Package ‘tfdatasets’

July 17, 2024

Type Package

Title Interface to 'TensorFlow' Datasets

Version 2.17.0

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 7.3.2

Suggests testthat, knitr, keras3, rsample, rmarkdown, Metrics, dplyr, progress, 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@posit.co>

Repository CRAN

Date/Publication 2024-07-16 22:30:01 UTC

Contents

all_nominal *Find all nominal variables.*

Description

Currently we only consider "string" type as nominal.

Usage

```
all_nominal()
```

See Also

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

all_numeric *Speciy all numeric variables.*

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()`

as_array_iterator *Convert tf_dataset to an iterator that yields R arrays.*

Description

Convert tf_dataset to an iterator that yields R arrays.

Usage

```
as_array_iterator(dataset)
```

Arguments

`dataset`	A tensorflow dataset

Value

An iterable. Use iterate() or iter_next() to access values from the iterator.

---

as_tensor.tensorflow.python.data.ops.dataset_ops.DatasetV2

Get the single element of the dataset.

Description

The function enables you to use a TF Dataset in a stateless "tensor-in tensor-out" expression, without creating an iterator. This facilitates the ease of data transformation on tensors using the optimized TF Dataset abstraction on top of them.

Usage

```
## S3 method for class 'tensorflow.python.data.ops.dataset_ops.DatasetV2'
as_tensor(x, ..., name = NULL)

## S3 method for class 'tensorflow.python.data.ops.dataset_ops.DatasetV2'
as.array(x, ...)
```

Arguments

x	A TF Dataset
...	passed on to tensorflow::as_tensor()
name	(Optional.) A name for the TensorFlow operation.

Details

For example, consider a preprocess_batch() which would take as an input a batch of raw features and returns the processed feature.

```
preprocess_one_case <- function(x) x + 100

preprocess_batch   <- function(raw_features) {
  batch_size <- dim(raw_features)[1]
  ds <- raw_features %>%
    tensor_slices_dataset() %>%
    dataset_map(preprocess_one_case, num_parallel_calls = batch_size) %>%
    dataset_batch(batch_size)
  as_tensor(ds)
}

raw_features <- array(seq(prod(4, 5)), c(4, 5))
preprocess_batch(raw_features)
```

In the above example, the batch of `raw_features` was converted to a TF Dataset. Next, each of the raw_feature cases in the batch was mapped using the preprocess_one_case and the processed features were grouped into a single batch. The final dataset contains only one element which is a batch of all the processed features.

Note: The dataset should contain only one element. Now, instead of creating an iterator for the dataset and retrieving the batch of features, the `as_tensor()` function is used to skip the iterator creation process and directly output the batch of features.

This can be particularly useful when your tensor transformations are expressed as TF Dataset operations, and you want to use those transformations while serving your model.

See Also

- `https://www.tensorflow.org/api_docs/python/tf/data/Dataset#get_single_element`

`choose_from_datasets` *Creates a dataset that deterministically chooses elements from datasets.*

Description

Creates a dataset that deterministically chooses elements from datasets.

Usage

```
choose_from_datasets(datasets, choice_dataset, stop_on_empty_dataset = TRUE)
```

Arguments

`datasets` A non-empty list of tf.data.Dataset objects with compatible structure.

`choice_dataset` A `tf.data.Dataset` of scalar `tf.int64` tensors between `0` and `length(datasets) - 1`.

`stop_on_empty_dataset`
If `TRUE`, selection stops if it encounters an empty dataset. If `FALSE`, it skips empty datasets. It is recommended to set it to `TRUE`. Otherwise, the selected elements start off as the user intends, but may change as input datasets become empty. This can be difficult to detect since the dataset starts off looking correct. Defaults to `TRUE`.

Value

Returns a dataset that interleaves elements from datasets according to the values of choice_dataset.

