  the new dataset will sample.

seed
  (Optional) An integer, representing the random seed that will be used to create
  the distribution.

reshuffle_each_iteration
  (Optional) A boolean, which if true indicates that the dataset should be pseudo-
  randomly reshuffled each time it is iterated over. (Defaults to TRUE). Not used if
  TF version < 1.15

Value

A dataset

See Also

Other dataset methods: dataset_batch(), dataset_cache(), dataset_collect(), dataset_concatenate(),
dataset_decode_delim(), dataset_filter(), dataset_interleave(), dataset_map_and_batch(),
dataset_map(), dataset_padded_batch(), dataset_prefetch_to_device(), dataset_prefetch(),
dataset_reduce(), dataset_repeat(), dataset_shuffle_and_repeat(), dataset_skip(), dataset_take(),
dataset_window()
**dataset_shuffle_and_repeat**

*Shuffles and repeats a dataset returning a new permutation for each epoch.*

**Description**

Shuffles and repeats a dataset returning a new permutation for each epoch.

**Usage**

```r
dataset_shuffle_and_repeat(dataset, buffer_size, count = NULL, seed = NULL)
```

**Arguments**

- **dataset**: A dataset
- **buffer_size**: An integer, representing the number of elements from this dataset from which the new dataset will sample.
- **count**: (Optional.) An integer value representing the number of times the elements of this dataset should be repeated. The default behavior (if `count` is `NULL` or `-1`) is for the elements to be repeated indefinitely.
- **seed**: (Optional) An integer, representing the random seed that will be used to create the distribution.

**See Also**

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`, `dataset_window()`

**dataset_skip**

*Creates a dataset that skips count elements from this dataset*

**Description**

Creates a dataset that skips count elements from this dataset

**Usage**

```r
dataset_skip(dataset, count)
```
**dataset_snapshot**

Persist the output of a dataset

---

**Description**

Persist the output of a dataset

**Usage**

```r
dataset_snapshot(
  dataset,
  path,
  compression = c("AUTO", "GZIP", "SNAPPY", "None"),
  reader_func = NULL,
  shard_func = NULL
)
```

**Arguments**

- `dataset`: A tensorflow dataset
- `path`: Required. A directory to use for storing/loading the snapshot to/from.
- `compression`: Optional. The type of compression to apply to the snapshot written to disk. Supported options are "GZIP", "SNAPPY", "AUTO" or NULL (values of "", NA, and "None" are synonymous with NULL) Defaults to AUTO, which attempts to pick an appropriate compression algorithm for the dataset.
- `reader_func`: Optional. A function to control how to read data from snapshot shards.
- `shard_func`: Optional. A function to control how to shard data when writing a snapshot.
Details

The snapshot API allows users to transparently persist the output of their preprocessing pipeline to disk, and materialize the pre-processed data on a different training run.

This API enables repeated preprocessing steps to be consolidated, and allows re-use of already processed data, trading off disk storage and network bandwidth for freeing up more valuable CPU resources and accelerator compute time.

https://github.com/tensorflow/community/blob/master/rfcs/20200107-tf-data-snapshot.md has detailed design documentation of this feature.

Users can specify various options to control the behavior of snapshot, including how snapshots are read from and written to by passing in user-defined functions to the reader_func and shard_func parameters.

shard_func is a user specified function that maps input elements to snapshot shards.

```r
NUM_SHARDS <- parallel::detectCores()
dataset %>%
  dataset_enumerate() %>%
  dataset_snapshot(
    "/path/to/snapshot/dir",
    shard_func = function(index, ds_elem) x %% NUM_SHARDS) %>%
  dataset_map(function(index, ds_elem) ds_elem)
```

reader_func is a user specified function that accepts a single argument: a Dataset of Datasets, each representing a "split" of elements of the original dataset. The cardinality of the input dataset matches the number of the shards specified in the shard_func. The function should return a Dataset of elements of the original dataset.

Users may want specify this function to control how snapshot files should be read from disk, including the amount of shuffling and parallelism.

Here is an example of a standard reader function a user can define. This function enables both dataset shuffling and parallel reading of datasets:

```r
user_reader_func <- function(datasets) {
  num_cores <- parallel::detectCores()
  datasets %>%
    dataset_shuffle(num_cores) %>%
    dataset_interleave(function(x) x, num_parallel_calls=AUTOTUNE)
}

dataset <- dataset %>%
  dataset_snapshot("/path/to/snapshot/dir",
    reader_func = user_reader_func)
```

By default, snapshot parallelizes reads by the number of cores available on the system, but will not attempt to shuffle the data.
**dataset_take**  
*Creates a dataset with at most count elements from this dataset*

**Description**  
Creates a dataset with at most count elements from this dataset.

**Usage**

```python
dataset_take(dataset, count)
```

**Arguments**

- **dataset**: A dataset.
- **count**: Integer representing the number of elements of this dataset that should be taken to form the new dataset. If count is -1, or if count is greater than the size of this dataset, the new dataset will contain all elements of this dataset.

**Value**

A dataset

**See Also**

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_window()`

---

**dataset_use_spec**  
*Transform the dataset using the provided spec.*

**Description**  
Prepares the dataset to be used directly in a model. The transformed dataset is prepared to return tuples (x,y) that can be used directly in Keras.

**Usage**

```python
dataset_use_spec(dataset, spec)
```

**Arguments**

- **dataset**: A TensorFlow dataset.
- **spec**: A feature specification created with `feature_spec()`.
dataset_window

Combines input elements into a dataset of windows.

Description

Combines input elements into a dataset of windows.

Usage

dataset_window(dataset, size, shift = NULL, stride = 1, drop_remainder = FALSE)

Arguments

dataset A dataset

type A dataset representing the number of elements of the input dataset to combine into a window.
**delim_record_spec**

- **shift**: representing the forward shift of the sliding window in each iteration. Defaults to size.
- **stride**: representing the stride of the input elements in the sliding window.
- **drop_remainder**: representing whether a window should be dropped in case its size is smaller than window_size.

### See Also

Other dataset methods: `dataset_batch()`, `dataset_cache()`, `dataset_collect()`, `dataset_concatenate()`, `dataset_decode_delim()`, `dataset_filter()`, `dataset_interleave()`, `dataset_map_and_batch()`, `dataset_map()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_shuffle()`, `dataset_skip()`, `dataset_take()`

---

**delim_record_spec** Specification for reading a record from a text file with delimited values

### Description

Specification for reading a record from a text file with delimited values

### Usage

```r
 delim_record_spec(
   example_file,
   delim = "",
   skip = 0,
   names = NULL,
   types = NULL,
   defaults = NULL
)

csv_record_spec(
   example_file,
   skip = 0,
   names = NULL,
   types = NULL,
   defaults = NULL
)

tsv_record_spec(
   example_file,
   skip = 0,
   names = NULL,
   types = NULL,
   defaults = NULL
)
```
Arguments

example_file  File that provides an example of the records to be read. If you don’t explicitly specify names and types (or defaults) then this file will be read to generate default values.
delim  Character delimiter to separate fields in a record (defaults to ",")
skip  Number of lines to skip before reading data. Note that if names is explicitly provided and there are column names within the file then skip should be set to 1 to ensure that the column names are bypassed.
names  Character vector with column names (or NULL to automatically detect the column names from the first row of example_file).
If names is a character vector, the values will be used as the names of the columns, and the first row of the input will be read into the first row of the dataset. Note that if the underlying text file also includes column names in its first row, this row should be skipped explicitly with skip = 1.
If NULL, the first row of the example_file will be used as the column names, and will be skipped when reading the dataset.
types  Column types. If NULL and defaults is specified then types will be imputed from the defaults. Otherwise, all column types will be imputed from the first 1000 rows of the example_file. This is convenient (and fast), but not robust. If the imputation fails, you’ll need to supply the correct types yourself.
Types can be explicitly specified in a character vector as "integer", "double", and "character" (e.g. col_types = c("double", "double", "integer"). Alternatively, you can use a compact string representation where each character represents one column: c = character, i = integer, d = double (e.g. types = ddi').
defaults  List of default values which are used when data is missing from a record (e.g. list(0, 0, 0L). If NULL then defaults will be automatically provided based on types (0 for numeric columns and "" for character columns).

dense_features  Dense Features

Description

Retrives the Dense Features from a spec.

