keras-pandas Documentation

Release latest

Brendan Herger

Jan 11, 2019

Contents

Welcome to keras-pandas! I'd recommend that you start with the keras-pandas section & Quick Start.

CHAPTER 1

User Guide

1.1 keras-pandas

tl;dr: keras-pandas allows users to rapidly build and iterate on deep learning models.

Getting data formatted and into keras can be tedious, time consuming, and require domain expertise, whether your a veteran or new to Deep Learning. `keras-pandas` overcomes these issues by (automatically) providing:

- **Data transformations**: A cleaned, transformed and correctly formatted `X` and `y` (good for keras, sklearn or any other ML platform)
- **Data piping**: Correctly formatted keras input, hidden and output layers to quickly start iterating on

These approaches are build on best in world approaches from practitioners, kaggle grand masters, papers, blog posts, and coffee chats, to simple entry point into the world of deep learning, and a strong foundation for deep learning experts.

For more info, check out the:

- Code
- Documentation
- PyPi
- Issue tracker
- CI/CD
- Author's website

1.1.1 Quick Start

Let's build a model with the lending club data set. This data set is particularly fun because this data set contains a mix of text, categorical and numerical data types, and features a lot of null values.

```
pip install --upgrade keras-pandas
```

```
from keras import Model
from keras_pandas import lib
from keras_pandas.Automater import Automater
from sklearn.model_selection import train_test_split

# Load data
observations = lib.load_lending_club()

# Train /test split
train_observations, test_observations = train_test_split(observations)
train_observations = train_observations.copy()
test_observations = test_observations.copy()

# List out variable types

data_type_dict = {'numerical': ['loan_amnt', 'annual_inc', 'open_acc', 'dti', 'delinq_
↪2yrs',
                                'inq_last_6mths', 'mths_since_last_delinq', 'pub_rec',
↪ 'revol_bal',
                                'revol_util',
                                'total_acc', 'pub_rec_bankruptcies'],
                  'categorical': ['term', 'grade', 'emp_length', 'home_ownership',
↪'loan_status', 'addr_state',
                                  'application_type', 'disbursement_method'],
                  'text': ['desc', 'purpose', 'title']}
output_var = 'loan_status'

# Create and fit Automater
auto = Automater(data_type_dict=data_type_dict, output_var=output_var)
auto.fit(train_observations)

# Transform data
train_X, train_y = auto.fit_transform(train_observations)
test_X, test_y = auto.transform(test_observations)

# Create and fit keras (deep learning) model.
x = auto.input_nub
x = auto.output_nub(x)

model = Model(inputs=auto.input_layers, outputs=x)
model.compile(optimizer='adam', loss=auto.suggest_loss())
```

And that's it! In a couple of lines, we've created a model that accepts a few dozen variables, and can create a world class deep learning model

1.1.2 Usage

Installation

You can install `keras-pandas` with `pip`:

```
pip install -U keras-pandas
```

Creating an Automater

The `Automater` object is the central object in `keras-pandas`. It accepts a dictionary of the format `{'datatype': ['var1', var2']}`

For example we could create an automater using the built in `numerical`, `categorical`, and `text` datatypes, by calling:

```
# List out variable types
data_type_dict = {'numerical': ['loan_amnt', 'annual_inc', 'open_acc', 'dti', 'delinq_
↪2yrs',
                                'inq_last_6mths', 'mths_since_last_delinq', 'pub_rec',
↪ 'revol_bal',
                                'revol_util',
                                'total_acc', 'pub_rec_bankruptcies'],
                  'categorical': ['term', 'grade', 'emp_length', 'home_ownership',
↪'loan_status', 'addr_state',
                                  'application_type', 'disbursement_method'],
                  'text': ['desc', 'purpose', 'title']}
output_var = 'loan_status'

# Create and fit Automater
auto = Automater(data_type_dict=data_type_dict, output_var=output_var)
```

As a side note, the response variable must be in one of the variable type lists (e.g. `loan_status` is in `categorical_vars`)

One variable type

If you only have one variable type, only use one variable type!

```
# List out variable types
data_type_dict = {'categorical': ['term', 'grade', 'emp_length', 'home_ownership',
↪'loan_status', 'addr_state',
                                  'application_type', 'disbursement_method']}
output_var = 'loan_status'

# Create and fit Automater
auto = Automater(data_type_dict=data_type_dict, output_var=output_var)
```

