configurator documentation

Release 0.2.0

Yasser Gonzalez

February 04, 2015

Contents

1 Introduction

2 API reference

2.1 Main modules

configurator

Adaptive configuration dialogs.

`configurator.__version__`
The current version string.

configurator.base

Base configuration dialogs.

These classes are not intented to be instantiated directly, see the classes defined in `configurator.dp`, `configurator.rl` and `configurator.optim`.

class `configurator.base.`**`Dialog`**(*config_values*, *rules*, *validate=False*)

Base configuration dialog.

This is the base class of all the configuration dialogs defined in the package (not intented to be instantiated directly). It defines a common interface shared by the dialogs generated using the different `configurator.base.DialogBuilder` subclasses.

Parameters

- **config_values** – A list with one entry for each variable, containing an enumerable with all the possible values of the variable.
- **rules** – A list of `configurator.assoc_rules.AssociationRule`.
- **validate** – Indicates whether the dialog initialization should be validated or not. A *ValueError* exception will be raised if an error is found.

The interaction with all subclasses must be as follows. First, `reset()` should be called to begin at a state where all the configuration variables are unknown. Next, a call to `get_next_question()` will suggest a question, which can be posed to the user and the answer should be given as feedback to the dialog using `set_answer()`. It is possible to ignore the suggestion given by the dialog and answer the questions in any order. In this case, simply call `set_answer()` and future calls to `get_next_question()` will act accordingly.

The `config` attribute can be used at any time to retrieve the configuration values collected so far. `is_complete()` can be used to check whether all the variables have been set.

`config`

The current configuration state, i.e. a dict mapping variable indices to their values.

`config_values`

A list with one entry for each variable, containing an enumerable with all the possible values of the variable.

`rules`

A list of `configurator.assoc_rules.AssociationRule`.

`get_next_question`()

Get the question that should be asked next.

Returns the question that should be asked next to the user, according to this dialog. Each question is identified by the index of the corresponding variable.

Returns An integer, the variable index.

`is_complete`()

Check if the configuration is complete.

Returns *True* if the values of all the variables has been set, *False* otherwise.

`reset`()

Reset the configurator to the initial state.

In the initial configuration state the value of all the variables is unknown. This method must be called before making any call to `get_next_question()` or `set_answer()`.

`set_answer`(*var_index*, *var_value*)

Set the value of a configuration variable.

It wil be usually called with a variable index returned right before by `get_next_question()` and the answer that the user gave to the question.

Parameters

- **var_index** – An integer, the variable index.
- **var_value** – The value of the variable. It must be one of the possible values of the variable in the `config_values` attribute.

class `configurator.base.`**`DialogBuilder`**(*config_sample=None*, *config_values=None*, *validate=False*, *assoc_rule_min_support=0.1*, *assoc_rule_min_confidence=0.99*)

Base configuration dialog builder.

Parameters

- **config_sample** – A two-dimensional numpy array containing a sample of the configuration variables.
- **config_values** – A list with one entry for each variable, containing an enumerable with all the possible values of the variable. If it is not given, it is computed from `config_sample`.
- **validate** – Whether or not to run some (generally costly) checks on the generated model and the resulting `configurator.base.Dialog` instance. Mostly intended for testing purposes.
- **assoc_rule_min_support** – Minimum item set support in [0,1].
- **assoc_rule_min_confidence** – Minimum confidence in [0,1].

`build_dialog`()

Construct a configuration dialog.

Returns An instance of a `configurator.base.Dialog` subclass.

configurator.dp

Configuration dialogs based on dynamic programming.

class `configurator.dp.`**`DPDialogBuilder`**(*dp_algorithm='policy-iteration'*, *dp_max_iter=1000*, *dp_discard_states=True*, *dp_partial_assoc_rules=True*, *dp_aggregate_terminals=True*, ***kwargs*)

Bases: `configurator.base.DialogBuilder`

Build a configuration dialog using dynamic programming.