Usage

dense_features(spec)

Arguments

spec  A feature specification created with feature_spec().

Value

A list of feature columns.
Create a feature specification.

Description

Used to create initialize a feature columns specification.

Usage

feature_spec(dataset, x, y = NULL)

Arguments

dataset A TensorFlow dataset.

x Features to include can use tidyselect::select_helpers() or a formula.

y (Optional) The response variable. Can also be specified using a formula in the x argument.

Details

After creating the feature_spec object you can add steps using the step functions.

Value

a FeatureSpec object.

See Also

• fit.FeatureSpec() to fit the FeatureSpec
• dataset_use_spec() to create a tensorflow dataset prepared to modeling.
• steps to a list of all implemented steps.

Other Feature Spec Functions: dataset_use_spec(), fit.FeatureSpec(), step_bucketized_column(), step_categorical_column_with_hash_bucket(), step_categorical_column_with_identity(), step_categorical_column_with_vocabulary_file(), step_categorical_column_with_vocabulary_list(), step_crossed_column(), step_embedding_column(), step_indicator_column(), step_numeric_column(), step_remove_column(), step_shared_embeddings_column(), steps

Examples

## Not run:
library(tfdatasets)
data(hearts)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ .)
# select using `tidyselect` helpers
spec <- feature_spec(hearts, x = c(thal, age), y = target)

## End(Not run)

---

**file_list_dataset**

*A dataset of all files matching a pattern*

### Description

A dataset of all files matching a pattern

### Usage

```
file_list_dataset(file_pattern, shuffle = NULL, seed = NULL)
```

### Arguments

- **file_pattern**
  A string, representing the filename pattern that will be matched.

- **shuffle**
  (Optional) If TRUE, the file names will be shuffled randomly. Defaults to TRUE.

- **seed**
  (Optional) An integer, representing the random seed that will be used to create the distribution.

### Details

For example, if we had the following files on our filesystem:

```
/path/to/dir/a.txt
/path/to/dir/b.csv
/path/to/dir/c.csv
```

If we pass "/path/to/dir/*.csv" as the file_pattern, the dataset would produce:

```
/path/to/dir/b.csv
/path/to/dir/c.csv
```

### Value

A dataset of string corresponding to file names

### Note

The shuffle and seed arguments only apply for TensorFlow >= v1.8
fit.FeatureSpec Fits a feature specification.

Description
This function will fit the specification. Depending on the steps added to the specification it will compute for example, the levels of categorical features, normalization constants, etc.

Usage

```r
## S3 method for class 'FeatureSpec'
fit(object, dataset = NULL, ...)
```

Arguments

- `object`: A feature specification created with `feature_spec()`.  
- `dataset`: (Optional) A TensorFlow dataset. If `NULL` it will use the dataset provided when initializing the `feature_spec`.  
- `...`: (unused)

Value

a fitted `FeatureSpec` object.

See Also

- `feature_spec()` to initialize the feature specification.  
- `dataset_use_spec()` to create a tensorflow dataset prepared to modeling.  
- `steps` to a list of all implemented steps.

Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `step_bucketized_column()`, `step_categorical_column_with_hash_bucket()`, `step_categorical_column_with_identity()`, `step_categorical_column_with_vocabulary_file()`, `step_categorical_column_with_vocabulary_list()`, `step_crossed_column()`, `step_embedding_column()`, `step_indicator_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`, `steps`

Examples

```r
## Not run:
library(tfdatasets)
data(hearts)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ age) %>%
  step_numeric_column(age)

spec_fit <- fit(spec)
```
fixed_length_record_dataset

A dataset of fixed-length records from one or more binary files.

Description

A dataset of fixed-length records from one or more binary files.

Usage

```r
fixed_length_record_dataset(
  filenames,
  record_bytes,
  header_bytes = NULL,
  footer_bytes = NULL,
  buffer_size = NULL
)
```

Arguments

- **filenames**: A string tensor containing one or more filenames.
- **record_bytes**: An integer representing the number of bytes in each record.
- **header_bytes**: (Optional) An integer scalar representing the number of bytes to skip at the start of a file.
- **footer_bytes**: (Optional) A integer scalar representing the number of bytes to ignore at the end of a file.
- **buffer_size**: (Optional) A integer scalar representing the number of bytes to buffer when reading.

Value

A dataset
has_type

Identify the type of the variable.

Description
Can only be used inside the steps specifications to find variables by type.

Usage

```r
has_type(match = "float32")
```

Arguments

- `match`: A list of types to match.

See Also

Other Selectors: `all_nominal()`, `all_numeric()`

---

hearts

Heart Disease Data Set

Description
Heart disease (angiographic disease status) dataset.

Usage

```r
hearts
```

Format
A data frame with 303 rows and 14 variables:

- **age**: age in years
- **sex**: sex (1 = male; 0 = female)
- **cp**: chest pain type: Value 1: typical angina, Value 2: atypical angina, Value 3: non-anginal pain, Value 4: asymptomatic
- **trestbps**: resting blood pressure (in mm Hg on admission to the hospital)
- **chol**: serum cholestoral in mg/dl
- **fbs**: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- **restecg**: resting electrocardiographic results: Value 0: normal, Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), Value 2: showing probable or definite left ventricular hypertrophy by Estes’ criteria
thalach maximum heart rate achieved
exang exercise induced angina (1 = yes; 0 = no)
oldpeak ST depression induced by exercise relative to rest
slope the slope of the peak exercise ST segment: Value 1: upsloping, Value 2: flat, Value 3: downsloping
ca number of major vessels (0-3) colored by flourosopy
thal 3 = normal; 6 = fixed defect; 7 = reversible defect
target diagnosis of heart disease angiographic

Source
https://archive.ics.uci.edu/ml/datasets/heart+Disease

References
The authors of the databases have requested that any publications resulting from the use of the data include the names of the principal investigator responsible for the data collection at each institution. They would be:

1. Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
2. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
3. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
4. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

input_fn.tf_dataset  Construct a tfestimators input function from a dataset

Description
Construct a tfestimators input function from a dataset

Usage
input_fn.tf_dataset(dataset, features, response = NULL)

Arguments
dataset A dataset
features The names of feature variables to be used.
response The name of the response variable.

Details
Creating an input_fn from a dataset requires that the dataset consist of a set of named output tensors (e.g. like the dataset produced by the tfrecord_dataset() or text_line_dataset() function).
Value

An input_fn suitable for use with tfestimators train, evaluate, and predict methods

iterator_get_next

Get next element from iterator

Description

Returns a nested list of tensors that when evaluated will yield the next element(s) in the dataset.

Usage

iterator_get_next(iterator, name = NULL)

Arguments

iterator An iterator
name (Optional) A name for the created operation.

Value

A nested list of tensors

See Also

Other iterator functions: iterator_initializer(), iterator_make_initializer(), iterator_string_handle().

make-iterator

iterator_initializer

An operation that should be run to initialize this iterator.

