Package ‘tabnet’

December 5, 2023

Title  Fit 'TabNet' Models for Classification and Regression
Version  0.5.0
Description  Implements the 'TabNet' model by Sercan O. Arik et al. (2019) <arXiv:1908.07442>
with 'Coherent Hierarchical Multi-label Classification Networks' by Giunchiglia et al. <arXiv:2010.10151> and provides a consistent interface for fitting and creating predictions. It's also fully compatible with the 'tidymodels' ecosystem.
License  MIT + file LICENSE
Encoding  UTF-8
RoxygenNote  7.2.3
URL  https://mlverse.github.io/tabnet/,
     https://github.com/mlverse/tabnet
BugReports  https://github.com/mlverse/tabnet/issues
Depends  R (>= 3.6)
Imports  torch (>= 0.4.0), hardhat (>= 1.3.0), magrittr, progress, rlang, methods, dplyr, tibble, tidyr, coro, vctrs, zeallot
Suggests  testthat (>= 3.0.0), data.tree, Matrix, modeldata, recipes, rsample, parsnip, dials, withr, knitr, rmarkdown, vip, tidyverse, ggplot2, purrr, stringr, tune, workflows, yardstick
VignetteBuilder  knitr
Config/testthat/edition  3
Config/testthat/parallel  false
Config/testthat/start-first  interface, explain, params
NeedsCompilation  no
Author  Daniel Falbel [aut],
        RStudio [cph],
        Christophe Regouby [cre, ctb],
        Egill Fridgeirsson [ctb],
        Philipp Haarmeyer [ctb],
        Sven Verweij [ctb] (<https://orcid.org/0000-0002-5573-3952>)
Maintainer  Christophe Regouby <christophe.regouby@free.fr>
Description

Plot tabnet_explain mask importance heatmap

Usage

```r
autoplot.tabnet_explain(
  object,
  type = c("mask_agg", "steps"),
  quantile = 1,
  ...
)
```

Arguments

- **object**: A tabnet_explain object as a result of `tabnet_explain()`.
- **type**: a character value. Either "mask_agg" the default, for a single heatmap of aggregated mask importance per predictor along the dataset, or "steps" for one heatmap at each mask step.
- **quantile**: numerical value between 0 and 1. Provides quantile clipping of the mask values not used.
Details

Plot the tabnet_explain object mask importance per variable along the predicted dataset. type="mask_agg" output a single heatmap of mask aggregated values, type="steps" provides a plot faceted along the n_steps mask present in the model. quantile=.995 may be used for strong outlier clipping, in order to better highlight low values. quantile=1, the default, do not clip any values.

Value

A ggplot object.

Examples

```r
library(ggplot2)
data("attrition", package = "modeldata")

## Single-outcome binary classification of 'Attrition' in 'attrition' dataset
attrition_fit <- tabnet_fit(Attrition ~., data=attrition, epoch=11)
attrition_explain <- tabnet_explain(attrition_fit, attrition)
# Plot the model aggregated mask interpretation heatmap
autoplot(attrition_explain)

## Multi-outcome regression on 'Sale_Price' and 'Pool_Area' in 'ames' dataset,
data("ames", package = "modeldata")
ids <- sample(nrow(ames), 256)
x <- ames[ids,-which(names(ames) %in% c("Sale_Price", "Pool_Area"))]
y <- ames[ids, c("Sale_Price", "Pool_Area")]
ames_fit <- tabnet_fit(x, y, epochs = 5, verbose=TRUE)
ames_explain <- tabnet_explain(ames_fit, x)
autoplot(ames_explain, quantile = 0.99)
```

autplot.tabnet_fit  
Plot tabnet_fit model loss along epochs

Description

Plot tabnet_fit model loss along epochs

Usage

```r
autplot.tabnet_fit(object, ...)
autplot.tabnet_pretrain(object, ...)
```

Arguments

- `object` A tabnet_fit or tabnet_pretrain object as a result of `tabnet_fit()` or `tabnet_pretrain()`.
- `...` not used.
Details

Plot the training loss along epochs, and validation loss along epochs if any. A dot is added on epochs where model snapshot is available, helping the choice of \texttt{from\_epoch} value for later model training resume.

Value

A \texttt{ggplot} object.