Examples

```
## Not run:
datasets <- list(tensors_dataset("foo") %>% dataset_repeat(),
                 tensors_dataset("bar") %>% dataset_repeat(),
                 tensors_dataset("baz") %>% dataset_repeat())

# Define a dataset containing `[0, 1, 2, 0, 1, 2, 0, 1, 2]`.
choice_dataset <- range_dataset(0, 3) %>% dataset_repeat(3)
result <- choose_from_datasets(datasets, choice_dataset)
result %>% as_array_iterator() %>% iterate(function(s) s$decode()) %>% print()
# [1] "foo" "bar" "baz" "foo" "bar" "baz" "foo" "bar" "baz"

## End(Not run)
```

dataset_batch *Combines consecutive elements of this dataset into batches.*

Description

The components of the resulting element will have an additional outer dimension, which will be `batch_size` (or `N %% batch_size` for the last element if `batch_size` does not divide the number of input elements `N` evenly and `drop_remainder` is `FALSE`). If your program depends on the batches having the same outer dimension, you should set the `drop_remainder` argument to `TRUE` to prevent the smaller batch from being produced.

Usage

```
dataset_batch(
  dataset,
  batch_size,
  drop_remainder = FALSE,
  num_parallel_calls = NULL,
  deterministic = NULL
)
```

Arguments

dataset	A dataset
batch_size	An integer, representing the number of consecutive elements of this dataset to combine in a single batch.
drop_remainder	(Optional.) A boolean, representing whether the last batch should be dropped in the case it has fewer than `batch_size` elements; the default behavior is not to drop the smaller batch.
num_parallel_calls	(Optional.) A scalar integer, representing the number of batches to compute asynchronously in parallel. If not specified, batches will be computed sequentially. If the value `tf$data$AUTOTUNE` is used, then the number of parallel calls is set dynamically based on available resources.

`deterministic` (Optional.) When `num_parallel_calls` is specified, if this boolean is specified (`TRUE` or `FALSE`), it controls the order in which the transformation produces elements. If set to `FALSE`, the transformation is allowed to yield elements out of order to trade determinism for performance. If not specified, the `tf.data.Options.experimental_deterministic` option (`TRUE` by default) controls the behavior. See `dataset_options()` for how to set dataset options.

Value

A dataset

Note

If your program requires data to have a statically known shape (e.g., when using XLA), you should use `drop_remainder=TRUE`. Without `drop_remainder=TRUE` the shape of the output dataset will have an unknown leading dimension due to the possibility of a smaller final batch.

See Also

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

dataset_bucket_by_sequence_length — *A transformation that buckets elements in a* `Dataset` *by length*

Description

A transformation that buckets elements in a `Dataset` by length

Usage

```
dataset_bucket_by_sequence_length(
  dataset,
  element_length_func,
  bucket_boundaries,
  bucket_batch_sizes,
  padded_shapes = NULL,
  padding_values = NULL,
  pad_to_bucket_boundary = FALSE,
  no_padding = FALSE,
  drop_remainder = FALSE,
  name = NULL
)
```

Arguments

`dataset` A `tf_dataset`

`element_length_func`
function from element in `Dataset` to `tf$int32`, determines the length of the element, which will determine the bucket it goes into.

`bucket_boundaries`
integers, upper length boundaries of the buckets.

`bucket_batch_sizes`
integers, batch size per bucket. Length should be `length(bucket_boundaries) + 1`.

`padded_shapes` Nested structure of `tf.TensorShape` (returned by `tensorflow::shape()`) to pass to `tf.data.Dataset.padded_batch`. If not provided, will use `dataset.output_shapes`, which will result in variable length dimensions being padded out to the maximum length in each batch.

`padding_values` Values to pad with, passed to `tf.data.Dataset.padded_batch`. Defaults to padding with 0.

`pad_to_bucket_boundary`
bool, if `FALSE`, will pad dimensions with unknown size to maximum length in batch. If `TRUE`, will pad dimensions with unknown size to bucket boundary minus 1 (i.e., the maximum length in each bucket), and caller must ensure that the source `Dataset` does not contain any elements with length longer than `max(bucket_boundaries)`.

`no_padding` boolean, indicates whether to pad the batch features (features need to be either of type `tf.sparse.SparseTensor` or of same shape).