Description

An operation that should be run to initialize this iterator.

Usage

iterator_initializer(iterator)

Arguments

iterator An iterator

See Also

Other iterator functions: iterator_get_next(), iterator_make_initializer(), iterator_string_handle().

make-iterator
iterator_make_initializer

Create an operation that can be run to initialize this iterator

Description

Create an operation that can be run to initialize this iterator

Usage

iterator_make_initializer(iterator, dataset, name = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterator</td>
<td>An iterator</td>
</tr>
<tr>
<td>dataset</td>
<td>A dataset</td>
</tr>
<tr>
<td>name</td>
<td>(Optional) A name for the created operation.</td>
</tr>
</tbody>
</table>

Value

A tf$Operation that can be run to initialize this iterator on the given dataset.

See Also

Other iterator functions: iterator_get_next(), iterator_initializer(), iterator_string_handle(), make_iterator

iterator_string_handle

String-valued tensor that represents this iterator

Description

String-valued tensor that represents this iterator

Usage

iterator_string_handle(iterator, name = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterator</td>
<td>An iterator</td>
</tr>
<tr>
<td>name</td>
<td>(Optional) A name for the created operation.</td>
</tr>
</tbody>
</table>
layer_input_from_dataset

Value

Scalar tensor of type string

See Also

Other iterator functions: `iterator_get_next()`, `iterator_initializer()`, `iterator_make_initializer()`, `make-iterator`

layer_input_from_dataset

*Creates a list of inputs from a dataset*

Description

Create a list of Keras input layers that can be used together with `keras::layer_dense_features()`.

Usage

```
layer_input_from_dataset(dataset)
```

Arguments

- `dataset` a TensorFlow dataset or a data.frame

Value

a list of Keras input layers

Examples

```r
## Not run:
library(tfdatasets)
data(hearts)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ age + slope) %>%
     step_numeric_column(age, slope) %>%
     step_bucketized_column(age, boundaries = c(10, 20, 30))

spec <- fit(spec)
dataset <- hearts %>% dataset_use_spec(spec)

input <- layer_input_from_dataset(dataset)

## End(Not run)
```
### length.tf_dataset  
*Get Dataset length*

**Description**

Returns the length of the dataset.

**Usage**

```r
## S3 method for class 'tf_dataset'
length(x)
```

```r
## S3 method for class 'tensorflow.python.data.ops.dataset_ops.DatasetV2'
length(x)
```

**Arguments**

- `x` a `tf.data.Dataset` object.

**Value**

Either `Inf` if the dataset is infinite, `NA` if the dataset length is unknown, or an R numeric if it is known.

**Examples**

```r
# Not run:
range_dataset(0, 42) %>% length()
# 42

range_dataset(0, 42) %>% dataset_repeat() %>% length()
# Inf

range_dataset(0, 42) %>% dataset_repeat() %>%
  dataset_filter(function(x) TRUE) %>% length()
# NA

# End(Not run)
```

---

### make-iterator

*Creates an iterator for enumerating the elements of this dataset.*

**Description**

Creates an iterator for enumerating the elements of this dataset.
**Usage**

- `make_iterator_one_shot(dataset)`
- `make_iterator_initializable(dataset, shared_name = NULL)`
- `make_iterator_from_structure(
  output_types,
  output_shapes = NULL,
  shared_name = NULL
)
- `make_iterator_from_string_handle(
  string_handle,
  output_types,
  output_shapes = NULL
)

**Arguments**

- **dataset** A dataset
- **shared_name** (Optional) If non-empty, the returned iterator will be shared under the given name across multiple sessions that share the same devices (e.g. when using a remote server).
- **output_types** A nested structure of `tf$DType` objects corresponding to each component of an element of this iterator.
- **output_shapes** (Optional) A nested structure of `tf$TensorShape` objects corresponding to each component of an element of this dataset. If omitted, each component will have an unconstrained shape.
- **string_handle** A scalar tensor of type string that evaluates to a handle produced by the `iterator_string_handle()` method.

**Value**

An Iterator over the elements of this dataset.

**Initialization**

For `make_iterator_one_shot()`, the returned iterator will be initialized automatically. A "one-shot" iterator does not currently support re-initialization.

For `make_iterator_initializable()`, the returned iterator will be in an uninitialized state, and you must run the object returned from `iterator_initializer()` before using it.

For `make_iterator_from_structure()`, the returned iterator is not bound to a particular dataset, and it has no initializer. To initialize the iterator, run the operation returned by `iterator_make_initializer()`.
make_csv_dataset

Reads CSV files into a batched dataset

Description

Reads CSV files into a dataset, where each element is a (features, labels) list that corresponds to a batch of CSV rows. The features dictionary maps feature column names to tensors containing the corresponding feature data, and labels is a tensor containing the batch’s label data.

Usage

make_csv_dataset(
  file_pattern,
  batch_size,
  column_names = NULL,
  column_defaults = NULL,
  label_name = NULL,
  select_columns = NULL,
  field_delim = ",",
  use_quote_delim = TRUE,
  na_value = "",
  header = TRUE,
  num_epochs = NULL,
  shuffle = TRUE,
  shuffle_buffer_size = 10000,
  shuffle_seed = NULL,
  prefetch_buffer_size = 1,
  num_parallel_reads = 1,
  num_parallel_parser_calls = 2,
  sloppy = FALSE,
  num_rows_for_inference = 100
)

Arguments

file_pattern       List of files or glob patterns of file paths containing CSV records.
batch_size         An integer representing the number of records to combine in a single batch.
column_names       An optional list of strings that corresponds to the CSV columns, in order. One per column of the input record. If this is not provided, infers the column names from the first row of the records. These names will be the keys of the features dict of each dataset element.
column_defaults

A optional list of default values for the CSV fields. One item per selected column of the input record. Each item in the list is either a valid CSV dtype (integer, numeric, or string), or a tensor with one of the aforementioned types. The tensor can either be a scalar default value (if the column is optional), or an empty tensor (if the column is required). If a dtype is provided instead of a tensor, the column is also treated as required. If this list is not provided, tries to infer types based on reading the first `num_rows_for_inference` rows of files specified, and assumes all columns are optional, defaulting to 0 for numeric values and "" for string values. If both this and `select_columns` are specified, these must have the same lengths, and `column_defaults` is assumed to be sorted in order of increasing column index.

label_name

A optional string corresponding to the label column. If provided, the data for this column is returned as a separate tensor from the features dictionary, so that the dataset complies with the format expected by a TF Estiamtors and Keras.

select_columns

(Ignored if using TensorFlow version 1.8.) An optional list of integer indices or string column names, that specifies a subset of columns of CSV data to select. If column names are provided, these must correspond to names provided in `column_names` or inferred from the file header lines. When this argument is specified, only a subset of CSV columns will be parsed and returned, corresponding to the columns specified. Using this results in faster parsing and lower memory usage. If both this and `column_defaults` are specified, these must have the same lengths, and `column_defaults` is assumed to be sorted in order of increasing column index.

field_delim

An optional string. Defaults to ",". Char delimiter to separate fields in a record.

use_quote_delim

An optional bool. Defaults to TRUE. If false, treats double quotation marks as regular characters inside of the string fields.

na_value

Additional string to recognize as NA/NaN.

header

A bool that indicates whether the first rows of provided CSV files correspond to header lines with column names, and should not be included in the data.

num_epochs

An integer specifying the number of times this dataset is repeated. If NULL, cycles through the dataset forever.

shuffle

A bool that indicates whether the input should be shuffled.

shuffle_buffer_size

Buffer size to use for shuffling. A large buffer size ensures better shuffling, but increases memory usage and startup time.

shuffle_seed

Randomization seed to use for shuffling.

prefetch_buffer_size

An int specifying the number of feature batches to prefetch for performance improvement. Recommended value is the number of batches consumed per training step.

num_parallel_reads

Number of threads used to read CSV records from files. If >1, the results will be interleaved.
num_parallel_parser_calls

(Ignored if using TensorFlow version 1.11 or later.) Number of parallel invocations of the CSV parsing function on CSV records.

sloppy

If TRUE, reading performance will be improved at the cost of non-deterministic ordering. If FALSE, the order of elements produced is deterministic prior to shuffling (elements are still randomized if shuffle=TRUE. Note that if the seed is set, then order of elements after shuffling is deterministic). Defaults to FALSE.

num_rows_for_inference

Number of rows of a file to use for type inference if record_defaults is not provided. If NULL, reads all the rows of all the files. Defaults to 100.