Examples

```r
library(ggplot2)
data("attrition", package = "modeldata")
attrition_fit <- tabnet_fit(Attrition ~., data=attrition, valid_split=0.2, epoch=11)

# Plot the model loss over epochs
autoplot(attrition_fit)
```

---

\texttt{check\_compliant\_node} \quad \textit{Check that Node object names are compliant}

Description

Check that Node object names are compliant

Usage

\texttt{check\_compliant\_node(node)}

Arguments

\begin{itemize}
  \item \texttt{node} \quad the Node object, or a dataframe ready to be parsed by \texttt{data.tree::as.Node()}
\end{itemize}

Value

\begin{itemize}
  \item \texttt{node} if it is compliant, else an Error with the column names to fix
\end{itemize}

Examples

```r
library(dplyr)
library(data.tree)
data(starwars)
starwars_tree <- starwars %>%
  mutate(pathString = paste("tree", species, homeworld, `name`, sep = "/"))

# pre as.Node() check
```
decision_width

try(check_compliant_node(starwars_tree))

# post as.Node() check
check_compliant_node(as.Node(starwars_tree))

decision_width

Parameters for the tabnet model

Description

Parameters for the tabnet model

Usage

decision_width(range = c(8L, 64L), trans = NULL)
attention_width(range = c(8L, 64L), trans = NULL)
num_steps(range = c(3L, 10L), trans = NULL)
feature_reusage(range = c(1, 2), trans = NULL)
num_independent(range = c(1L, 5L), trans = NULL)
num_shared(range = c(1L, 5L), trans = NULL)
momentum(range = c(0.01, 0.4), trans = NULL)
mask_type(values = c("sparsemax", "entmax"))

Arguments

range the default range for the parameter value
trans whether to apply a transformation to the parameter
values possible values for factor parameters

These functions are used with tune grid functions to generate candidates.

Value

A dials parameter to be used when tuning TabNet models.
nn_prune_head.tabnet_fit

*Prune top layer(s) of a tabnet network*

**Description**

Prune head_size last layers of a tabnet network in order to use the pruned module as a sequential embedding module.

**Usage**

nn_prune_head.tabnet_fit(x, head_size)

nn_prune_head.tabnet_pretrain(x, head_size)

**Arguments**

- **x**: nn_network to prune
- **head_size**: number of nn_layers to prune, should be less than 2

**Value**

a tabnet network with the top nn_layer removed

---

node_to_df

*Turn a Node object into predictor and outcome.*

**Description**

Turn a Node object into predictor and outcome.

**Usage**

node_to_df(x, drop_last_level = TRUE)

**Arguments**

- **x**: Node object
- **drop_last_level**: TRUE unused

**Value**

a named list of x and y, being respectively the predictor data-frame and the outcomes data-frame, as expected inputs for `hardhat::mold()` function.
Examples

```r
library(dplyr)
library(data.tree)
data(starwars)
starwars_tree <- starwars %>%
    mutate(pathString = paste("tree", species, homeworld, "name", sep = "/")) %>%
as.Node()
node_to_df(starwars_tree)$x %>% head()
node_to_df(starwars_tree)$y %>% head()
```

---

**Description**

Parsnip compatible tabnet model

**Usage**

```r
tabnet(
    mode = "unknown",
    epochs = NULL,
    penalty = NULL,
    batch_size = NULL,
    learn_rate = NULL,
    decision_width = NULL,
    attention_width = NULL,
    num_steps = NULL,
    feature_reusage = NULL,
    virtual_batch_size = NULL,
    num_independent = NULL,
    num_shared = NULL,
    momentum = NULL
)
```

**Arguments**

- **mode**    A single character string for the type of model. Possible values for this model are "unknown", "regression", or "classification".
- **epochs**  (int) Number of training epochs.
- **penalty** This is the extra sparsity loss coefficient as proposed in the original paper. The bigger this coefficient is, the sparser your model will be in terms of feature selection. Depending on the difficulty of your problem, reducing this value could help (default 1e-3).
batch_size (int) Number of examples per batch, large batch sizes are recommended. (default: 1024^2)
learn_rate initial learning rate for the optimizer.
decision_width (int) Width of the decision prediction layer. Bigger values gives more capacity to the model with the risk of overfitting. Values typically range from 8 to 64.
attention_width (int) Width of the attention embedding for each mask. According to the paper n_d = n_a is usually a good choice. (default=8)
num_steps (int) Number of steps in the architecture (usually between 3 and 10)
feature_reusage (float) This is the coefficient for feature reusage in the masks. A value close to 1 will make mask selection least correlated between layers. Values range from 1.0 to 2.0.
virtual_batch_size (int) Size of the mini batches used for "Ghost Batch Normalization" (default=256^2)
num_independent Number of independent Gated Linear Units layers at each step of the encoder. Usual values range from 1 to 5.
num_shared Number of shared Gated Linear Units at each step of the decoder. Usual values range from 1 to 5
momentum Momentum for batch normalization, typically ranges from 0.01 to 0.4 (default=0.02)

Value
A TabNet parsnip instance. It can be used to fit tabnet models using parsnip machinery.