`drop_remainder` (Optional.) A logical scalar, representing whether the last batch should be dropped in the case it has fewer than `batch_size` elements; the default behavior is not to drop the smaller batch.

`name` (Optional.) A name for the tf.data operation.

Details

Elements of the `Dataset` are grouped together by length and then are padded and batched.

This is useful for sequence tasks in which the elements have variable length. Grouping together elements that have similar lengths reduces the total fraction of padding in a batch which increases training step efficiency.

Below is an example to bucketize the input data to the 3 buckets "[0, 3), [3, 5), [5, Inf)" based on sequence length, with batch size 2.

See Also

- `https://www.tensorflow.org/api_docs/python/tf/data/Dataset#bucket_by_sequence_length`

Examples

```
## Not run:
dataset <- list(c(0),
                c(1, 2, 3, 4),
                c(5, 6, 7),
                c(7, 8, 9, 10, 11),
                c(13, 14, 15, 16, 17, 18, 19, 20),
                c(21, 22)) %>%
  lapply(as.array) %>% lapply(as_tensor, "int32") %>%
  lapply(tensors_dataset) %>%
  Reduce(dataset_concatenate, .)

dataset %>%
  dataset_bucket_by_sequence_length(
    element_length_func = function(elem) tf$shape(elem)[1],
    bucket_boundaries = c(3, 5),
    bucket_batch_sizes = c(2, 2, 2)
  ) %>%
  as_array_iterator() %>%
  iterate(print)
#      [,1] [,2] [,3] [,4]
# [1,]    1    2    3    4
# [2,]    5    6    7    0
#      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
# [1,]    7    8    9   10   11    0    0    0
# [2,]   13   14   15   16   17   18   19   20
#      [,1] [,2]
# [1,]    0    0
# [2,]   21   22

## End(Not run)
```

---

dataset_cache *Caches the elements in this dataset.*

Description

Caches the elements in this dataset.

Usage

```
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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `dataset_window()`

`dataset_collect` *Collects a dataset*

Description

Iterates throught 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

```
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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `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

```
dataset_concatenate(dataset, ...)
```

Arguments

`dataset, ...`	`tf_dataset`s to be concatenated

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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `dataset_window()`

`dataset_decode_delim` *Transform a dataset with delimted text lines into a dataset with named columns*

Description

Transform a dataset with delimted 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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `dataset_window()`

`dataset_enumerate` *Enumerates the elements of this dataset*

Description

Enumerates the elements of this dataset

Usage

```
dataset_enumerate(dataset, start = 0L)
```

Arguments

`dataset` A tensorflow dataset

`start` An integer (coerced to a `tf$int64` scalar `tf.Tensor`), representing the start value for enumeration.

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

```
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 comparisions (this is illustrated by the example below).

Value

A dataset composed of records that matched the predicate.

See Also

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

Examples

```
## 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_group_by_window` *Group windows of elements by key and reduce them*

Description

Group windows of elements by key and reduce them

Usage

```
dataset_group_by_window(
  dataset,
  key_func,
  reduce_func,
  window_size = NULL,
  window_size_func = NULL,
  name = NULL
)
```

Arguments

`dataset` a TF Dataset

`key_func` A function mapping a nested structure of tensors (having shapes and types defined by `self$output_shapes` and `self$output_types`) to a scalar `tf.int64` tensor.

`reduce_func` A function mapping a key and a dataset of up to `window_size` consecutive elements matching that key to another dataset.

`window_size` A `tf.int64` scalar `tf.Tensor`, representing the number of consecutive elements matching the same key to combine in a single batch, which will be passed to `reduce_func`. Mutually exclusive with `window_size_func`.

`window_size_func` A function mapping a key to a `tf.int64` scalar `tf.Tensor`, representing the number of consecutive elements matching the same key to combine in a single batch, which will be passed to `reduce_func`. Mutually exclusive with `window_size`.

`name` (Optional.) A name for the Tensorflow operation.

Details

This transformation maps each consecutive element in a dataset to a key using `key_func()` and groups the elements by key. It then applies `reduce_func()` to at most `window_size_func(key)` elements matching the same key. All except the final window for each key will contain `window_size_func(key)` elements; the final window may be smaller.