Value

A dataset, where each element is a (features, labels) list that corresponds to a batch of batch_size CSV rows. The features dictionary maps feature column names to tensors containing the corresponding column data, and labels is a tensor containing the column data for the label column specified by label_name.

---

next_batch

Tensor(s) for retrieving the next batch from a dataset

Description

Tensor(s) for retrieving the next batch from a dataset

Usage

next_batch(dataset)

Arguments

dataset A dataset

Details

To access the underlying data within the dataset you iteratively evaluate the tensor(s) to read batches of data.

Note that in many cases you won’t need to explicitly evaluate the tensors. Rather, you will pass the tensors to another function that will perform the evaluation (e.g. the Keras layer_input() and compile() functions).

If you do need to perform iteration manually by evaluating the tensors, there are a couple of possible approaches to controlling/detecting when iteration should end.

One approach is to create a dataset that yields batches infinitely (traversing the dataset multiple times with different batches randomly drawn). In this case you’d use another mechanism like a global step counter or detecting a learning plateau.
Another approach is to detect when all batches have been yielded from the dataset. When the tensor reaches the end of iteration a runtime error will occur. You can catch and ignore the error when it occurs by wrapping your iteration code in the `with_dataset()` function.

See the examples below for a demonstration of each of these methods of iteration.

**Value**

Tensor(s) that can be evaluated to yield the next batch of training data.

**Examples**

```r
## Not run:

# iteration with 'infinite' dataset and explicit step counter

library(tfdatasets)
dataset <- text_line_dataset("mtcars.csv", record_spec = mtcars_spec) %>%
  dataset_prepare(x = c(mpg, disp), y = cyl) %>%
  dataset_shuffle(5000) %>%
  dataset_batch(128) %>%
  dataset_repeat() # repeat infinitely
batch <- next_batch(dataset)
steps <- 200
for (i in 1:steps) {
  # use batch$x and batch$y tensors
}

# iteration that detects and ignores end of iteration error

library(tfdatasets)
dataset <- text_line_dataset("mtcars.csv", record_spec = mtcars_spec) %>%
  dataset_prepare(x = c(mpg, disp), y = cyl) %>%
  dataset_batch(128) %>%
  dataset_repeat(10)
batch <- next_batch(dataset)
with_dataset({
  while(TRUE) {
    # use batch$x and batch$y tensors
  }
})

## End(Not run)
```

---

**output_types**

**Output types and shapes**

**Description**

Output types and shapes
Usage

\texttt{output_types(object)}

\texttt{output_shapes(object)}

Arguments

\texttt{object} \hspace{1em} A dataset or iterator

Value

\texttt{output_types()} \hspace{1em} returns the type of each component of an element of this object; \texttt{output_shapes()} \hspace{1em} returns the shape of each component of an element of this object

---

\texttt{random_integer_dataset}

\textit{Creates a Dataset of pseudorandom values}

Description

Creates a Dataset of pseudorandom values

Usage

\texttt{random_integer_dataset(seed = NULL)}

Arguments

\texttt{seed} \hspace{1em} (Optional) If specified, the dataset produces a deterministic sequence of values.

Details

The dataset generates a sequence of uniformly distributed integer values (dtype int64).

---

\texttt{range_dataset}

\textit{Creates a dataset of a step-separated range of values.}

Description

Creates a dataset of a step-separated range of values.

Usage

\texttt{range_dataset(from = 0, to = 0, by = 1, ..., dtype = tf$int64)}
read_files

Arguments

- `from` Range start
- `to` Range end (exclusive)
- `by` Increment of the sequence
- ... ignored
- `dtype` Output dtype. (Optional, default: tf\$int64).

---

read_files Read a dataset from a set of files

Description

Read files into a dataset, optionally processing them in parallel.

Usage

```r
read_files(
  files,
  reader,
  ...,
  parallel_files = 1,
  parallel_interleave = 1,
  num_shards = NULL,
  shard_index = NULL
)
```

Arguments

- `files` List of filenames or glob pattern for files (e.g. "*.csv")
- `reader` Function that maps a file into a dataset (e.g. text_line_dataset() or tfrecord_dataset()).
- ... Additional arguments to pass to reader function
- `parallel_files` An integer, number of files to process in parallel
- `parallel_interleave` An integer, number of consecutive records to produce from each file before cycling to another file.
- `num_shards` An integer representing the number of shards operating in parallel.
- `shard_index` An integer, representing the worker index. Shared indexes are 0 based so for e.g. 8 shards valid indexes would be 0-7.

Value

A dataset
sample_from_datasets  \textit{Samples elements at random from the datasets in datasets.}

\section*{Description}

Samples elements at random from the datasets in datasets.

\section*{Usage}

\begin{verbatim}
sample_from_datasets(datasets, weights = NULL, seed = NULL)
\end{verbatim}

\section*{Arguments}

- \textbf{datasets}  
  A list of objects with compatible structure.

- \textbf{weights}  
  (Optional.) A list of length(datasets) floating-point values where weights[[i]] represents the probability with which an element should be sampled from datasets[[i]], or a dataset object where each element is such a list. Defaults to a uniform distribution across datasets.

- \textbf{seed}  
  (Optional.) An integer, representing the random seed that will be used to create the distribution.

\section*{Value}

A dataset that interleaves elements from datasets at random, according to weights if provided, otherwise with uniform probability.

\section*{scaler}

\textit{List of pre-made scalers}

\section*{Description}

- \textbf{scaler\_standard}: mean and standard deviation normalizer.
- \textbf{scaler\_min\_max}: min max normalizer

\section*{See Also}

\begin{verbatim}
step_numeric_column
\end{verbatim}
scaler_min_max

Creates an instance of a min max scaler

Description
This scaler will learn the min and max of the numeric variable and use this to create a normalizer_fn.

Usage
scaler_min_max()

See Also
scaler to a complete list of normalizers
Other scaler: scaler_standard()

scaler_standard

Creates an instance of a standard scaler

Description
This scaler will learn the mean and the standard deviation and use this to create a normalizer_fn.

Usage
scaler_standard()

See Also
scaler to a complete list of normalizers
Other scaler: scaler_min_max()
### Selectors

**Description**

List of selectors that can be used to specify variables inside steps.

**Usage**

```r
cur_info_env
```

**Format**

An object of class `environment` of length 0.

**Selectors**

- `has_type()`
- `all_numeric()`  
- `all_nominal()`  
- `starts_with()`  
- `ends_with()`  
- `one_of()`  
- `matches()`  
- `contains()`  
- `everything()`

---

### sparse_tensor_slices_dataset

*Splits each rank-N `tf$SparseTensor` in this dataset row-wise.*

**Description**

Splits each rank-N `tf$SparseTensor` in this dataset row-wise.