Multiple variable types

If you have multiple variable types, feel free to use all of them! Built in datatypes are listed in `Automater.datatype_handlers`

```
# List out variable types
data_type_dict = {'numerical': ['loan_amnt', 'annual_inc', 'open_acc', 'dti', 'delinq_
↪2yrs',
                                'inq_last_6mths', 'mths_since_last_delinq', 'pub_rec',
↪ 'revol_bal',
                                'revol_util',
                                'total_acc', 'pub_rec_bankruptcies'],
                  'categorical': ['term', 'grade', 'emp_length', 'home_ownership',
↪'loan_status', 'addr_state',
                                  'application_type', 'disbursement_method'],
```

(continues on next page)

(continued from previous page)

```
                    'text': ['desc', 'purpose', 'title']}
output_var = 'loan_status'

# Create and fit Automater
auto = Automater(data_type_dict=data_type_dict, output_var=output_var)
```

Custom datatypes

If there's a specific datatype you'd like to use that's not built in (such as images, videos, or geospatial), you can include it by using `Automater`'s `datatype_handlers` parameter.

A template datatype can be found in `keras_pandas/data_types/Abstract.py`. Filling out this template will yield a new datatype handler. If you're happy with your work and want to share your new datatype handler, create a PR (and check out `contributing.md`)

No `output_var`

If your model doesn't need a response var, or your use case doesn't use `keras-pandas`'s output functionality, you can skip the `output_var` by setting it to None

```
# List out variable types
data_type_dict = {'categorical': ['term', 'grade', 'emp_length', 'home_ownership',
↪'loan_status', 'addr_state',
                                  'application_type', 'disbursement_method']}
output_var = None

# Create and fit Automater
auto = Automater(data_type_dict=data_type_dict, output_var=output_var)
```

Fitting the Automater

Before use, the `Automator` must be fit. The `fit()` method accepts a pandas DataFrame, which must contain all of the columns listed during initialization.

```
auto.fit(observations)
```

Transforming data

Now, we can use our `Automater` to transform the dataset, from a pandas DataFrame to numpy objects properly formatted for Keras's input and output layers.

```
X, y = auto.transform(observations, df_out=False)
```

This will return two objects:

- `X`: An array, containing numpy object for each Keras input. This is generally one Keras input for each user input variable.
- `y`: A numpy object, containing the response variable (if one was provided)

Using input / output nubs

Setting up correctly formatted, heuristically 'good' input and output layers is often

- Tedious
- Time consuming
- Difficult for those new to Keras

With this in mind, `keras-pandas` provides correctly formatted input and output 'nubs'.

The input nub is correctly formatted to accept the output from `auto.transform()`. It contains one Keras Input layer for each generated input, may contain addition layers, and has all input piplines joined with a `Concatenate` layer.

The output layer is correctly formatted to accept the response variable numpy object.

1.1.3 Contact

Hey, I'm Brendan Herger, avaiable at https://www.hergertarian.com/. Please feel free to reach out to me at `13herger <at> gmail <dot> com`

I enjoy bridging the gap between data science and engineering, to build and deploy data products. I'm not currently pursuing contract work.

I've enjoyed building a unique combination of machine learning, deep learning, and software engineering skills. In my previous work at Capital One and startups, I've has built authorization fraud, insider threat, and legal discovery automation platforms. In each of these cases I've lead a team of data scientists and data engineers to enable and elevate our client's business workflow (and capture some amazing data).

When I'm not knee deep in a code base, I can be found traveling, sharing my collection of Japanese teas, and playing board games with my partner in Seattle.

1.1.4 Changelog

- PR title (#PR number, or #Issue if no PR)
- There's nothing here! (yet)

Development

- Updated README and setup.py links (No PR)

3.1.0

- Add boolean datatype (#104)
- Added Contributing.md section for new datatypes (#101)
- Added datatypes to docs in index.rst (#101)
- Modified documentation to automatically generate API docs (#101)

3.0.1

- Changing CI to Circleci (#100)
- Adding datatypes to CONTRIBUTING.md, adding CONTRIBUTING.md to docs (#96)
- Adding docs badge (#95)
- Adding support for unusual variable names / format keras names to be valid in name scope (#92)
- Adding examples (#93)
- Upgraded `requests` library to `requests==2.20.1`, based on security concern (#94)

3.0.0

Brand new release, with

Added

- New `Datatype` interface, with easier to understand pipelines for each datatype
  - All existing datatypes (`Numerical`, `Categorical`, `Text` & `TimeSeries`) re-implmented in this new format
  - Support for custom data types generated by users
  - Duck-typing helper method (`keras_pandas/lib.check_valid_datatype()`) to confirm that a datatype has valid signature
- New testing, streamlined and standardized
- Support for transforming unseen categorical levels, via the `UNK` token (experimental)