Threading
TabNet uses torch as its backend for computation and torch uses all available threads by default. You can control the number of threads used by torch with:

```
torch::torch_set_num_threads(1)
torch::torch_set_num_interop_threads(1)
```

See Also
tabnet_fit

Examples

```r
library(parsnip)
data("ames", package = "modeldata")
model <- tabnet() %>%
  set_mode("regression") %>%
  set_engine("torch") %>%
  fit(Sale_Price ~ ., data = ames)
```
tabnet_config  

Configuration for TabNet models

Description

Configuration for TabNet models

Usage

tabnet_config(
    batch_size = 1024^2,
    penalty = 0.001,
    clip_value = NULL,
    loss = "auto",
    epochs = 5,
    drop_last = FALSE,
    decision_width = NULL,
    attention_width = NULL,
    num_steps = 3,
    feature_reusage = 1.3,
    mask_type = "sparsemax",
    virtual_batch_size = 256^2,
    valid_split = 0,
    learn_rate = 0.02,
    optimizer = "adam",
    lr_scheduler = NULL,
    lr_decay = 0.1,
    step_size = 30,
    checkpoint_epochs = 10,
    cat_emb_dim = 1,
    num_independent = 2,
    num_shared = 2,
    num_independent_decoder = 1,
    num_shared_decoder = 1,
    momentum = 0.02,
    pretraining_ratio = 0.5,
    verbose = FALSE,
    device = "auto",
    importance_sample_size = NULL,
    early_stopping_monitor = "auto",
    early_stopping_tolerance = 0,
    early_stopping_patience = 0L,
    num_workers = 0L,
    skip_importance = FALSE
)
)
Arguments

**batch_size**  (int) Number of examples per batch, large batch sizes are recommended. (default: $1024^2$)

**penalty**  This is the extra sparsity loss coefficient as proposed in the original paper. The bigger this coefficient is, the sparser your model will be in terms of feature selection. Depending on the difficulty of your problem, reducing this value could help (default 1e-3).

**clip_value**  If a float is given this will clip the gradient at clip_value. Pass NULL to not clip.

**loss**  (character or function) Loss function for training (default to mse for regression and cross entropy for classification)

**epochs**  (int) Number of training epochs.

**drop_last**  (logical) Whether to drop last batch if not complete during training

**decision_width**  (int) Width of the decision prediction layer. Bigger values gives more capacity to the model with the risk of overfitting. Values typically range from 8 to 64.

**attention_width**  (int) Width of the attention embedding for each mask. According to the paper $n_d = n_a$ is usually a good choice. (default=8)

**num_steps**  (int) Number of steps in the architecture (usually between 3 and 10)

**feature_reusage**  (float) This is the coefficient for feature reusage in the masks. A value close to 1 will make mask selection least correlated between layers. Values range from 1.0 to 2.0.

**mask_type**  (character) Final layer of feature selector in the attentive_transformer block, either "sparsemax" or "entmax". Defaults to "sparsemax".

**virtual_batch_size**  (int) Size of the mini batches used for "Ghost Batch Normalization" (default=$256^2$)

**valid_split**  ([0, 1)) The fraction of the dataset used for validation. (default = 0 means no split)

**learn_rate**  initial learning rate for the optimizer.

**optimizer**  the optimization method. currently only 'adam' is supported, you can also pass any torch optimizer function.

**lr_scheduler**  if NULL, no learning rate decay is used. If "step" decays the learning rate by lr_decay every step_size epochs. If "reduce_on_plateau" decays the learning rate by lr_decay when no improvement after step_size epochs. It can also be a torch::lr_scheduler function that only takes the optimizer as parameter. The step method is called once per epoch.

**lr_decay**  multiplies the initial learning rate by lr_decay every step_size epochs. Unused if lr_scheduler is a torch::lr_scheduler or NULL.