**Usage**

```r
sparse_tensor_slices_dataset(sparse_tensor)
```

**Arguments**

- `sparse_tensor` A `tf$SparseTensor`. 
**sql_record_spec**

Value

A dataset of rank-(N-1) sparse tensors.

See Also

Other tensor datasets: `tensor_slices_dataset()`, `tensors_dataset()`

---

**sql_record_spec**  
*A dataset consisting of the results from a SQL query*

---

Description

A dataset consisting of the results from a SQL query

Usage

```python
sql_record_spec(names, types)
```

```python
sql_dataset(driver_name, data_source_name, query, record_spec)
```

```python
sqlite_dataset(filename, query, record_spec)
```

Arguments

- **names**: Names of columns returned from the query
- **types**: List of `tf$DType` objects (e.g. `tf$int32`, `tf$double`, `tf$string`) representing the types of the columns returned by the query.
- **driver_name**: String containing the database type. Currently, the only supported value is 'sqlite'.
- **data_source_name**: String containing a connection string to connect to the database.
- **query**: String containing the SQL query to execute.
- **record_spec**: Names and types of database columns
- **filename**: Filename for the database

Value

A dataset
Description

List of steps that can be used to specify columns in the feature_spec interface.

Steps

- `step_numeric_column()` to define numeric columns.
- `step_categorical_column_with_vocabulary_list()` to define categorical columns.
- `step_categorical_column_with_hash_bucket()` to define categorical columns where ids are set by hashing.
- `step_categorical_column_with_identity()` to define categorical columns represented by integers in the range [0-num_buckets).
- `step_categorical_column_with_vocabulary_file()` to define categorical columns when their vocabulary is available in a file.
- `step_indicator_column()` to create indicator columns from categorical columns.
- `step_embedding_column()` to create embeddings columns from categorical columns.
- `step_bucketized_column()` to create bucketized columns from numeric columns.
- `step_crossed_column()` to perform crosses of categorical columns.
- `step_shared_embeddings_column()` to share embeddings between a list of categorical columns.
- `step_remove_column()` to remove columns from the specification.

See Also

- `selectors` for a list of selectors that can be used to specify variables.

Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step_bucketized_column()`, `step_categorical_column_with_hash_bucket()`, `step_categorical_column_with_identity()`, `step_categorical_column_with_vocabulary_file()`, `step_categorical_column_with_vocabulary_list()`, `step_crossed_column()`, `step_embedding_column()`, `step_indicator_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`

---

**step_bucketized_column**

*Creates bucketized columns*

Description

Use this step to create bucketized columns from numeric columns.
Usage

```r
step_bucketized_column(spec, ..., boundaries)
```

Arguments

- `spec` A feature specification created with `feature_spec()`.
- `...` Comma separated list of variable names to apply the step. `selectors` can also be used.
- `boundaries` A sorted list or tuple of floats specifying the boundaries.

Value

A `FeatureSpec` object.

See Also

- `steps` for a complete list of allowed steps.
- Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step_categorical_column_with_hash_bucket()`, `step_categorical_column_with_identity()`, `step_categorical_column_with_vocabulary_file()`, `step_categorical_column_with_vocabulary_list()`, `step_crossed_column()`, `step_embedding_column()`, `step_indicator_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`, `steps`

Examples

```r
## Not run:
library(tfdatasets)
data(hearts)
file <- tempfile()
writeLines(unique(hearts$thal), file)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)
# use the formula interface
spec <- feature_spec(hearts, target ~ age) %>%
  step_numeric_column(age) %>%
  step_bucketized_column(age, boundaries = c(10, 20, 30))
spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)
## End(Not run)
```

---

### step_categorical_column_with_hash_bucket

**Creates a categorical column with hash buckets specification**

Description

Represents sparse feature where ids are set by hashing.
**Usage**

```r
code
step_categorical_column_with_hash_bucket(
  spec,
  ...,  # Comma separated list of variable names to apply the step. selectors can also be used.
  hash_bucket_size,
  dtype = tf$string
)
```

**Arguments**

- `spec` A feature specification created with `feature_spec()`.
- `...` Comma separated list of variable names to apply the step. selectors can also be used.
- `hash_bucket_size` An int > 1. The number of buckets.
- `dtype` The type of features. Only string and integer types are supported.

**Value**

a FeatureSpec object.

**See Also**

- `steps` for a complete list of allowed steps.
- Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step_bucketized_column()`, `step_categorical_column_with_identity()`, `step_categorical_column_with_vocabulary_file()`, `step_categorical_column_with_vocabulary_list()`, `step_crossed_column()`, `step_embedding_column()`, `step_indicator_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`, `steps`

**Examples**

```r
code
## Not run:
library(tfdatasets)
data(hearts)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ thal) %>%
  step_categorical_column_with_hash_bucket(thal, hash_bucket_size = 3)

spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)

## End(Not run)
```
step_categorical_column_with_identity

Create a categorical column with identity

Description

Use this when your inputs are integers in the range [0-num_buckets).

Usage

```r
code
step_categorical_column_with_identity(
  spec,
  ..., 
  num_buckets, 
  default_value = NULL
)
```

Arguments

- `spec`: A feature specification created with `feature_spec()`.
- `...`: Comma separated list of variable names to apply the step. `selectors` can also be used.
- `num_buckets`: Range of inputs and outputs is [0, num_buckets).
- `default_value`: If `NULL`, this column’s graph operations will fail for out-of-range inputs. Otherwise, this value must be in the range [0, num_buckets), and will replace inputs in that range.

Value

A `FeatureSpec` object.

See Also

- `steps` for a complete list of allowed steps.
- Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step_bucketized_column()`, `step_categorical_column_with_hash_bucket()`, `step_categorical_column_with_vocabulary_file()`, `step_categorical_column_with_vocabulary_list()`, `step_crossed_column()`, `step_embedding_column()`, `step_indicator_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`, `steps`

Examples

```r
## Not run:
library(tfdatasets)
data(hearts)

hearts$thal <- as.integer(as.factor(hearts$thal)) - 1L
```
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ thal) %>%
  step_categorical_column_with_identity(thal, num_buckets = 5)

spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)

## End(Not run)

---

### step_categorical_column_with_vocabulary_file

*Creates a categorical column with vocabulary file*

---

#### Description

Use this function when the vocabulary of a categorical variable is written to a file.

#### Usage

```r
step_categorical_column_with_vocabulary_file(
  spec,
  ..., vocabulary_file, vocabulary_size = NULL,
  dtype = tf$string,
  default_value = NULL,
  num_oov_buckets = 0L
)
```

#### Arguments

- **spec**: A feature specification created with `feature_spec()`.
- **...**: Comma separated list of variable names to apply the step. `selectors` can also be used.
- **vocabulary_file**: The vocabulary file name.
- **vocabulary_size**: Number of the elements in the vocabulary. This must be no greater than length of `vocabulary_file`, if less than length, later values are ignored. If None, it is set to the length of `vocabulary_file`.
- **dtype**: The type of features. Only string and integer types are supported.
- **default_value**: The integer ID value to return for out-of-vocabulary feature values, defaults to `-1`. This can not be specified with a positive `num_oov_buckets`. 
**step_categorical_column_with_vocabulary_list**

num_oov_buckets

Non-negative integer, the number of out-of-vocabulary buckets. All out-of-vocabulary inputs will be assigned IDs in the range \( [\text{vocabulary\_size}, \text{vocabulary\_size}+\text{num\_oov\_buckets}) \) based on a hash of the input value. A positive num_oov_buckets can not be specified with default_value.