Modified

- Updated `Automater` interface, which accepts a dictionary of data types
- Heavily updated README
- More consistent logging and data formatting for sample data sets

Removed

- Removed examples, will be re-implemented in future release
- All existing unittests
- Bulk of new datatypes in `contributing.md`, will be re-added in future release

2.2.0

- Add timeseries support (#78)
- Add timeseries examples (#79)

2.1.0

- Boolean support deprecated. Boolean (bool) data type can be treated as a special case of categorical data types

2.0.2

- Remove a lot of the unnecessary dependencies (#75)
- Update dependencies to contemporary versions (#74)

2.0.1

- Fix issue w/ PyPi conflict

2.0.0

- Adding CI/CD and PyPi links, and updating contact section w/ about the author (#70)
- Major rewrite / update of examples (#72)
  - Fixes bug in embedding transformer. Embeddings will now be at least length 1.
  - Add functionality to check if `resp_var` is in the list of user provided variables
  - Added better null filling w/ `CategoricalImputer`
  - Added filling unseen values w/ `CategoricalImputer`
  - Converted default transformer pipeline to use `copy.deepcopy` instead of `copy.copy`. This was a hotfix for a previously unknown issue.
  - Standardizing setting logging level, only in test base class and examples (when `__main__`)

1.3.5

- Adding regression example w/ inverse_transformation (#64)
- Fixing issue where web socket connections were being opened needlessly (#65)

1.3.4

- Adding `Manifest.in`, with including files references in `setup.py` (#54)

1.3.2

- Fixed poorly written text embedding index unit test (#52)
- Added license (#49)

Earlier

- Lots of things happened. Break things and move fast

1.2 Contributing

If you're interested in helping out, all open tasks are listed the GitHub Issues tab. The issues tagged with `first issue` are a good place to start if your new to the project or new to open source projects.

If you're interested in a new major feature, please feel free to reach out to me

1.2.1 Bug reports

The best bug reports are Pull Requests. The second best bug reports are new issues on this repo.

1.2.2 Test

This framework uses `unittest` for unit testing. Tests can be run by calling:

```
cd tests/

python -m unittest discover -s . -t .
```

1.2.3 Style guide

This codebase should follow Google's Python Style Guide.

1.2.4 Changelog

If you've changed any code, update the changelog on `README.md`

1.2.5 Generating documentation

This codebase uses sphinx's autodoc feature. To generate new documentation, to reflect updated documentation, run:

```
cd docs

make html
```

1.2.6 Adding new data types

If there's a specific datatype you'd like to use that's not built in (such as images, videos, or geospatial), you can include it by using `Automater`'s `datatype_handlers` parameter.

A template datatype can be found in `keras_pandas/data_types/Abstract.py`. Filling out this template will yield a new datatype handler. If you're happy with your work and want to share your new datatype handler, create a PR.

To create add a new datatype:

- Create a new `.py` file in `keras_pandas/data_types`, based on `keras_pandas/data_types/Abstract.py` (and perhaps referencing `keras_pandas/data_types/Numerical.py`)
- Fill out your new datatype's `.py` file

- Create a new test class for your new datatype (perhaps based on `tests/testDatatypeTemplate.py` and / or `tests/testNumerical.py`)
- Add the new datatype to `keras_pandas/Automater.datatype_handlers`, in `keras_pandas/Automater.__init__()`
- Add the new datatype to `docs/index.rst`, in `autosummary list`

1.2.7 Adding new examples

To contribute a new example

- Add data loader method to `keras_pandas/lib.py` (perhaps in the style of `load_titanic()`)
- Add a new `.py` file under `examples` (perhaps by copying and pasting `example_interface.py`)
- Implement the required steps
- Add the new file to `tests/testExamples.py`
- Add the new example to `examples/README.md`

