
WTTE-RNN Documentation

Release

wtte-rnn

Jul 26, 2017

1	Examples	3
2	wtte	5
2.1	wtte package	5
2.1.1	Subpackages	5
2.1.1.1	wtte.objectives package	5
2.1.1.2	wtte.plots package	6
2.1.2	Submodules	6
2.1.3	wtte.data_generators module	6
2.1.4	wtte.pipelines module	7
2.1.5	wtte.transforms module	8
2.1.6	wtte.tte_util module	11
2.1.7	wtte.weibull module	12
2.1.8	wtte.wtte module	14
2.1.9	Module contents	16
3	Indices and tables	17
	Python Module Index	19

Contents:

CHAPTER 1

Examples

wtte package

Subpackages

wtte.objectives package

Submodules

wtte.objectives.tensorflow module

Objective functions for TensorFlow

```
wtte.objectives.tensorflow.betapenalty(b, location=10.0, growth=20.0, out-
                                     put_collection=(), name=None)
```

Returns a positive penalty term exploding when beta approaches location.

Adding this term to the loss may prevent overfitting and numerical instability of large values of beta (overconfidence). Remember that loss = -loglik+penalty

:param b:beta. Positive nonzero *Tensor*. :type b: *float32* or *float64*. :param output_collection:name of the collection to collect result of this op. :type output_collection: Tuple of Strings. :param String name: name of the operation. :return: A positive *Tensor* of same shape as *b* being a penalty term.

```
wtte.objectives.tensorflow.loglik_continuous(a, b, y_, u_, output_collection=(),
                                             name=None)
```

Returns element-wise Weibull censored log-likelihood.

Continuous weibull log-likelihood. loss=-loglikelihood. All input values must be of same type and shape.

:param a:alpha. Positive nonzero *Tensor*. :type a: *float32* or *float64*. :param b:beta. Positive nonzero *Tensor*. :type b: *float32* or *float64*. :param **y_**: time to event. Positive nonzero *Tensor* :type **y_**: *float32* or *float64*. :param **u_**: indicator. 0.0 if right censored, 1.0 if uncensored *Tensor* :type **u_**: *float32* or *float64*. :param output_collection:name of the collection to collect result of this op. :type output_collection: Tuple of Strings. :param String name: name of the operation. :return: A *Tensor* of log-likelihoods of same shape as a, b, **y_**, **u_**

```
wtte.objectives.tensorflow.loglik_discrete(a, b, y_, u_, output_collection=(),
                                           name=None)
```

Returns element-wise Weibull censored discrete log-likelihood.

Unit-discretized weibull log-likelihood. loss=-loglikelihood.

Note: All input values must be of same type and shape.

:param a:alpha. Positive nonzero *Tensor*. :type a: *float32* or *float64*. :param b:beta. Positive nonzero *Tensor*. :type b: *float32* or *float64*. :param **y_**: time to event. Positive nonzero *Tensor* :type **y_**: *float32* or *float64*. :param **u_**: indicator. 0.0 if right censored, 1.0 if uncensored *Tensor* :type **u_**: *float32* or *float64*. :param output_collection:name of the collection to collect result of this op. :type output_collection: Tuple of Strings. :param String name: name of the operation. :return: A *Tensor* of log-likelihoods of same shape as a, b, **y_**, **u_**.

Module contents

wtte.plots package

Submodules

wtte.plots.misc module

wtte.plots.weibull_contour module

wtte.plots.weibull_heatmap module

Module contents

Submodules

wtte.data_generators module

```
wtte.data_generators.generate_random_df(n_seqs=5, max_seq_length=10,
                                       unique_times=True, starttimes_min=0, start-
                                       times_max=0)
```

Generates random dataframe for testing.