**Value**

A FeatureSpec object.

**See Also**

steps for a complete list of allowed steps.

Other Feature Spec Functions: dataset_use_spec(), feature_spec(), fit.FeatureSpec(), step_bucketized_column(), step_categorical_column_with_hash_bucket(), step_categorical_column_with_identity(), step_categorical_column_with_vocabulary_list(), step_crossed_column(), step_embedding_column(), step_indicator_column(), step_numeric_column(), step_remove_column(), step_shared_embeddings_column().

**Examples**

```r
## Not run:
library(tfdatasets)
data(hearts)
file <- tempfile()
writeLines(unique(hearts$thal), file)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ thal) %>%
  step_categorical_column_with_vocabulary_file(thal, vocabulary_file = file)

spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)

## End(Not run)
```
Usage

```r
step_categorical_column_with_vocabulary_list(
  spec,
  ..., 
  vocabulary_list = NULL,
  dtype = NULL,
  default_value = -1L,
  num_oov_buckets = 0L
)
```

Arguments

- `spec`: A feature specification created with `feature_spec()`.

- `...`: Comma separated list of variable names to apply the step. `selectors` can also be used.

- `vocabulary_list`: An ordered iterable defining the vocabulary. Each feature is mapped to the index of its value (if present) in `vocabulary_list`. Must be castable to `dtype`. If `NULL` the vocabulary will be defined as all unique values in the dataset provided when fitting the specification.

- `dtype`: The type of features. Only string and integer types are supported. If `NULL`, it will be inferred from `vocabulary_list`.

- `default_value`: The integer ID value to return for out-of-vocabulary feature values, defaults to -1. This can not be specified with a positive `num_oov_buckets`.

- `num_oov_buckets`: Non-negative integer, the number of out-of-vocabulary buckets. All out-of-vocabulary inputs will be assigned IDs in the range `[length(vocabulary_list), length(vocabulary_list)+num_oov_buckets)` based on a hash of the input value. A positive `num_oov_buckets` can not be specified with `default_value`.

Value

A `FeatureSpec` object.

See Also

- `steps` for a complete list of allowed steps.

Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step_bucketized_column()`, `step_categorical_column_with_hash_bucket()`, `step_categorical_column_with_identity()`, `step_categorical_column_with_vocabulary_file()`, `step_crossed_column()`, `step_embedding_column()`, `step_indicator_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`, `steps`

Examples

```r
## Not run:
library(tfdatasets)
data(hearts)
```
```
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ thal) %>%
  step_categorical_column_with_vocabulary_list(thal)

spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)
```

## End(Not run)

---

### step_crossed_column

**Creates crosses of categorical columns**

#### Description

Use this step to create crosses between categorical columns.

#### Usage

```
step_crossed_column(spec, ..., hash_bucket_size, hash_key = NULL)
```

#### Arguments

- **spec**: A feature specification created with `feature_spec()`.
- **...**: Comma separated list of variable names to apply the step. `selectors` can also be used.
- **hash_bucket_size**: An int > 1. The number of buckets.
- **hash_key**: (optional) Specify the hash_key that will be used by the FingerprintCat64 function to combine the crosses fingerprints on SparseCrossOp.

#### Value

A FeatureSpec object.

#### See Also

- `steps` for a complete list of allowed steps.
- Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step.bucketized_column()`, `step_categorical_column_with_hash_bucket()`, `step_categorical_column_with_identity()`, `step_categorical_column_with_vocabulary_file()`, `step_categorical_column_with_vocabulary_list()`, `step_embedding_column()`, `step_indicator_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`, `steps`
step_embedding_column

**Examples**

```r
## Not run:
library(tfdatasets)
data(hearts)
file <- tempfile()
writeLines(unique(hearts$thal), file)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ age) %>%
  step_numeric_column(age) %>%
  step_bucketized_column(age, boundaries = c(10, 20, 30))
spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)

## End(Not run)
```

---

**step_embedding_column**  *Creates embeddings columns*

**Description**

Use this step to create embeddings columns from categorical columns.

**Usage**

```r
step_embedding_column(
  spec,
  ..., 
  dimension = function(x) { as.integer(x^0.25) },
  combiner = "mean",
  initializer = NULL,
  ckpt_to_load_from = NULL,
  tensor_name_in_ckpt = NULL,
  max_norm = NULL,
  trainable = TRUE
)
```

**Arguments**

- `spec` A feature specification created with `feature_spec()`.
- `...` Comma separated list of variable names to apply the step. `selectors` can also be used.
- `dimension` An integer specifying dimension of the embedding, must be > 0. Can also be a function of the size of the vocabulary.
**combiner**  
A string specifying how to reduce if there are multiple entries in a single row. Currently 'mean', 'sqrtn' and 'sum' are supported, with 'mean' the default. 'sqrtn' often achieves good accuracy, in particular with bag-of-words columns. Each of this can be thought as example level normalizations on the column. For more information, see `tf.embedding_lookup_sparse`.

**initializer**  
A variable initializer function to be used in embedding variable initialization. If not specified, defaults to `tf.truncated_normal_initializer` with mean 0.0 and standard deviation `1/sqrt(dimension)`.

**ckpt_to_load_from**  
String representing checkpoint name/pattern from which to restore column weights. Required if `tensor_name_in_ckpt` is not NULL.

**tensor_name_in_ckpt**  
Name of the Tensor in `ckpt_to_load_from` from which to restore the column weights. Required if `ckpt_to_load_from` is not NULL.

**max_norm**  
If not NULL, embedding values are l2-normalized to this value.

**trainable**  
Whether or not the embedding is trainable. Default is TRUE.

**Value**  
a FeatureSpec object.

**See Also**  
`steps` for a complete list of allowed steps.

Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step.bucketized_column()`, `step.categorical_column_with_hash_bucket()`, `step.categorical_column_with_identity()`, `step.categorical_column_with_vocabulary_file()`, `step.categorical_column_with_vocabulary_list()`, `step.crossed_column()`, `step.indicator_column()`, `step.numeric_column()`, `step.remove_column()`, `step.shared_embeddings_column()`, `steps`

**Examples**

```r
## Not run:
library(tfdatasets)
data(hearts)
file <- tempfile()
writeLines(unique(hearts$thal), file)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ thal) %>%
  step_categorical_column_with_vocabulary_list(thal) %>%
  step_embedding_column(thal, dimension = 3)
spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)

## End(Not run)
```
step_indicator_column  Creates Indicator Columns

Description

Use this step to create indicator columns from categorical columns.

Usage

```r
step_indicator_column(spec, ...)
```

Arguments

- `spec` - A feature specification created with `feature_spec()`.
- `...` - Comma separated list of variable names to apply the step. `selectors` can also be used.

Value

-a FeatureSpec object.

See Also

- `steps` for a complete list of allowed steps.

Other Feature Spec Functions: `dataset_use_spec()`, `feature_spec()`, `fit.FeatureSpec()`, `step_bucketized_column()`, `step_categorical_column_with_hash_bucket()`, `step_categorical_column_with_identity()`, `step_categorical_column_with_vocabulary_file()`, `step_categorical_column_with_vocabulary_list()`, `step_crossed_column()`, `step_embedding_column()`, `step_numeric_column()`, `step_remove_column()`, `step_shared_embeddings_column()`, `steps`

Examples

```r
## Not run:
library(tfdatasets)
data(hearts)
file <- tempfile()
writeLines(unique(hearts$thal), file)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ thal) %>%
  step_categorical_column_with_vocabulary_list(thal) %>%
  step_indicator_column(thal)
spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)

## End(Not run)
```
step_numeric_column

Creates a numeric column specification

Description

step_numeric_column creates a numeric column specification. It can also be used to normalize numeric columns.