CHAPTER 2

API

Automater.Automater([data_type_dict, ...])	An Automater object, allows users to rapidly build and iterate on deep learning models.
data_types.Abstract.AbstractDatatype()	Interface for all future datatypes
data_types.Boolean.Boolean()	Support for boolean variables, such as owns_home: *[True, False, True]*, or existing_acct: *[True, True, False]*.
data_types.Categorical.Categorical()	Support for categorical variables, such as fruits: *['apple', 'banana', 'coconut']*, or home_ownership: *['rent', 'own]*.
data_types.Numerical.Numerical()	Support for numerical variables, such as annual_salary: *[175000, 105000, 30000000]*, or countries_visited: *[1, 7, 22, 183, 12]*.
data_types.Text.Text()	Support for text variables, such as title: *['The count of Monte Cristo', 'Alice in Wonderland']*, or article_text: ['Politicians in deadlock over latest international disagreement...', '42 is truly the answer to life, the universe and everything according to a study by British ...']'
data_types.TimeSeries.TimeSeries()	Support for time series data, such as previous_day_closes: *[[123, 3, 0], [777, 42, 0]]* or last_three_purchase_prices: *[[222, 111, 891], [12312312, 412412, 12]]*
lib	Library / helper functions for Keras-Pandas
transformations	SKLearn-compliant transformers, for use as part of pipelines

2.1 Automater.Automater

class Automater.**Automater**(*data_type_dict={}*, *output_var=None*, *datatype_handlers={}*)

An Automater object, allows users to rapidly build and iterate on deep learning models.

This class supports building and iterating on deep learning models by providing:

- A cleaned, transformed and correctly formatted X and y (good for keras, sklearn or any other ML platform)
- An *input_nub*, without the hassle of worrying about input shapes or data types
- An *nub*, correctly formatted for the kind of response variable provided

__init__(*data_type_dict={}*, *output_var=None*, *datatype_handlers={}*)

Parameters

- **data_type_dict** (*{str:[str]}*) – A dictionary, in the format {'datatype': ['variable_name_1', 'variable_name_2']}
- **output_var** (*str*) – The name of the response variable
- **datatype_handlers** (*{str:class}*) – Any custom or external datatype handlers, in the format {'datatype': DataTypeClass}

Methods

__init__([data_type_dict, output_var, ...])	**param data_type_dict** A dictionary, in the format {'datatype': ['variable_name_1', 'variable_name_2']}
fit(observations)	• Fit input mapper
fit_transform(observations)	Perform a *fit*, and then a *transform*.
inverse_transform_output(y)	Transform the output_var to be in the same basis (scale / domain) as it was in the original data set.
suggest_loss()	Suggest a loss function, based on:
transform(observations[, df_out])	• Transform the keras input columns

_check_fitted()

_check_has_response_var()

_check_input_df(*input_dataframe*)

_create_input_nub(*transformed_observations*)

Generate a nub, appropriate for feeding all input variables into a Keras model. Each input variable has one input layer and one input nub, with:

- One Input (required)
- Possible additional layers (optional, such as embedding layers for text)

All input nubs are then joined with a Concatenate layer :param transformed_observations: A pan-

das dataframe, containing all keras input variables after their transformation pipelines :type transformed_observations: pandas.DataFrame :return: A Keras layer, which can be fed into future layers :rtype: ([keras,Input], Layer)

_create_mapper(*variable_list*)

_create_output_nub(*output_observations_transformed*)

_valid_configurations_check()

fit(*observations*)

- Fit input mapper
- Create input layer and nub
- Create output mapper (if supervised)
- Create output nub (if supervised)
- Set *self.fitted* to *True*

Parameters **observations** (*pandas.DataFrame*) – A pandas DataFrame, containing the relevant variables

Returns self, now in a fitted state. The Automater now has initialized input layers, output layer(s) (if response variable is present), and can be used for the transform step

Return type *Automater*

fit_transform(*observations*)
Perform a *fit*, and then a *transform*. See *transform* for return documentation

inverse_transform_output(*y*)
Transform the output_var to be in the same basis (scale / domain) as it was in the original data set. This is convenient for comparing predictions to actual data, and computing metrics relative to actual data and other models :param y: The output of a Keras model's .predict function :type y: numpy.ndarray :return: Data, which can be compared to the original data set :rtype numpy.ndarray

suggest_loss()
Suggest a loss function, based on:

- Output variable datatype
- Observations of output_var

Returns A Keras supported loss function

transform(*observations*, *df_out=False*)

- Transform the keras input columns
- Transform the output_var, if supervised and the output_var is present
- Format the data, consistent w/ df_out

Parameters

- **observations** (*pandas.DataFrame*) – A pandas dataframe, containing all keras input layers
- **df_out** – Whether to return a Pandas DataFrame. Returns DataFrame if True, keras-compatable object if

false :type df_out: bool :return: Either a pandas dataframe (if *df_out* = *True*), or a numpy object (if *df_out* = *False*). This object

will contain: the transformed input variables, and the transformed output variables (if the output variable is present in *input_dataframe*

2.2 data_types.Abstract.AbstractDatatype

class data_types.Abstract.**AbstractDatatype**

Interface for all future datatypes

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
input_nub_generator(variable, ...)	Generate an input layer and input 'nub' for a Keras network.
output_inverse_transform(y_pred, ...)	Undo the transforming that was done to get data into a keras model.
output_nub_generator(variable, ...)	Generate an output layer for a Keras network.
output_suggested_loss()	

_check_output_support()

input_nub_generator(*variable*, *transformed_observations*)

Generate an input layer and input 'nub' for a Keras network.