For every sequence:

- 1.generate a random seq_length from [1,'max_seq_length']
- 2.generate the number of observations in the sequence from [1,seq_length]
- 3.randomly pick observation elapsed times from [1,'seq_length']
- 4.randomly pick a starttime [0,'starttimes_max']
- 5.Generate random data in the columns at these timesteps

This means that the only thing we know about a sequence is that it's at maximum *max_seq_length*

Parameters

- **df** – pandas dataframe with columns
 - *id*: integer

- *t*: integer
- *dt*: integer mimicking a global event time
- *t_ix*: integer contiguous user time count per id 0,1,2,..
- *t_elapsed*: integer the time from starttime per id ex 0,1,10,..
- *event*: 0 or 1
- *int_column*: random data
- *double_column*: random data
- **unique_times** (*int*) – whether there id,elapsed_time has only one obs. Default true
- **starttimes_min** (*int*) – integer to generate *dt* the absolute time
- **starttimes_max** (*int*) – integer to generate *dt* the absolute time

Return df A randomly generated dataframe.

`wtte.data_generators.generate_weibull(A, B, C, shape, discrete_time)`
Generate Weibull random variables.

Inputs can be scalar or broadcastable to *shape*.

Parameters

- **A** – Generating alpha
- **B** – Generating beta
- **C** – Censoring time

Returns

list of [*W*, *Y*, *U*]

- *W*: Actual TTE
- *Y*: Censored TTE
- *U*: non-censoring indicators

Return type ndarray

wtte.pipelines module

`wtte.pipelines.data_pipeline(df, id_col='id', abs_time_col='time_int', column_names=['event'], constant_cols=[], discrete_time=True, pad_between_steps=True, infer_seq_endtime=True, time_sec_interval=86400, timestep_aggregation_dict=None, drop_last_timestep=True)`

Preprocess dataframe and return it in padded tensor format.

This function is due to change alot.

- 1.Lowers the resolution of the (int) *abs_time_col* ex from epoch sec to epoch day by aggregating each column using *timestep_aggregation_dict*.
- 2.Padds out with zeros between timesteps and fills with value of *constant_cols*.
- 3.Infers or adds/fills an endtime.

This outputs tensor as is and leave it to downstream to define events, disalign targets and features (see *shift_discrete_padded_features*) and from that censoring-indicator and tte.

wtte.transforms module

```
wtte.transforms.df_join_in_endtime(df, constant_per_id_cols='id', abs_time_col='dt',
                                   abs_endtime=None, fill_zeros=False)
```

Join in NaN-rows at timestep of when we stopped observing non-events.

If we have a dataset consisting of events recorded until a fixed timestamp, that timestamp won't show up in the dataset (it's a non-event). By joining in a row with NaN data at *abs_endtime* we get a boundarytime for each sequence used for TTE-calculation and padding.

This is simpler in SQL where you join *on df.dt <= df_last_timestamp.dt*

Parameters

- **df** (*pandas.dataframe*) – Pandas dataframe
- **constant_per_id_cols** (*String or String list*) – identifying id and columns remaining constant per id×tep
- **abs_time_col** (*String*) – identifying the wall-clock column *df[abs_time_cols]*.
- **abs_endtime** (*None or same as df[abs_time_cols] values.*) – The time to join in. If None it's inferred.

:param bool fill_zeros : Whether to attempt to fill NaN with zeros after merge. :return pandas.dataframe df: pandas dataframe where each *id* has rows at the endtime.

```
wtte.transforms.df_to_array(df, column_names, nanpad_right=True, return_lists=False,
                             id_col='id', t_col='t')
```

Converts flat pandas df with cols *id,t,col1,col2,...* to array indexed *[id,t,col]*.