Usage

step_numeric_column(
  spec,
  ..., shape = 1L,
  default_value = NULL,
  dtype = tf$float32,
  normalizer_fn = NULL
)

Arguments

spec A feature specification created with feature_spec().

... Comma separated list of variable names to apply the step. selectors can also be used.

shape An iterable of integers specifies the shape of the Tensor. An integer can be given which means a single dimension Tensor with given width. The Tensor representing the column will have the shape of batch_size + shape.

default_value A single value compatible with dtype or an iterable of values compatible with dtype which the column takes on during tf.Example parsing if data is missing. A default value of NULL will cause tf.parse_example to fail if an example does not contain this column. If a single value is provided, the same value will be applied as the default value for every item. If an iterable of values is provided, the shape of the default_value should be equal to the given shape.

dtype defines the type of values. Default value is tf$float32. Must be a non-quantized, real integer or floating point type.

normalizer_fn If not NULL, a function that can be used to normalize the value of the tensor after default_value is applied for parsing. Normalizer function takes the input Tensor as its argument, and returns the output Tensor. (e.g. function(x) (x - 3.0) / 4.2). Please note that even though the most common use case of this function is normalization, it can be used for any kind of Tensorflow transformations. You can also a pre-made scaler, in this case a function will be created after fit.FeatureSpec is called on the feature specification.

Value

a FeatureSpec object.
step_remove_column

See Also

steps for a complete list of allowed steps.

Other Feature Spec Functions: dataset_use_spec(), feature_spec(), fit.FeatureSpec(),
step_bucketized_column(), step_categorical_column_with_hash_bucket(), step_categorical_column_with_identity(),
step_categorical_column_with_vocabulary_file(), step_categorical_column_with_vocabulary_list(),
step_crossed_column(), step_embedding_column(), step_indicator_column(), step_remove_column(),
step_shared_embeddings_column(), steps

Examples

## Not run:
library(tfdatasets)
data(hearts)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ age) %>%
  step_numeric_column(age, normalizer_fn = standard_scaler())

spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)

## End(Not run)

---

**step_remove_column**  Creates a step that can remove columns

Description

Removes features of the feature specification.

Usage

```r
step_remove_column(spec, ...)
```

Arguments

- `spec`: A feature specification created with `feature_spec()`.
- `...`: Comma separated list of variable names to apply the step. `selectors` can also be used.

Value

A FeatureSpec object.
See Also

steps for a complete list of allowed steps.

Other Feature Spec Functions: dataset_use_spec(), feature_spec(), fit.FeatureSpec(),
step_bucketized_column(), step_categorical_column_with_hash_bucket(), step_categorical_column_with_identity(),
step_categorical_column_with_vocabulary_file(), step_categorical_column_with_vocabulary_list(),
step_crossed_column(), step_embedding_column(), step_indicator_column(), step_numeric_column(),
step_shared_embeddings_column(), steps

Examples

```r
## Not run:
library(tfdatasets)
data(hearts)
hearts <- tensor_slices_dataset(hearts) %>% dataset_batch(32)

# use the formula interface
spec <- feature_spec(hearts, target ~ age) %>%
  step_numeric_column(age, normalizer_fn = scaler_standard()) %>%
  step_bucketized_column(age, boundaries = c(20, 50)) %>%
  step_remove_column(age)

spec_fit <- fit(spec)
final_dataset <- hearts %>% dataset_use_spec(spec_fit)
## End(Not run)
```

---

**step_shared_embeddings_column**

Creates shared embeddings for categorical columns

Description

This is similar to step_embedding_column, except that it produces a list of embedding columns that share the same embedding weights.

Usage

```r
step_shared_embeddings_column(
  spec,
  ...,
  dimension,
  combiner = "mean",
  initializer = NULL,
  shared_embedding_collection_name = NULL,
  ckpt_to_load_from = NULL,
  tensor_name_in_ckpt = NULL,
  max_norm = NULL,
```
trainable = TRUE

Arguments

spec | A feature specification created with feature_spec().
... | Comma separated list of variable names to apply the step. selectors can also be used.
dimension | An integer specifying dimension of the embedding, must be > 0. Can also be a function of the size of the vocabulary.
combiner | A string specifying how to reduce if there are multiple entries in a single row. Currently 'mean', 'sqrtn' and 'sum' are supported, with 'mean' the default. 'sqrtn' often achieves good accuracy, in particular with bag-of-words columns. Each of this can be thought as example level normalizations on the column. For more information, see tf.embedding_lookup_sparse.
initializer | A variable initializer function to be used in embedding variable initialization. If not specified, defaults to tf.truncated_normal_initializer with mean 0.0 and standard deviation 1/sqrt(dimension).
shared_embedding_collection_name | Optional collective name of these columns. If not given, a reasonable name will be chosen based on the names of categorical_columns.
ckpt_to_load_from | String representing checkpoint name/pattern from which to restore column weights. Required if tensor_name_in_ckpt is not NULL.
tensor_name_in_ckpt | Name of the Tensor in ckpt_to_load_from from which to restore the column weights. Required if ckpt_to_load_from is not NULL.
max_norm | If not NULL, embedding values are l2-normalized to this value.
trainable | Whether or not the embedding is trainable. Default is TRUE.

Value

a FeatureSpec object.

Note

Does not work in the eager mode.

See Also

steps for a complete list of allowed steps.

Other Feature Spec Functions: dataset_use_spec(), feature_spec(), fit.FeatureSpec(), step_bucketized_column(), step_categorical_column_with_hash_bucket(), step_categorical_column_with_id(), step_categorical_column_with_vocabulary_file(), step_categorical_column_with_vocabulary_list(), step_crossed_column(), step_embedding_column(), step_indicator_column(), step_numeric_column(), step_remove_column(), steps
tensors_dataset

Creates a dataset with a single element, comprising the given tensors.

Description

Creates a dataset with a single element, comprising the given tensors.

Usage

tensors_dataset(tensors)

Arguments

tensors A nested structure of tensors.

Value

A dataset.

See Also

Other tensor datasets: sparse_tensor_slices_dataset(), tensor_slices_dataset()

---

tensor_slices_dataset

Creates a dataset whose elements are slices of the given tensors.

Description

Creates a dataset whose elements are slices of the given tensors.

Usage

tensor_slices_dataset(tensors)

Arguments

tensors A nested structure of tensors, each having the same size in the 0th dimension.

Value

A dataset.

See Also

Other tensor datasets: sparse_tensor_slices_dataset(), tensors_dataset()
text_line_dataset  A dataset comprising lines from one or more text files.

Description
A dataset comprising lines from one or more text files.

Usage

text_line_dataset(
    filenames,
    compression_type = NULL,
    record_spec = NULL,
    parallel_records = NULL
)

Arguments

  filenames   String(s) specifying one or more filenames
  compression_type
      A string, one of: NULL (no compression), "ZLIB", or "GZIP".
  record_spec  (Optional) Specification used to decode delimitied text lines into records (see
delim_record_spec()).
  parallel_records
      (Optional) An integer, representing the number of records to decode in parallel.
      If not specified, records will be processed sequentially.

Value
A dataset

tfrecord_dataset  A dataset comprising records from one or more TFRecord files.

Description
A dataset comprising records from one or more TFRecord files.