- input_layer: The input layer accepts data from the outside world.
- input_nub: The input nub will always include the input_layer as its first layer. It may also include

other layers for handling the data type in specific ways

Parameters

- **variable** (*str*) – Name of the variable
- **transformed_obervations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns A tuple containing the input layer, and the last layer of the nub

output_inverse_transform(*y_pred*, *response_transform_pipeline*)

Undo the transforming that was done to get data into a keras model. This inverse transformation will render the observations so they can be compared to the data in the natural scale provided by the user :param response_transform_pipeline: An SKLearn transformation pipeline, trained on the same variable as the model which produced y_pred :param y_pred: The data predicted by keras :return: The same data, in the natural basis

output_nub_generator(*variable*, *input_observations*)

Generate an output layer for a Keras network.

- output_layer: A keras layer, which is formatted to correctly accept the response variable

Parameters

- **variable** (*str*) – A Variable contained in the input_df
- **input_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns output_layer

output_suggested_loss()

2.3 data_types.Boolean.Boolean

class data_types.Boolean.**Boolean**

Support for boolean variables, such as owns_home: *[True, False, True]*, or existing_acct: *[True, True, False]*.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
input_nub_generator(variable, ...)	Generate an input layer and input 'nub' for a Keras network.
output_inverse_transform(y_pred, ...)	Undo the transforming that was done to get data into a keras model.
output_nub_generator(variable, ...)	Generate an output layer for a Keras network.
output_suggested_loss()	

_check_output_support()

input_nub_generator(*variable*, *transformed_observations*)

Generate an input layer and input 'nub' for a Keras network.

- input_layer: The input layer accepts data from the outside world.
- input_nub: The input nub will always include the input_layer as its first layer. It may also include

other layers for handling the data type in specific ways

Parameters

- **variable** (*str*) – Name of the variable
- **transformed_obervations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns A tuple containing the input layer, and the last layer of the nub

output_inverse_transform(*y_pred*, *response_transform_pipeline*)

Undo the transforming that was done to get data into a keras model. This inverse transformation will render the observations so they can be compared to the data in the natural scale provided by the user :param response_transform_pipeline: An SKLearn transformation pipeline, trained on the same variable as the model which produced y_pred :param y_pred: The data predicted by keras :return: The same data, in the natural basis

output_nub_generator(*variable*, *input_observations*)
Generate an output layer for a Keras network.

- output_layer: A keras layer, which is formatted to correctly accept the response variable

Parameters

- **variable** (*str*) – A Variable contained in the input_df
- **input_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns output_layer

output_suggested_loss()

2.4 data_types.Categorical.Categorical

class data_types.Categorical.**Categorical**
Support for categorical variables, such as fruits: *['apple', 'banana', 'coconut']*, or home_ownership: *['rent', 'own]*.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
input_nub_generator(variable, ...)	Generate an input layer and input 'nub' for a Keras network.
output_inverse_transform(y_pred, ...)	Undo the transforming that was done to get data into a keras model.
output_nub_generator(variable, ...)	Generate an output layer for a Keras network.
output_suggested_loss()	

_check_output_support()

static input_nub_generator(*variable*, *transformed_observations*)
Generate an input layer and input 'nub' for a Keras network.

- input_layer: The input layer accepts data from the outside world.
- input_nub: The input nub will always include the input_layer as its first layer. It may also include

other layers for handling the data type in specific ways

Parameters

- **variable** (*str*) – Name of the variable
- **transformed_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns A tuple containing the input layer, and the last layer of the nub

output_inverse_transform(*y_pred*, *response_transform_pipeline*)
Undo the transforming that was done to get data into a keras model. This inverse transformation will render the observations so they can be compared to the data in the natural scale provided by the user :param response_transform_pipeline: An SKLearn transformation pipeline, trained on the same variable as the model which produced y_pred :param y_pred: The data predicted by keras :return: The same data, in the natural basis

output_nub_generator(*variable*, *transformed_observations*)
Generate an output layer for a Keras network.