Parameters

- **df** (*Pandas dataframe*) – dataframe with columns:
 - *id*: Any type. A unique key for the sequence.
 - *t*: integer. If *t* is a non-contiguous int vec per id then steps in between t's are padded with zeros.
 - *columns* in *column_names* (String list)
- **nanpad_right** (*Boolean*) – If *True*, sequences are *np.nan*-padded to *max_seq_len*
- **return_lists** – Put every tensor in its own subarray
- **id_col** – string column name for *id*
- **t_col** – string column name for *t*

Return padded With *seqlen* the max value of *t* per id

- if *nanpad_right* & *!return_lists*: a numpy float array of dimension *[n_seqs,max_seqlen,n_features]*
- if *nanpad_right* & *return_lists*: *n_seqs* numpy float sub-arrays of dimension *[max_seqlen,n_features]*
- if *!nanpad_right* & *return_lists*: *n_seqs* numpy float sub-arrays of dimension *[seqlen,n_features]*

```
wtte.transforms.df_to_padded(df, column_names, id_col='id', t_col='t')
```

Pads pandas df to a numpy array of shape *[n_seqs,max_seqlen,n_features]*. see *df_to_array* for details

```
wtte.transforms.df_to_subarrays(df, column_names, id_col='id', t_col='t')
```

Pads pandas df to subarrays of shape *[n_seqs][seqlen[s],n_features]*. see *df_to_array* for details

`wtte.transforms.get_padded_seq_lengths(padded)`

Returns the number of (seq_len) non-nan elements per sequence.

Parameters `padded` – 2d or 3d tensor with dim 2 the time dimension

`wtte.transforms.left_pad_to_right_pad(padded)`

Change left padded to right padded.

`wtte.transforms.normalize_padded(padded, means=None, stds=None)`

Normalize by last dim of padded with means/stds or calculate them.

`wtte.transforms.padded_events_to_not_censored(events, discrete_time)`

`wtte.transforms.padded_events_to_not_censored_vectorized(events)`

(Legacy) calculates (non) right-censoring indicators from padded binary events

`wtte.transforms.padded_events_to_tte(events, discrete_time, t_elapsed=None)`

computes (right censored) time to event from padded binary events.

For details see `tte_util.get_tte`

Parameters

- **events** (*Array*) – Events array.
- **discrete_time** (*Boolean*) – *True* when applying discrete time scheme.
- **t_elapsed** (*Array*) – Elapsed time. Default value is *None*.

Return `Array time_to_events` Time-to-event tensor.

`wtte.transforms.padded_to_df(padded, column_names, dtypes, ids=None, id_col='id', t_col='t')`

Takes padded numpy array and converts nonzero entries to pandas dataframe row.

Inverse to `df_to_padded`.

Parameters

- **padded** (*Array*) – a numpy float array of dimension $[n_seqs, max_seqlen, n_features]$.
- **column_names** (*list*) – other columns to expand from df
- **dtypes** (*String list*) – the type to cast the float-entries to.
- **ids** – (optional) the ids to attach to each sequence
- **id_col** – Column where *id* is located. Default value is *id*.
- **t_col** – Column where *t* is located. Default value is *t*.

Return `df` Dataframe with Columns

- *id* (Integer) or the value of *ids*
- *t* (Integer).

A row in df is the *t*'th event for a *id* and has columns from *column_names*

`wtte.transforms.right_pad_to_left_pad(padded)`

Change right padded to left padded.

`wtte.transforms.shift_discrete_padded_features(padded, fill=0)`

Parameters

- **padded** – padded (np array): Array [batch,timestep,...]
- **fill** (*float*) – value to replace nans with.

For mathematical purity and to avoid confusion, in the Discrete case “2015-12-15” means an interval “2015-12-15 00.00 - 2015-12-15 23.59” i.e the data is accessible at “2015-12-15 23.59” (time when we query our database to do prediction about next day.)

In the continuous case “2015-12-15 23.59” means exactly at “2015-12-15 23.59: 00000000”.

t	dt	Event
0	2015-12-15 00.00-23.59	1
1	2015-12-16 00.00-23.59	1
2	2015-12-17 00.00-23.59	0

etc. In detail:

t	0	1	2	3	4	5	
event	1	1	0	0	1	?	
feature	?	1	1	0	0	1	
TTE	0	0	2	1	0	?	
Observed*	F	T	T	T	T	T	

t	dt	Event
0	2015-12-15 14.39	1
1	2015-12-16 16.11	1
2	2015-12-17 22.18	0

etc. In detail:

t	0	1	2	3	4	5	...
event	1	1	0	0	1	?	...
feature	1	1	0	0	1	?	...
TTE	1	3	2	1	?	?	...
Observed*	T	T	T	T	T	T	...