You may provide either a constant `window_size` or a window size determined by the key through `window_size_func`.

```
window_size <-  5
dataset <- range_dataset(to = 10) %>%
  dataset_group_by_window(
    key_func = function(x) x %% 2,
    reduce_func = function(key, ds) dataset_batch(ds, window_size),
    window_size = window_size
  )

it <- as_array_iterator(dataset)
while (!is.null(elem <- iter_next(it)))
  print(elem)
#> tf.Tensor([0 2 4 6 8], shape=(5), dtype=int64)
#> tf.Tensor([1 3 5 7 9], shape=(5), dtype=int64)
```

See Also

- https://www.tensorflow.org/api_docs/python/tf/data/Dataset#group_by_window

---

dataset_interleave *Maps map_func across this dataset, and interleaves the results*

Description

Maps map_func across this dataset, and interleaves the results

Usage

```
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(), dataset_map_and_batch(), dataset_padded_batch(dataset_prefetch(), dataset_prefetch_to_device(), dataset_reduce(), dataset_repeat(), dataset_shuffle(), dataset_shuffle_and_repeat(), dataset_skip(), dataset_take(), dataset_take_while(), dataset_window()

Examples

```
## 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):
c(1, 1, 1, 1,
  2, 2, 2, 2,
  1, 1,
  2, 2,
  3, 3, 3, 3,
  4, 4, 4, 4,
  3, 3,
  4, 4,
  5, 5, 5, 5,
  5, 5,
)


## End(Not run)
```

---

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()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `dataset_window()`

`dataset_map_and_batch` *Fused implementation of dataset_map() and dataset_batch()*

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`	(Optional.) A boolean, representing whether the last batch should be dropped in the case it has fewer than `batch_size` elements; the default behavior is not to drop the smaller batch.
`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()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `dataset_window()`

`dataset_options` *Get or Set Dataset Options*

Description

Get or Set Dataset Options

Usage

```
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

```
## 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

Combines consecutive elements of this dataset into padded batches.

Usage

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

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` (Optional.) A (nested) structure of `tf.TensorShape` (returned by `tensorflow::shape()`) or `tf$int64` 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 will be padded to the maximum size of that dimension in each batch. If unset, all dimensions of all components are padded to the maximum size in the batch. `padded_shapes` must be set if any component has an unknown rank.

`padding_values` (Optional.) A (nested) structure of scalar-shaped `tf.Tensor`, representing the padding values to use for the respective components. `NULL` represents that the (nested) structure should be padded with default values. Defaults are `0` for numeric types and the empty string `""` for string types. The `padding_values` should have the same (nested) structure as the input dataset. If `padding_values` is a single element and the input dataset has multiple components, then the same `padding_values` will be used to pad every component of the dataset. If `padding_values` is a scalar, then its value will be broadcasted to match the shape of each component.

`drop_remainder` (Optional.) A boolean scalar, representing whether the last batch should be dropped in the case it has fewer than `batch_size` elements; the default behavior is not to drop the smaller batch.

`name` (Optional.) A name for the tf.data operation. Requires tensorflow version >= 2.7.

Details

This transformation combines multiple consecutive elements of the input dataset into a single element.

Like `dataset_batch()`, the components of the resulting element will have an additional outer dimension, which will be `batch_size` (or `N %% batch_size` for the last element if `batch_size` does not divide the number of input elements `N` evenly and `drop_remainder` is `FALSE`). If your program depends on the batches having the same outer dimension, you should set the `drop_remainder` argument to `TRUE` to prevent the smaller batch from being produced.

Unlike dataset_batch(), the input elements to be batched may have different shapes, and this transformation will pad each component to the respective shape in padded_shapes. The padded_shapes argument determines the resulting shape for each dimension of each component in an output element:

- If the dimension is a constant, the component will be padded out to that length in that dimension.
- If the dimension is unknown, the component will be padded out to the maximum length of all elements in that dimension.

See also tf$data$experimental$dense_to_sparse_batch, which combines elements that may have different shapes into a tf$sparse$SparseTensor.

Value

A tf_dataset

See Also

- https://www.tensorflow.org/api_docs/python/tf/data/Dataset#padded_batch

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

Examples