**step_size**  the learning rate scheduler step size. Unused if lr_scheduler is a torch::lr_scheduler or NULL.

**checkpoint_epochs**  checkpoint model weights and architecture every checkpoint_epochs. (default is 10). This may cause large memory usage. Use 0 to disable checkpoints.
tabnet_config

**cat_emb_dim** 
Size of the embedding of categorical features. If int, all categorical features will have same embedding size, if list of int, every corresponding feature will have specific embedding size.

**num_independent** 
Number of independent Gated Linear Units layers at each step of the encoder. Usual values range from 1 to 5.

**num_shared** 
Number of shared Gated Linear Units at each step of the encoder. Usual values range from 1 to 5.

**num_independent_decoder**
For pretraining, number of independent Gated Linear Units layers. Usual values range from 1 to 5.

**num_shared_decoder**
For pretraining, number of shared Gated Linear Units at each step of the decoder. Usual values range from 1 to 5.

**momentum** 
Momentum for batch normalization, typically ranges from 0.01 to 0.4 (default=0.02)

**pretraining_ratio** 
Ratio of features to mask for reconstruction during pretraining. Ranges from 0 to 1 (default=0.5)

**verbose** 
(logical) Whether to print progress and loss values during training.

**device** 
the device to use for training. "cpu" or "cuda". The default ("auto") uses "cuda" if it's available, otherwise uses "cpu".

**importance_sample_size** 
Sample of the dataset to compute importance metrics. If the dataset is larger than 1e5 obs we will use a sample of size 1e5 and display a warning.

**early_stopping_monitor** 
Metric to monitor for early_stopping. One of "valid_loss", "train_loss" or "auto" (defaults to "auto").

**early_stopping_tolerance** 
Minimum relative improvement to reset the patience counter. 0.01 for 1% tolerance (default 0)

**early_stopping_patience** 
Number of epochs without improving until stopping training. (default=5)

**num_workers** 
(int, optional): how many subprocesses to use for data loading. 0 means that the data will be loaded in the main process. (default: 0)

**skip_importance** 
if feature importance calculation should be skipped (default: FALSE)

**Value**
A named list with all hyperparameters of the TabNet implementation.
tabnet_explain

Interpretation metrics from a TabNet model

Description

Interpretation metrics from a TabNet model

Usage

```
tabnet_explain(object, new_data)
```

## Default S3 method:
tabnet_explain(object, new_data)

## S3 method for class 'tabnet_fit'
tabnet_explain(object, new_data)

## S3 method for class 'tabnet_pretrain'
tabnet_explain(object, new_data)

## S3 method for class 'model_fit'
tabnet_explain(object, new_data)

Arguments

- `object` a TabNet fit object
- `new_data` a data.frame to obtain interpretation metrics.

Value

Returns a list with

- `M_explain`: the aggregated feature importance masks as detailed in TabNet's paper.
- `masks`: a list containing the masks for each step.

Examples

```
set.seed(2021)

n <- 1000
x <- data.frame(
  x = rnorm(n),
  y = rnorm(n),
  z = rnorm(n)
)
```
y <- x$x

fit <- tabnet_fit(x, y, epochs = 20,
                   num_steps = 1,
                   batch_size = 512,
                   attention_width = 1,
                   num_shared = 1,
                   num_independent = 1)

ex <- tabnet_explain(fit, x)

tabnet_fit

Description

Fits the TabNet: Attentive Interpretable Tabular Learning model

Usage

tabnet_fit(x, ...)

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

## S3 method for class 'data.frame'
tabnet_fit(
  x,
  y,
  tabnet_model = NULL,
  config = tabnet_config(),
  ..., 
  from_epoch = NULL
)

## S3 method for class 'formula'
tabnet_fit(
  formula,
  data,
  tabnet_model = NULL,
  config = tabnet_config(),
  ..., 
  from_epoch = NULL
)
## S3 method for class 'recipe'

```
tabnet_fit(
  x,
  data,
  tabnet_model = NULL,
  config = tabnet_config(),
  ...,  
  from_epoch = NULL
)
```

## S3 method for class 'Node'

```
tabnet_fit(
  x,
  tabnet_model = NULL,
  config = tabnet_config(),
  ...,  
  from_epoch = NULL
)
```

### Arguments

- **x** Depending on the context:
  - A **data frame** of predictors.
  - A **matrix** of predictors.
  - A **recipe** specifying a set of preprocessing steps created from `recipes::recipe()`.

  The predictor data should be standardized (e.g. centered or scaled). The model treats categorical predictors internally thus, you don’t need to make any treatment.