Observed = Do we have feature data at this time?

In the discrete case:

-> we need to roll data intent as features to the right.

-> First timestep typically has no measured features (and we may not even know until the end of the first interval if the sequence even exists!)

So there’s two options after rolling features to the right:

1.Fill in 0s at $t=0$. (‘shift_discrete_padded_features’)

- if (data -> event) this is (randomly) leaky (potentially safe)
- if (data <-> event) this exposes the truth (unsafe)!

2.Remove $t=0$ from target data

- (dont learn to predict about prospective customers first purchase)

Safest!

note: We never have target data for the last timestep after rolling.

Example: Customer has first click leading to day 0 so at day 1 we can use features about that click to predict time to purchase. Since click does not imply purchase we can predict time to purchase at step 0 (but with no feature data, ex using zeros as input).

wtte.tte_util module

`wtte.tte_util.carry_backward_if(x, is_true)`

Locomote backward $x[i]$ if $is_true[i]$. remain x untouched after last pos of truth.

Parameters

- **x** (*Array*) – object whos elements are to carry backward
- **is_true** (*Array*) – same length as x containing true/false boolean.

Return *Array x* backwarded object

`wtte.tte_util.carry_forward_if(x, is_true)`

Locomote forward $x[i]$ if $is_true[i]$. remain x untouched before first pos of truth.

Parameters

- **x** (*Array*) – object whos elements are to carry forward
- **is_true** (*Array*) – same length as x containing true/false boolean.

Return *Array x* forwarded object

`wtte.tte_util.get_is_not_censored(is_event, discrete_time=True)`

Calculates non-censoring indicator u for one vector.

Parameters

- **is_event** (*array*) – logical or numeric array indicating event.
- **discrete_time** (*Boolean*) – if *True*, last observation is conditionally censored.

`wtte.tte_util.get_tse(is_event, t_elapsed=None)`

Wrapper to calculate *Time Since Event* for input vector.

Inverse of tte. Safe to use as a feature. Always “continuous” method of calculating it. $tse > 0$ at time of event

(if discrete we dont know about the event yet, if continuous we know at record of event so superfluous to have $tse=0$)

$tse = 0$ at first step

Parameters

- **is_event** (*Array*) – Boolean array
- **t_elapsed** (*IntArray*) – None or integer array with same length as *is_event*.
– If none, it will use $t_elapsed.max() - t_elapsed[::-1]$.

reverse-indexing is pretty slow and ugly and not a helpful template for implementing in other languages.

`wtte.tte_util.get_tte(is_event, discrete_time, t_elapsed=None)`

wrapper to calculate *Time To Event* for input vector.

Parameters **discrete_time** (*Boolean*) – if *True*, use *get_tte_discrete*. If *False*, use *get_tte_continuous*.

`wtte.tte_util.get_tte_continuous(is_event, t_elapsed)`

Calculates time to (pointwise measured) next event over a vector.

Parameters

- **is_event** (*Array*) – Boolean array
- **t_elapsed** (*IntArray*) – integer array with same length as *is_event* that supports vectorized subtraction. If none, it will use $xrange(len(is_event))$

Return Array tte Time-to-event (continuous version)

TODO:: Should support discretely sampled, continuously measured TTE

`wtte.tte_util.get_tte_discrete(is_event, t_elapsed=None)`

Calculates discretely measured tte over a vector.