```
## Not run:
A <- range_dataset(1, 5, dtype = tf$int32) %>%
  dataset_map(function(x) tf$fill(list(x), x))

# Pad to the smallest per-batch size that fits all elements.
B <- A %>% dataset_padded_batch(2)
B %>% as_array_iterator() %>% iterate(print)

# Pad to a fixed size.
C <- A %>% dataset_padded_batch(2, padded_shapes=5)
C %>% as_array_iterator() %>% iterate(print)

# Pad with a custom value.
D <- A %>% dataset_padded_batch(2, padded_shapes=5, padding_values = -1L)
D %>% as_array_iterator() %>% iterate(print)

# Pad with a single value and multiple components.
E <- zip_datasets(A, A) %>%  dataset_padded_batch(2, padding_values = -1L)
E %>% as_array_iterator() %>% iterate(print)

## End(Not run)
```

dataset_prefetch *Creates a Dataset that prefetches elements from this dataset.*

Description

Creates a Dataset that prefetches elements from this dataset.

Usage

```
dataset_prefetch(dataset, buffer_size = tf$data$AUTOTUNE)
```

Arguments

`dataset`	A dataset
`buffer_size`	An integer, representing the maximum number elements that will be buffered when prefetching.

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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `dataset_window()`

dataset_prefetch_to_device *A transformation that prefetches dataset values to the given* `device`

Description

A transformation that prefetches dataset values to the given `device`

Usage

```
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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `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

```
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` (Optional.) A boolean, representing whether the last batch should be dropped in the case it has fewer than `batch_size` elements; the default behavior is not to drop the smaller batch.

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

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

Arguments

`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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_prefetch_to_device()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `dataset_window()`

`dataset_rejection_resample` *A transformation that resamples a dataset to a target distribution.*

Description

A transformation that resamples a dataset to a target distribution.

Usage

```
dataset_rejection_resample(
  dataset,
  class_func,
  target_dist,
  initial_dist = NULL,
  seed = NULL,
  name = NULL
)
```

Arguments

`dataset`	A `tf.Dataset`
`class_func`	A function mapping an element of the input dataset to a scalar `tf.int32` tensor. Values should be in `[0, num_classes)`.
`target_dist`	A floating point type tensor, shaped `[num_classes]`.
`initial_dist`	(Optional.) A floating point type tensor, shaped `[num_classes]`. If not provided, the true class distribution is estimated live in a streaming fashion.
`seed`	(Optional.) Integer seed for the resampler.
`name`	(Optional.) A name for the tf.data operation.

Value

A `tf.Dataset`

Examples

```
## Not run:
initial_dist <- c(.5, .5)
target_dist <- c(.6, .4)
num_classes <- length(initial_dist)
num_samples <- 100000
data <- sample.int(num_classes, num_samples, prob = initial_dist, replace = TRUE)
dataset <- tensor_slices_dataset(data)
tally <- c(0, 0)
`add<-` <- function (x, value) x + value
# tfautograph::autograph({
#   for(i in dataset)
#     add(tally[as.numeric(i)]) <- 1
# })
dataset %>%
  as_array_iterator() %>%
  iterate(function(i) {
    add(tally[i]) <<- 1
  }, simplify = FALSE)
# The value of `tally` will be close to c(50000, 50000) as
# per the `initial_dist` distribution.
tally # c(50287, 49713)
tally <- c(0, 0)
dataset %>%
  dataset_rejection_resample(
    class_func = function(x) (x-1) %% 2,
    target_dist = target_dist,
    initial_dist = initial_dist
  ) %>%
  as_array_iterator() %>%
  iterate(function(element) {
    names(element) <- c("class_id", "i")
    add(tally[element$i]) <<- 1
  }, simplify = FALSE)
# The value of tally will be now be close to c(75000, 50000)
# thus satisfying the target_dist distribution.
tally # c(74822, 49921)