- output_layer: A keras layer, which is formatted to correctly accept the response variable

Parameters

- **variable** (*str*) – A Variable contained in the input_df
- **transformed_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns output_layer

output_suggested_loss()

2.5 data_types.Numerical.Numerical

class data_types.Numerical.**Numerical**
Support for numerical variables, such as annual_salary: *[175000, 105000, 30000000]*, or countries_visited: *[1, 7, 22, 183, 12]*.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
input_nub_generator(variable, ...)	Generate an input layer and input 'nub' for a keras network.
output_inverse_transform(y_pred, ...)	Undo the transforming that was done to get data into a keras model.
output_nub_generator(variable, ...)	Generate an output layer for a Keras network.
output_suggested_loss()	

_check_output_support()

static input_nub_generator(*variable*, *transformed_observations*)
Generate an input layer and input 'nub' for a keras network.

- input_layer: The input layer accepts data from the outside world.
- input_nub: The input nub will always include the input_layer as its first layer. It may also include

other layers for handling the data type in specific ways

Parameters

- **variable** (*str*) – Name of the variable
- **transformed_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns A tuple containing the input layer, and the last layer of the nub

output_inverse_transform(*y_pred*, *response_transform_pipeline*)

Undo the transforming that was done to get data into a keras model. This inverse transformation will render the observations so they can be compared to the data in the natural scale provided by the user :param response_transform_pipeline: An SKLearn transformation pipeline, trained on the same variable as the model which produced y_pred :param y_pred: The data predicted by keras :return: The same data, in the natural basis

output_nub_generator(*variable*, *transformed_observations*)

Generate an output layer for a Keras network.

- output_layer: A keras layer, which is formatted to correctly accept the response variable

Parameters

- **variable** (*str*) – A Variable contained in the input_df
- **transformed_observations** – A dataframe, containing either the specified variable, or derived

variables :type transformed_observations: pandas.DataFrame :return: output_layer

output_suggested_loss()

2.6 data_types.Text.Text

class data_types.Text.**Text**

Support for text variables, such as title: *['The count of Monte Cristo', 'Alice in Wonderland']*, or article_text: ['Politicians in deadlock over latest international disagreement...', '42 is truly the answer to life, the universe and everything according to a study by British ...']'

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
input_nub_generator(variable, ...)	Generate an input layer and input 'nub' for a Keras network.
output_inverse_transform(y_pred, ...)	Undo the transforming that was done to get data into a keras model.
output_nub_generator(variable, ...)	Generate an output layer for a Keras network.
output_suggested_loss()	

_check_output_support()

static input_nub_generator(*variable*, *transformed_observations*)

Generate an input layer and input 'nub' for a Keras network.

- input_layer: The input layer accepts data from the outside world.
- input_nub: The input nub will always include the input_layer as its first layer. It may also include

other layers for handling the data type in specific ways

Parameters

- **variable** (*str*) – Name of the variable
- **transformed_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns A tuple containing the input layer, and the last layer of the nub

output_inverse_transform(*y_pred*, *response_transform_pipeline*)

Undo the transforming that was done to get data into a keras model. This inverse transformation will render the observations so they can be compared to the data in the natural scale provided by the user :param response_transform_pipeline: An SKLearn transformation pipeline, trained on the same variable as the model which produced y_pred :param y_pred: The data predicted by keras :return: The same data, in the natural basis

output_nub_generator(*variable*, *transformed_observations*)

Generate an output layer for a Keras network.

- output_layer: A keras layer, which is formatted to correctly accept the response variable

Parameters

- **variable** (*str*) – A Variable contained in the input_df
- **transformed_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns output_layer

output_suggested_loss()

2.7 data_types.TimeSeries.TimeSeries

class data_types.TimeSeries.**TimeSeries**

Support for time series data, such as previous_day_closes: *[[123, 3, 0], [777, 42, 0]]* or last_three_purchase_prices: *[[222, 111, 891], [12312312, 412412, 12]]*

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()	Initialize self.
input_nub_generator(variable, ...)	Generate an input layer and input 'nub' for a Keras network.
output_inverse_transform(y_pred, ...)	Undo the transforming that was done to get data into a keras model.