- **...** Model hyperparameters. Any hyperparameters set here will update those set by the config argument. See `tabnet_config()` for a list of all possible hyperparameters.

- **y** When `x` is a **data frame** or **matrix**, `y` is the outcome specified as:
  - A **data frame** with 1 or many numeric column (regression) or 1 or many categorical columns (classification).
  - A **matrix** with 1 column.
  - A **vector**, either numeric or categorical.

- **tabnet_model** A previously fitted TabNet model object to continue the fitting on. If `NULL` (the default) a brand new model is initialized.

- **config** A set of hyperparameters created using the `tabnet_config` function. If no argument is supplied, this will use the default values in `tabnet_config()`.

- **from_epoch** When a `tabnet_model` is provided, restore the network weights from a specific epoch. Default is last available checkpoint for restored model, or last epoch for in-memory model.

- **formula** A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.
When a recipe or formula is used, data is specified as:

- A data frame containing both the predictors and the outcome.

**Value**

A TabNet model object. It can be used for serialization, predictions, or further fitting.

**Fitting a pre-trained model**

When providing a parent `tabnet_model` parameter, the model fitting resumes from that model weights at the following epoch:

- last fitted epoch for a model already in torch context
- Last model checkpoint epoch for a model loaded from file
- the epoch related to a checkpoint matching or preceding the `from_epoch` value if provided

The model fitting metrics append on top of the parent metrics in the returned TabNet model.

**Multi-outcome**

TabNet allows multi-outcome prediction, which is usually named multi-label classification or multi-output classification when outcomes are categorical. Multi-outcome currently expect outcomes to be either all numeric or all categorical.

**Threading**

TabNet uses torch as its backend for computation and torch uses all available threads by default.

You can control the number of threads used by torch with:

```r
torch::torch_set_num_threads(1)
torch::torch_set_num_interop_threads(1)
```

**Examples**

```r
data("ames", package = "modeldata")
data("attrition", package = "modeldata")
ids <- sample(nrow(attrition), 256)

## Single-outcome regression using formula specification
fit <- tabnet_fit(Sale_Price ~ ., data = ames, epochs = 1)

## Single-outcome classification using data-frame specification
attrition_x <- attrition[, -which(names(attrition) == "Attrition")]
fit <- tabnet_fit(attrition_x, attrition$Attrition, epochs = 1, verbose = TRUE)

## Multi-outcome regression on `Sale_Price` and `Pool_Area` in `ames` dataset using formula,
ames_fit <- tabnet_fit(Sale_Price + Pool_Area ~ ., data = ames[ids,, epochs = 2, valid_split = 0.2)

## Multi-label classification on `Attrition` and `JobSatisfaction` in
## Attrition dataset using recipe

```r
library(recipes)
rec <- recipe(Attrition + JobSatisfaction ~ ., data = attrition[ids,]) %>%
  step_normalize(all_numeric(), -all_outcomes())

attrition_fit <- tabnet_fit(rec, data = attrition[ids,], epochs = 2, valid_split = 0.2)
```

## Hierarchical classification on `acme`

```r
data(acme, package = "data.tree")
acme_fit <- tabnet_fit(acme, epochs = 2, verbose = TRUE)
```

# Note: Dataset number of rows and model number of epochs should be increased
# for publication-level results.

---

### TabNet Model Architecture

**Description**

This is a `nn_module` representing the TabNet architecture from *Attentive Interpretable Tabular Deep Learning*.

**Usage**

```r
tabnet_nn(
  input_dim, 
  output_dim, 
  n_d = 8, 
  n_a = 8, 
  n_steps = 3, 
  gamma = 1.3, 
  cat_idxs = c(), 
  catDims = c(), 
  cat_emb_dim = 1, 
  n_independent = 2, 
  n_shared = 2, 
  epsilon = 1e-15, 
  virtual_batch_size = 128, 
  momentum = 0.02, 
  mask_type = "sparsemax"
)
```

**Arguments**

- **input_dim**  
  Initial number of features.

- **output_dim**  
  Dimension of network output examples: one for regression, 2 for binary classification etc. Vector of those dimensions in case of multi-output.
### Parameters

- **n_d**: Dimension of the prediction layer (usually between 4 and 64).
- **n_a**: Dimension of the attention layer (usually between 4 and 64).
- **n_steps**: Number of successive steps in the network (usually between 3 and 10).
- **gamma**: Float above 1, scaling factor for attention updates (usually between 1 and 2).
- **cat_idx**: Index of each categorical column in the dataset.
- **cat_dims**: Number of categories in each categorical column.
- **cat_emb_dim**: Size of the embedding of categorical features if int, all categorical features will have same embedding size if list of int, every corresponding feature will have specific size.
- **n_independent**: Number of independent GLU layer in each GLU block of the encoder.
- **n_shared**: Number of independent GLU layer in each GLU block of the encoder.
- **epsilon**: Avoid log(0), this should be kept very low.
- **virtual_batch_size**: Batch size for Ghost Batch Normalization.
- **momentum**: Float value between 0 and 1 which will be used for momentum in all batch norm.
- **mask_type**: Either "sparsemax" or "entmax": this is the masking function to use.

---

### Description

Pretrain the TabNet: Attentive Interpretable Tabular Learning model on the predictor data exclusively (unsupervised training).

### Usage