## End(Not run)
```

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(), dataset_map_and_batch(), dataset_padded_batch(), dataset_prefetch(), dataset_prefetch_to_device(), dataset_reduce(), dataset_shuffle(), dataset_shuffle_and_repeat(), dataset_skip(), dataset_take(), dataset_take_while(), 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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take()`, `dataset_take_while()`, `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

```
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(), dataset_map_and_batch(), dataset_padded_batch(), dataset_prefetch(), dataset_prefetch_to_device(), dataset_reduce(), dataset_repeat(), dataset_shuffle(), dataset_skip(), dataset_take(), dataset_take_while(), 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

```
dataset_skip(dataset, count)
```

Arguments

`dataset`	A dataset
`count`	An integer, representing the number of elements of this dataset that should be skipped to form the new dataset. If count is greater than the size of this dataset, the new dataset will contain no elements. If count is -1, skips the entire 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(), dataset_map_and_batch(), dataset_padded_batch(), dataset_prefetch(), dataset_prefetch_to_device(), dataset_reduce(), dataset_repeat(), dataset_shuffle(), dataset_shuffle_and_repeat(), dataset_take(), dataset_take_while(), dataset_window()

dataset_snapshot *Persist the output of a dataset*

Description

Persist the output of a dataset

Usage

```
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.

```
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:

```
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

```
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()`, `dataset_map_and_batch()`, `dataset_padded_batch()`, `dataset_prefetch()`, `dataset_prefetch_to_device()`, `dataset_reduce()`, `dataset_repeat()`, `dataset_shuffle()`, `dataset_shuffle_and_repeat()`, `dataset_skip()`, `dataset_take_while()`, `dataset_window()`

`dataset_take_while` *A transformation that stops dataset iteration based on a predicate.*

Description

A transformation that stops dataset iteration based on a predicate.

Usage

```
dataset_take_while(dataset, predicate, name = NULL)
```

Arguments

`dataset`	A TF dataset
`predicate`	A function that maps a nested structure of tensors (having shapes and types defined by `self$output_shapes` and `self$output_types`) to a scalar `tf.bool` tensor.
`name`	(Optional.) A name for the tf.data operation.

Details

Example usage:

```
range_dataset(from = 0, to = 10) %>%
  dataset_take_while( ~ .x < 5) %>%
  as_array_iterator() %>%
  iterate(simplify = FALSE) %>% str()
#> List of 5
#> $ : num 0
#> $ : num 1
#> $ : num 2
#> $ : num 3
#> $ : num 4
```

Value

A TF Dataset

See Also

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

`dataset_unbatch` *Unbatch a dataset*

Description

Splits elements of a dataset into multiple elements.

Usage

```
dataset_unbatch(dataset, name = NULL)
```

Arguments

`dataset`	A dataset
`name`	(Optional.) A name for the tf.data operation.

`dataset_unique` *A transformation that discards duplicate elements of a Dataset.*

Description

Use this transformation to produce a dataset that contains one instance of each unique element in the input (See example).

Usage

```
dataset_unique(dataset, name = NULL)
```

Arguments

`dataset`	A tf.Dataset.
`name`	(Optional.) A name for the tf.data operation.

Value

A tf.Dataset

Note

This transformation only supports datasets which fit into memory and have elements of either tf.int32, tf.int64 or tf.string type.

Examples

```
## Not run:
c(0, 37, 2, 37, 2, 1) %>% as_tensor("int32") %>%
  tensor_slices_dataset() %>%
  dataset_unique() %>%
  as_array_iterator() %>% iterate() %>% sort()
# [1]  0  1  2 37

## End(Not run)
```

`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

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

Arguments

`dataset`	A TensorFlow dataset.
`spec`	A feature specification created with `feature_spec()`.

Value

A TensorFlow dataset.

See Also

- `feature_spec()` to initialize the feature specification.
- `fit.FeatureSpec()` to create a tensorflow dataset prepared to modeling.

- steps to a list of all implemented steps.