Continued on next page

Table 8 – continued from previous page

output_nub_generator(variable, ...)	Generate an output layer for a Keras network.
output_suggested_loss()	

_check_output_support()

static input_nub_generator(*variable*, *transformed_observations*)
Generate an input layer and input 'nub' for a Keras network.

- input_layer: The input layer accepts data from the outside world.
- input_nub: The input nub will always include the input_layer as its first layer. It may also include

other layers for handling the data type in specific ways

Parameters

- **variable** (*str*) – Name of the variable
- **transformed_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns A tuple containing the input layer, and the last layer of the nub

output_inverse_transform(*y_pred*, *response_transform_pipeline*)
Undo the transforming that was done to get data into a keras model. This inverse transformation will render the observations so they can be compared to the data in the natural scale provided by the user :param response_transform_pipeline: An SKLearn transformation pipeline, trained on the same variable as the model which produced y_pred :param y_pred: The data predicted by keras :return: The same data, in the natural basis

output_nub_generator(*variable*, *transformed_observations*)
Generate an output layer for a Keras network.

- output_layer: A keras layer, which is formatted to correctly accept the response variable

Parameters

- **variable** (*str*) – A Variable contained in the input_df
- **transformed_observations** (*pandas.DataFrame*) – A dataframe, containing either the specified variable, or derived variables

Returns output_layer

output_suggested_loss()

2.8 lib

Library / helper functions for Keras-Pandas

Functions

check_valid_datatype(datatype_class)	Check whether the provided *datatype_class* meets the requirements for use as a keras-pandas datatype handler, using duck-typing
check_variable_list_are_valid(variable_type_dict)	Checks that the provided variable_type_dict is valid, by:
download_file(url, local_file_path, filename)	Download the file at *url* in chunks, to the location at *local_file_path*
get_temp_dir()	
get_variable_type(variable_name, ...)	
load_instanbul_stocks([as_ts])	
load_iris()	
load_lending_club([test_run])	
load_mushroom()	
load_titanic()	Load the titanic data set, as a pandas DataFrame
namespace_conversion(input_string)	Convert input_string to be sfve in the tensorflow namespace

lib.**check_valid_datatype**(*datatype_class*)

Check whether the provided *datatype_class* meets the requirements for use as a keras-pandas datatype handler, using duck-typing

Parameters **datatype_class** – A class, with the expected signature

Returns Whether or not the datatype_class has the requirements

Return type bool

lib.**check_variable_list_are_valid**(*variable_type_dict*)

Checks that the provided variable_type_dict is valid, by:

- Confirming there is no overlap between all variable lists

Parameters **variable_type_dict** (*{str:[str]}*) – A dictionary, with keys describing variables types, and values listing particular variables

Returns True, if there is no overlap

Return type bool

lib.**download_file**(*url*, *local_file_path*, *filename*)

Download the file at *url* in chunks, to the location at *local_file_path*

Parameters

- **url** (*str*) – URL to a file to be downloaded
- **local_file_path** (*str*) – Path to download the file to
- **filename** (*str*) – Filename to save the data to

Returns The path to the file on the local machine (same as input *local_file_path*)

Return type str

lib.**get_temp_dir**()

lib.**get_variable_type**(*variable_name*, *variable_type_dict*, *response_var*)

lib.**load_instanbul_stocks**(*as_ts=False*)

lib.**load_iris**()

lib.**load_lending_club**(*test_run=True*)

lib.**load_mushroom**()

lib.**load_titanic**()
Load the titanic data set, as a pandas DataFrame

Returns A DataFrame, containing the titanic dataset

Return type pandas.DataFrame

lib.**namespace_conversion**(*input_string*)
Convert input_string to be sfve in the tensorflow namespace

Parameters **input_string** (*str*) – A string, to be converted

Returns Cleanly formatted version of input_string

Return type str

2.9 transformations

SKLearn-compliant transformers, for use as part of pipelines

Classes

CategoricalImputer([missing_values, ...])	Impute missing values from a categorical/string np.ndarray or pd.Series with the most frequent value on the training data.
EmbeddingVectorizer([max_sequence_length])	Converts text into padded sequences.
LabelEncoder	Encode labels with value between 0 and n_classes-1.
TimeSeriesVectorizer([max_sequence_length])	
TypeConversionEncoder(conversion_type)	

class transformations.**CategoricalImputer**(*missing_values='NaN'*, *strategy='most_frequent'*, *fill_value='?'*, *fill_unknown_labels=False*, *copy=True*)
Impute missing values from a categorical/string np.ndarray or pd.Series with the most frequent value on the training data. Parameters ———- missing_values : string or "NaN", optional (default="NaN")

The placeholder for the missing values. All occurrences of *missing_values* will be imputed. None and np.nan are treated as being the same, use the string value "NaN" for them.

copy [boolean, optional (default=True)] If True, a copy of X will be created.

strategy [string, optional (default = 'most_frequent')] The imputation strategy. - If "most_frequent", then replace missing using the most frequent

value along each column. Can be used with strings or numeric data.