```r
tabnet_pretrain(x, ...)  
## Default S3 method:  
## S3 method for class 'Var'  
## S3 method for class 'data.frame'  
## S3 method for class 'formula'
```
tabnet_pretrain(
  formula,
  data,
  tabnet_model = NULL,
  config = tabnet_config(),
  ..., 
  from_epoch = NULL
)

## S3 method for class 'recipe'

## S3 method for class 'Node'

Arguments

- **x**
  - Depending on the context:
    - A **data frame** of predictors.
    - A **matrix** of predictors.
    - A **recipe** specifying a set of preprocessing steps created from `recipes::recipe()`.
  - The predictor data should be standardized (e.g. centered or scaled). The model treats categorical predictors internally thus, you don’t need to make any treatment.

- **...**
  - Model hyperparameters. Any hyperparameters set here will update those set by the config argument. See `tabnet_config()` for a list of all possible hyperparameters.

- **y**
  - (optional) When x is a **data frame** or **matrix**, y is the outcome

- **tabnet_model**
  - A pretrained TabNet model object to continue the fitting on. if `NULL` (the default) a brand new model is initialized.

- **config**
  - A set of hyperparameters created using the `tabnet_config` function. If no argument is supplied, this will use the default values in `tabnet_config()`.
from_epoch  When a `tabnet_model` is provided, restore the network weights from a specific epoch. Default is last available checkpoint for restored model, or last epoch for in-memory model.

formula  A formula specifying the outcome terms on the left-hand side, and the predictor terms on the right-hand side.

data  When a `recipe` or `formula` is used, `data` is specified as:
  * A `data frame` containing both the predictors and the outcome.

Value

A TabNet model object. It can be used for serialization, predictions, or further fitting.

outcome

Outcome value are accepted here only for consistent syntax with `tabnet_fit`, but by design the outcome, if present, is ignored during pre-training.

pre-training from a previous model

When providing a parent `tabnet_model` parameter, the model pretraining resumes from that model weights at the following epoch:

  * last pretrained epoch for a model already in torch context
  * Last model checkpoint epoch for a model loaded from file
  * the epoch related to a checkpoint matching or preceding the `from_epoch` value if provided

The model pretraining metrics append on top of the parent metrics in the returned TabNet model.

Threading

TabNet uses `torch` as its backend for computation and `torch` uses all available threads by default.

You can control the number of threads used by `torch` with:

```r
torch::torch_set_num_threads(1)
torch::torch_set_num_interop_threads(1)
```

Examples

```r
data("ames", package = "modeldata")
pretrained <- tabnet_pretrain(Sale_Price ~ ., data = ames, epochs = 1)
```
Index

attention_width (decision_width), 5
autoplot.tabnet_explain, 2
autoplot.tabnet_fit, 3
autoplot.tabnet_pretrain
    (autoplot.tabnet_fit), 3

check_compliant_node, 4
decision_width, 5

feature_reusage (decision_width), 5
mask_type (decision_width), 5
momentum (decision_width), 5

nn_prune_head.tabnet_fit, 6
nn_prune_head.tabnet_pretrain
    (nn_prune_head.tabnet_fit), 6
node_to_df, 6
num_independent (decision_width), 5
num_shared (decision_width), 5
num_steps (decision_width), 5

recipes::recipe(), 14, 18

tabnet, 7
tabnet_config, 9
tabnet_config(), 14, 18
tabnet_explain, 12
tabnet_explain(), 2
tabnet_fit, 13
tabnet_fit(), 3
tabnet_nn, 16
tabnet_pretrain, 17
tabnet_pretrain(), 3
torch::lr_scheduler, 10