Usage

tfrecord_dataset(
    filenames,
    compression_type = NULL,
    buffer_size = NULL,
    num_parallel_reads = NULL
)
until_out_of_range

Arguments

filenames  String(s) specifying one or more filenames
compression_type  A string, one of: NULL (no compression), "ZLIB", or "GZIP".
buffer_size  An integer representing the number of bytes in the read buffer. (0 means no buffering).
num_parallel_reads  An integer representing the number of files to read in parallel. Defaults to reading files sequentially.

Details

If the dataset encodes a set of TFExample instances, then they can be decoded into named records using the dataset_map() function (see example below).

Examples

## Not run:

# Creates a dataset that reads all of the examples from two files, and extracts
# the image and label features.
filenames <- c("/var/data/file1.tfrecord", "/var/data/file2.tfrecord")
dataset <- tfrecord_dataset(filenames) %>%
dataset_map(function(example_proto) {
  features <- list(
    image = tf$FixedLenFeature(shape(), tf$string, default_value = ""),
    label = tf$FixedLenFeature(shape(), tf$int32, default_value = 0L)
  )
  tf$parse_single_example(example_proto, features)
})

## End(Not run)

until_out_of_range  Execute code that traverses a dataset until an out of range condition occurs

Description

Execute code that traverses a dataset until an out of range condition occurs

Usage

until_out_of_range(expr)

out_of_range_handler(e)
### with_dataset

Execute code that traverses a dataset

#### Description

Execute code that traverses a dataset

#### Usage

```r
with_dataset(expr)
```

#### Arguments

- `expr` Expression to execute

#### Details

When a dataset iterator reaches the end, an out of range runtime error will occur. You can catch and ignore the error when it occurs by wrapping your iteration code in a call to `with_dataset()` (see the example below for an illustration).
Examples

```r
## Not run:
library(tfdatasets)
dataset <- text_line_dataset("mtcars.csv", record_spec = mtcars_spec) %>%
dataset_prepare(x = c(mpg, disp), y = cyl) %>%
dataset_batch(128) %>%
dataset_repeat(10)

iter <- make_iterator_one_shot(dataset)
next_batch <- iterator_get_next(iter)

with_dataset({
    while(TRUE) {
        batch <- sess$run(next_batch)
        # use batch$x and batch$y tensors
    }
})

## End(Not run)
```

zip_datasets

Creates a dataset by zipping together the given datasets.

Description

Merges datasets together into pairs or tuples that contain an element from each dataset.

Usage

`zip_datasets(...)`

Arguments

`...`  
Datasets to zip (or a single argument with a list or list of lists of datasets).

Value

A dataset
Index

* Dataset methods
  dataset_shard, 20
* Feature Spec Functions
  dataset_use_spec, 25
  feature_spec, 29
  fit.FeatureSpec, 31
  step_bucketized_column, 50
  step_categorical_column_with_hash_bucket, 51
  step_categorical_column_with_identity, 53
  step_categorical_column_with_vocabulary_file, 54
  step_categorical_column_with_vocabulary_list, 55
  step_crossed_column, 57
  step_embedding_column, 58
  step_indicator_column, 60
  step_numeric_column, 61
  step_remove_column, 62
  step_shared_embeddings_column, 63
  steps, 50
* Selectors
  all_nominal, 3
  all_numeric, 4
  has_type, 33
* dataset methods
  dataset_batch, 4
  dataset_cache, 5
  dataset_collect, 5
  dataset_concatenate, 6
  dataset_decode_delim, 7
  dataset_filter, 8
  dataset_interleave, 10
  dataset_map, 11
  dataset_map_and_batch, 12
  dataset_padded_batch, 14
  dataset_prefetch, 15
  dataset_prefetch_to_device, 16
  dataset_reduce, 18
  dataset_repeat, 19
  dataset_shuffle, 21
  dataset_shuffle_and_repeat, 22
  dataset_skip, 22
  dataset_take, 25
  dataset_window, 26
* datasets
  hearts, 33
  selectors, 48
* iterator functions
  iterator_get_next, 35
  iterator_initializer, 35
  iterator_make_initializer, 36
  iterator_string_handle, 36
  make_iterator, 38
* reading datasets
  until_out_of_range, 67
* scaler
  scaler_min_max, 47
  scaler_standard, 47
* tensor datasets
  sparse_tensor_slices_dataset, 48
  tensor_slices_dataset, 65
  tensors_dataset, 65
* text datasets
  text_line_dataset, 66
  all_nominal, 3, 4, 33
  all_nominal(), 48
  all_numeric, 3, 4, 33
  all_numeric(), 48
  compile(), 42
  contains(), 48
  csv_record_spec (delim_record_spec), 27
  cur_info_env (selectors), 48
  dataset_batch, 4, 5–7, 9–12, 15, 16, 18, 19, 21–23, 25, 27
matches(), 48
next_batch, 42
one_of(), 48
out_of_range_handler (until_out_of_range), 67
output_shapes (output_types), 43
output_shapes(), 8–12
output_types, 43
output_types(), 8–12
predict, 35
random_integer_dataset, 44
range_dataset, 44
read_files, 45
rlang::as_function(), 11, 12
sample_from_datasets, 46
scaler, 46, 47, 61
scaler_min_max, 46, 47, 47
selectors, 48, 50–54, 56–58, 60–62, 64
sparse_tensor_slices_dataset, 48, 65
sql_dataset (sql_record_spec), 49
sql_record_spec, 49
sqlite_dataset (sql_record_spec), 49
starts_with(), 48
step_bucketized_column, 26, 29, 31, 50, 50,
52, 53, 55–57, 59, 60, 62–64
step_bucketized_column(), 50
step_categorical_column_with_hash_bucket,
26, 29, 31, 50, 51, 53, 55–57, 59,
60, 62–64
step_categorical_column_with_hash_bucket(),
50
step_categorical_column_with_identity,
26, 29, 31, 50–52, 53, 55–57, 59, 60,
62–64
step_categorical_column_with_identity(),
50
step_categorical_column_with_vocabulary_file,
26, 29, 31, 50–53, 54, 56, 57, 59, 60,
62–64
step_categorical_column_with_vocabulary_file(),
50
step_categorical_column_with_vocabulary_list,
26, 29, 31, 50–53, 55, 55, 57, 59, 60,
62–64
step_crosed_column, 26, 29, 31, 50–53, 55,
56, 57, 59, 60, 62–64
step_crosed_column(), 50
step_embedding_column, 26, 29, 31, 50–53,
55–57, 58, 60, 62–64
step_embedding_column(), 50
step_indicator_column, 26, 29, 31, 50–53,
55–57, 59, 60, 62–64
step_indicator_column(), 50
step_numeric_column, 26, 29, 31, 46, 50–53,
55–57, 59, 60, 61, 63, 64
step_numeric_column(), 50
step_remove_column, 26, 29, 31, 50–53,
55–57, 59, 60, 62, 62, 64
step_remove_column(), 50
step_shared_embeddings_column, 26, 29,
31, 50–53, 55–57, 59, 60, 62, 63, 63
step_shared_embeddings_column(), 50
steps, 26, 29, 31, 33, 50, 51–53, 55–57, 59,
60, 62–64
tensor_slices_dataset, 49, 65, 65
tensors_dataset, 49, 63, 65
text_line_dataset, 66
text_line_dataset(), 34, 45
tfrecord_dataset, 66
tfrecord_dataset(), 34, 45
tidyselect::select_helpers(), 29
train, 35
tsv_record_spec (delim_record_spec), 27
until_out_of_range, 67
with_dataset, 68
zip_datasets, 69