- If "constant", then replace missing values with fill_value. Can be used with strings or numeric data.

fill_value [string, optional (default='?')] The value that all instances of *missing_values* are replaced with if *strategy* is set to *constant*. This is useful if you don't want to impute with the mode, or if there are multiple modes in your data and you want to choose a particular one. If *strategy* is not set to *constant*, this parameter is ignored.

fill_ [str] The imputation fill value

static _get_null_mask(*X*, *value*)
Compute the boolean mask X == missing_values.

_get_unknown_label_mask(*X*)
Compute the boolean mask X == missing_values.

fit(*X*, *y=None*)
Get the most frequent value. Parameters ———-

X [np.ndarray or pd.Series] Training data.

y : Passthrough for `Pipeline` compatibility.

self: CategoricalImputer

transform(*X*)
Replaces missing values in the input data with the most frequent value of the training data. Parameters ———-

X [np.ndarray or pd.Series] Data with values to be imputed.

np.ndarray Data with imputed values.

class `transformations.`**EmbeddingVectorizer**(*max_sequence_length=None*)
Converts text into padded sequences. The output of this transformation is consistent with the required format for Keras embedding layers

For example *'the fat man'* might be transformed into *[2, 0, 27, 1, 1, 1]*, if the *embedding_sequence_length* is 6.

There are a few sentinel values used by this layer:

- *0* is used for the UNK token (tokens which were not seen during training)
- *1* is used for the padding token (to fill out sequences that shorter than *embedding_sequence_length*)

fit(*X*, *y=None*)

static generate_embedding_sequence_length(*observation_series*)

static pad(*input_sequence*, *length*, *pad_char*)
Pad the given iterable, so that it is the correct length.

Parameters

- **input_sequence** – Any iterable object
- **length** (*int*) – The desired length of the output.
- **pad_char** (*str or int*) – The character or int to be added to short sequences

Returns A sequence, of len *length*

Return type []

static prepare_input(*X*)

process_string(*input_string*)
Turn a string into padded sequences, consistent with Keras's Embedding layer

- Simple preprocess & tokenize
- Convert tokens to indices

- Pad sequence to be the correct length

Parameters **input_string** (*str*) – A string, to be converted into a padded sequence of token indices

Returns A padded, fixed-length array of token indices

Return type [int]

transform(*X*)

class transformations.**LabelEncoder**

Encode labels with value between 0 and n_classes-1.

Read more in the User Guide.

classes_ [array of shape (n_class,)] Holds the label for each class.

LabelEncoder can be used to normalize labels.

```
>>> from sklearn import preprocessing
>>> le = preprocessing.LabelEncoder()
>>> le.fit([1, 2, 2, 6])
LabelEncoder()
>>> le.classes_
array([1, 2, 6])
>>> le.transform([1, 1, 2, 6])
array([0, 0, 1, 2]...)
>>> le.inverse_transform([0, 0, 1, 2])
array([1, 1, 2, 6])
```

It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```

sklearn.preprocessing.OneHotEncoder [encode categorical integer features] using a one-hot aka one-of-K scheme.

fit(*y*)

Fit label encoder

y [array-like of shape (n_samples,)] Target values.

self : returns an instance of self.

fit_transform(*y*, ***kwargs*)

Fit label encoder and return encoded labels

y [array-like of shape [n_samples]] Target values.

y : array-like of shape [n_samples] :param **kwargs:

inverse_transform(*y*)
Transform labels back to original encoding.

y [numpy array of shape [n_samples]] Target values.

y : numpy array of shape [n_samples]

transform(*y*)
Transform labels to normalized encoding.

y [array-like of shape [n_samples]] Target values.

y : array-like of shape [n_samples]

class transformations.**TimeSeriesVectorizer**(*max_sequence_length=None*)

fit(*X*, *y=None*)

transform(*X*)

class transformations.**TypeConversionEncoder**(*conversion_type*)

fit(*X*, *y=None*)

transform(*X*)

Python Module Index

l

t

Index

Symbols

A

B

C

D

E

F

G

I

L

N

O

P

S

T