Parameters

- **is_event** (*Array*) – Boolean array
- **t_elapsed** (*IntArray*) – integer array with same length as *is_event*. If none, it will use `xrange(len(is_event))`

Return Array tte Time-to-event array (discrete version)

- **Caveats** `tte[i]` = numb. timesteps to timestep with event Step of event has `tte = 0` (event happened at time `[t,t+1)`) `tte[-1]=1` if no event (censored data)

`wtte.tte_util.roll_fun(x, size, fun=<function mean>, reverse=False)`

Like `cumsum` but with any function *fun*.

`wtte.tte_util.steps_since_true_minimal(is_event)`

(Time) since event over discrete (padded) event vector.

Parameters **is_event** (*Array*) – a vector of 0/1s or boolean

Return Array x steps since *is_event* was true

`wtte.tte_util.steps_to_true_minimal(is_event)`

(Time) to event for discrete (padded) event vector.

Parameters **is_event** (*Array*) – a vector of 0/1s or boolean

Return Array x steps until *is_event* is true

wtte.weibull module

Wrapper for Python Weibull functions

`wtte.weibull.cdf(t, a, b)`

Cumulative distribution function.

Parameters

- **t** – Value
- **a** – Alpha
- **b** – Beta

Returns $1 - np.exp(-np.power(t / a, b))$

`wtte.weibull.cmf(t, a, b)`

Cumulative Mass Function.

Parameters

- **t** – Value
- **a** – Alpha
- **b** – Beta

Returns $cdf(t + 1, a, b)$

class `wtte.weibull.conditional_excess`

Bases: `object`

Experimental class for conditional excess distribution.

The idea is to query s into the future after time t has passed without event. See thesis for details.

note: Note tested and may be incorrect!

cdf (t, s, a, b)

mean (t, a, b)

pdf (t, s, a, b)

quantile (t, a, b, p)

`wtte.weibull.continuous_loglik` ($t, a, b, u=1, equality=False$)

Continuous censored loglikelihood function.

Parameters **equality** (*bool*) – In ML we usually only care about the likelihood

with *proportionality*, removing terms not dependent on the parameters. If this is set to *True* we keep those terms.

`wtte.weibull.cumulative_hazard` (t, a, b)

Cumulative hazard

Parameters

- **t** – Value
- **a** – Alpha
- **b** – Beta

Returns $np.power(t/a, b)$

`wtte.weibull.discrete_loglik` ($t, a, b, u=1, equality=False$)

Discrete censored loglikelihood function.

Parameters **equality** (*bool*) – In ML we usually only care about the likelihood

with *proportionality*, removing terms not dependent on the parameters. If this is set to *True* we keep those terms.

`wtte.weibull.hazard` (t, a, b)

`wtte.weibull.mean` (a, b)

Continuous mean. Theoretically at most 1 step below discretized mean

$E[T] \leq E[T_d] + 1$ true for positive distributions.

Parameters

- **a** – Alpha
- **b** – Beta

Returns $a * \gamma(1.0 + 1.0/b)$

`wtte.weibull.mode` (a, b)

`wtte.weibull.pdf` (t, a, b)

Probability distribution function.

Parameters

- **t** – Value

- **a** – Alpha
- **b** – Beta

Returns $(b/a) * np.power(t/a, b-1) * np.exp(-np.power(t/a, b))$

`wtte.weibull.pmf(t, a, b)`

Probability mass function.

Parameters

- **t** – Value
- **a** – Alpha
- **b** – Beta

Returns $cdf(t+1.0, a, b) - cdf(t, a, b)$

`wtte.weibull.quantiles(a, b, p)`

Quantiles

Parameters

- **a** – Alpha
- **b** – Beta
- **p** –

Returns $a * np.power(-np.log(1.0 - p), 1.0/b)$

wtte.wtte module

class `wtte.wtte.WeightWatcher(per_batch=False, per_epoch=True)`

Bases: `keras.callbacks.Callback`

Keras Callback to keep an eye on output layer weights. (under development)