Other Feature Spec Functions: 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(), 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)

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

## End(Not run)
```

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
size	representing the number of elements of the input dataset to combine into a window.
shift	epresenting 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(), dataset_map_and_batch(), dataset_padded_batch(), dataset_prefetch(), dataset_prefetch_to_device(), dataset_reduce(), dataset_repeat(), dataset_shuffle(), dataset_shuffle_and_repeat(), dataset_skip(), dataset_take(), dataset_take_while()

---

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

```
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 witin 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 datset. Note that if the underlying text file also includes column names in it's 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 explicitliy 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.

feature_spec *Creates 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

```
## 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 initilializing 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

```
## 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)
```

```
spec_fit

## End(Not run)
```

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

```
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

```
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

```
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 = reversable 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

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

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

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

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

DEPRECATED: Use keras3::layer_feature_space() instead.

Usage

```
layer_input_from_dataset(dataset)
```

Arguments

dataset	a TensorFlow dataset or a data.frame

Details

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

Value

a list of Keras input layers

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 + 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

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

## 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

```
## 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().

See Also

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

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

```
## 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

```
output_types(object)

output_shapes(object)
```

Arguments

`object`	A dataset or iterator

Value

`output_types()` returns the type of each component of an element of this object; `output_shapes()` returns the shape of each component of an element of this object

`random_integer_dataset` *Creates a* `Dataset` *of pseudorandom values*

Description

Creates a `Dataset` of pseudorandom values

Usage

```
random_integer_dataset(seed = NULL)
```

Arguments

`seed`	(Optional) If specified, the dataset produces a deterministic sequence of values.

Details

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

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

Description

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

Usage

```
range_dataset(from = 0, to = 0, by = 1, ..., dtype = tf$int64)
```

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

```
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` *Samples elements at random from the datasets in* `datasets`.

Description

Samples elements at random from the datasets in `datasets`.

Usage

```
sample_from_datasets(
  datasets,
  weights = NULL,
  seed = NULL,
  stop_on_empty_dataset = TRUE
)
```

Arguments

`datasets` A list ofobjects with compatible structure.

`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`.

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

`stop_on_empty_dataset` If `TRUE`, selection stops if it encounters an empty dataset. If `FALSE`, it skips empty datasets. It is recommended to set it to `TRUE`. Otherwise, the selected elements start off as the user intends, but may change as input datasets become empty. This can be difficult to detect since the dataset starts off looking correct. Defaults to `TRUE`.

Value

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

`scaler` *List of pre-made scalers*

Description

- scaler_standard: mean and standard deviation normalizer.
- scaler_min_max: min max normalizer

See Also

step_numeric_column

`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 *Selectors*

Description

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

Usage

```
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

```
sparse_tensor_slices_dataset(sparse_tensor)
```

Arguments

`sparse_tensor`	A `tf$SparseTensor`.

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

```
sql_record_spec(names, types)

sql_dataset(driver_name, data_source_name, query, record_spec)

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

steps *Steps for feature columns specification.*

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_ide 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

```
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

```
## 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

```
step_categorical_column_with_hash_bucket(
  spec,
  ...,
  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_vocabu` `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 ~ 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

```
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_voc` `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$thal <- as.integer(as.factor(hearts$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

```
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.

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 [vocabulary_size, vocabulary_size+num_oov_bu 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_ide 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)
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)
```

---

step_categorical_column_with_vocabulary_list
Creates a categorical column specification

Description

Creates a categorical column specification

Usage

```
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 `[lenght(vocabulary_list), length(vocabulary_`
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_ide`
`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

```
## 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_ide`
`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`

Examples

```
## 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 ambeddings columns from categorical columns.

Usage

```
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_ide` `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

```
## 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

```
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_ide 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

```
## 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.

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_ide` `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

```
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_ide
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

```
## 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

```
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_ide` `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 first 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 delimimted 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
)
```

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)
```

Arguments

`expr`	Expression to execute (will be executed multiple times until the condition occurs)
`e`	Error object

Details

When a dataset iterator reaches the end, an out of range runtime error will occur. This function will catch and ignore the error when it occurs.

Examples

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

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

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

## End(Not run)
```

`with_dataset` *Execute code that traverses a dataset*

Description

Execute code that traverses a dataset

Usage

```
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

```
## 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