Usage: `weightwatcher = WeightWatcher(per_batch=True, per_epoch=False)`
`model.fit(..., callbacks=[weightwatcher]) weightwatcher.plot()`

append_metrics()

on_batch_begin(batch, logs={})

on_batch_end(batch, logs={})

on_epoch_begin(epoch, logs={})

on_epoch_end(epoch, logs={})

on_train_begin(logs={})

on_train_end(logs={})

plot()

class `wtte.wtte.loss(kind, reduce_loss=True, regularize=False, location=10.0, growth=20.0)`

Bases: `object`

Creates a keras WTTE-loss function. If regularize is called, a penalty is added creating ‘wall’ that beta do not want to pass over. This is not necessary with Sigmoid-beta activation.

- Usage

Example

Note: With masking keras needs to access each loss-contribution individually. Therefore we do not sum/reduce down to scalar (dim 1), instead return a tensor (with `reduce_loss=False`).

loss_function (*y_true*, *y_pred*)

class `wtte.wtte.output_activation` (*init_alpha=1.0*, *max_beta_value=5.0*)

Bases: `object`

Elementwise computation of alpha and regularized beta.

Object-Oriented Wrapper to `output_lambda` using `keras.layers.Activation`.

•Usage

```
wtte_activation = wtte.output_activation(init_alpha=1.,
                                         max_beta_value=4.0).activation

model.add(Dense(2))
model.add(Activation(wtte_activation))
```

activation (*ab*)

(Internal function) Activation wrapper

Parameters *ab* – original tensor with alpha and beta.

Return *ab* return of `output_lambda` with *init_alpha* and *max_beta_value*.

`wtte.wtte.output_lambda` (*x*, *init_alpha=1.0*, *max_beta_value=5.0*, *alpha_kernel_scalefactor=None*)

Elementwise (Lambda) computation of alpha and regularized beta.

•Alpha:

(activation) Exponential units seems to give faster training than the original papers softplus units. Makes sense due to logarithmic effect of change in alpha. (initialization) To get faster training and fewer exploding gradients, initialize alpha to be around its scale when beta is around 1.0, approx the expected value/mean of training tte. Because we're lazy we want the correct scale of output built into the model so initialize implicitly; multiply assumed $\exp(0)=1$ by scale factor *init_alpha*.

•Beta:

(activation) We want slow changes when $\beta \rightarrow 0$ so Softplus made sense in the original paper but we get similar effect with sigmoid. It also has nice features. (regularization) Use *max_beta_value* to implicitly regularize the model (initialization) Fixed to begin moving slowly around 1.0

•Usage

```
model.add(TimeDistributed(Dense(2)))
model.add(Lambda(wtte.output_lambda, arguments={"init_alpha":init_alpha,
                                                "max_beta_value":2.0
                                                }))
```

Parameters

- *x* (*Array*) – tensor with last dimension having length 2 with $x[...0] = \alpha$, $x[...1] = \beta$
- **init_alpha** (*Integer*) – initial value of *alpha*. Default value is 1.0.
- **max_beta_value** (*Integer*) – maximum beta value. Default value is 5.0.
- **max_alpha_value** (*Integer*) – maximum alpha value. Default is *None*.

Return x A positive *Tensor* of same shape as input

Return type Array

Module contents

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

W

- `wtte`, [16](#)
- `wtte.data_generators`, [6](#)
- `wtte.objectives`, [6](#)
- `wtte.objectives.tensorflow`, [5](#)
- `wtte.pipelines`, [7](#)
- `wtte.plots`, [6](#)
- `wtte.transforms`, [8](#)
- `wtte.tte_util`, [11](#)
- `wtte.weibull`, [12](#)
- `wtte.wtte`, [14](#)

A

activation() (wtte.wtte.output_activation method), 15
 append_metrics() (wtte.wtte.WeightWatcher method), 14

B

betapenalty() (in module wtte.objectives.tensorflow), 5

C

carry_backward_if() (in module wtte.tte_util), 11
 carry_forward_if() (in module wtte.tte_util), 11
 cdf() (in module wtte.weibull), 12
 cdf() (wtte.weibull.conditional_excess method), 13
 cmf() (in module wtte.weibull), 12
 conditional_excess (class in wtte.weibull), 13
 continuous_loglik() (in module wtte.weibull), 13
 cumulative_hazard() (in module wtte.weibull), 13

D

data_pipeline() (in module wtte.pipelines), 7
 df_join_in_endtime() (in module wtte.transforms), 8
 df_to_array() (in module wtte.transforms), 8
 df_to_padded() (in module wtte.transforms), 8
 df_to_subarrays() (in module wtte.transforms), 8
 discrete_loglik() (in module wtte.weibull), 13

G

generate_random_df() (in module wtte.data_generators), 6
 generate_weibull() (in module wtte.data_generators), 7
 get_is_not_censored() (in module wtte.tte_util), 11
 get_padded_seq_lengths() (in module wtte.transforms), 8
 get_tse() (in module wtte.tte_util), 11
 get_tte() (in module wtte.tte_util), 11
 get_tte_continuous() (in module wtte.tte_util), 11
 get_tte_discrete() (in module wtte.tte_util), 12

H

hazard() (in module wtte.weibull), 13

L

left_pad_to_right_pad() (in module wtte.transforms), 9
 loglik_continuous() (in module wtte.objectives.tensorflow), 5
 loglik_discrete() (in module wtte.objectives.tensorflow), 6
 loss (class in wtte.wtte), 14
 loss_function() (wtte.wtte.loss method), 15

M

mean() (in module wtte.weibull), 13
 mean() (wtte.weibull.conditional_excess method), 13
 mode() (in module wtte.weibull), 13

N

normalize_padded() (in module wtte.transforms), 9

O

on_batch_begin() (wtte.wtte.WeightWatcher method), 14
 on_batch_end() (wtte.wtte.WeightWatcher method), 14
 on_epoch_begin() (wtte.wtte.WeightWatcher method), 14
 on_epoch_end() (wtte.wtte.WeightWatcher method), 14
 on_train_begin() (wtte.wtte.WeightWatcher method), 14
 on_train_end() (wtte.wtte.WeightWatcher method), 14
 output_activation (class in wtte.wtte), 15
 output_lambda() (in module wtte.wtte), 15

P

padded_events_to_not_censored() (in module wtte.transforms), 9
 padded_events_to_not_censored_vectorized() (in module wtte.transforms), 9
 padded_events_to_tte() (in module wtte.transforms), 9
 padded_to_df() (in module wtte.transforms), 9
 pdf() (in module wtte.weibull), 13
 pdf() (wtte.weibull.conditional_excess method), 13
 plot() (wtte.wtte.WeightWatcher method), 14
 pmf() (in module wtte.weibull), 14

Q

`quantile()` (`wtte.weibull.conditional_excess` method), [13](#)
`quantiles()` (in module `wtte.weibull`), [14](#)

R

`right_pad_to_left_pad()` (in module `wtte.transforms`), [9](#)
`roll_fun()` (in module `wtte.tte_util`), [12](#)

S

`shift_discrete_padded_features()` (in module `wtte.transforms`), [9](#)
`steps_since_true_minimal()` (in module `wtte.tte_util`), [12](#)
`steps_to_true_minimal()` (in module `wtte.tte_util`), [12](#)

W

`WeightWatcher` (class in `wtte.wtte`), [14](#)
`wtte` (module), [16](#)
`wtte.data_generators` (module), [6](#)
`wtte.objectives` (module), [6](#)
`wtte.objectives.tensorflow` (module), [5](#)
`wtte.pipelines` (module), [7](#)
`wtte.plots` (module), [6](#)
`wtte.transforms` (module), [8](#)
`wtte.tte_util` (module), [11](#)
`wtte.weibull` (module), [12](#)
`wtte.wtte` (module), [14](#)