Parameters

- **dp_algorithm** – Algorithm for solving the MDP. Possible values are: *'policy-iteration'* and *'value-iteration'*.
- **dp_max_iter** – The maximum number of iterations of the algorithm used to solve the MDP.
- **dp_discard_states** – Indicates whether states that can't be reached from the initial state after applying the association rules should be discarded.
- **dp_partial_assoc_rules** – Indicates whether the association rules can be applied when some of the variables in the right-hand-side are already set to the correct values. (the opposite is to require that all variables in the left-hand-side are unknown).
- **dp_aggregate_terminals** – Indicates whether all terminal states should be aggregated into a single state.

See configurator.base.DialogBuilder for the remaining arguments.

build_dialog()
Construct a configuration dialog.

Returns An instance of a configurator.base.Dialog subclass.

configurator.rl

Configuration dialogs based on reinforcement learning.

class configurator.rl.**RLDialogBuilder**(*rl_algorithm='q-learning'*, *rl_table='exact'*, *rl_table_features=None*, *rl_learning_rate=0.3*, *rl_epsilon=0.5*, *rl_epsilon_decay=0.99*, *rl_max_episodes=1000*, ***kwargs*)
Bases: configurator.base.DialogBuilder

Build a configuration dialog using reinforcement learning.

Parameters

- **rl_table** – Representation of the action-value table. Possible values are *'exact'* (full table) and *'approx'* (approximate table with state aggregation).
- **rl_table_features** – Set of features used to approximate the action-value table with state aggregation. The features are given in an iterable containing any subset of: *'known-vars'* (number of known variables), *'last-answer'* (the last question asked to the user and his/her answer).
- **rl_algorithm** – The reinforcement learning algorithm. Possible values are: *'q-learning'* and *'sarsa'*.
- **rl_learning_rate** – Q-learning and SARSA learning rate.
- **rl_epsilon** – Initial epsilon value for the epsilon-greedy exploration strategy.
- **rl_epsilon_decay** – Epsilon decay rate. The epsilon value is decayed after every episode.
- **rl_max_episodes** – Maximum number of simulated episodes.

See configurator.base.DialogBuilder for the remaining arguments.

build_dialog()
Construct a configuration dialog.

Returns An instance of a configurator.base.Dialog subclass.

configurator.optim

Configuration dialogs based on optimization.

2.2 Supporting modules

configurator.assoc_rules

Association rule mining.

class configurator.assoc_rules.**AssociationRule**(*lhs*, *rhs*, *support=None*, *confidence=None*)
An association rule mined by configurator.assoc_rules.AssociationRuleMiner.

Parameters

- **lhs** – Left-hand-side (also called antecedent or body) of the rule. A dict mapping variable indices to their values.
- **rhs** – Right-hand-side (also called consequent or head) of the rule. A dict mapping variable indices to their values.
- **support** – Item set support in [0,1].
- **confidence** – Confidence of the association rule in [0,1].

All the arguments are available as instance attributes.

apply_rule(*observation*)
Apply the rule.

Complete the values of a partial observation of the variables by setting the variables in the right-hand-side to the values indicated by the rule. `is_applicable()` must be called first to ensure that no variables will be overwritten.

Parameters **observation** – A dict mapping variable indices to their values. It is updated in-place.

is_applicable(*observation*)
Check if the rule can be applied.

Check if both the left-hand-side and the right-hand-side of the rule are compatible with a partial observation of the variables. See the documentation of the is_lhs_compatible and is_rhs_compatible methods.

Parameters **observation** – A dict mapping variable indices to their values.

Returns *True* if rule is applicable, *False* if not.

is_lhs_compatible(*observation*)
Check left-hand-side compatibility.

Check if the rules' left-hand-side is compatible with an observation of the variables. All the variables in the left-hand-side must match the observed values.

Parameters **observation** – A dict mapping variable indices to their values.

Returns *True* if the left-hand-side is compaible, *False* if not.

is_rhs_compatible(*observation*)
Check right-hand-side compatibility.

Check if the rules' right-hand-side is compatible with a partial observation of the variables. Each variable in the right-hand-side must be unknown or have the same observed value. At least one variable must be unknown to avoid trivial applications of the rule.

Parameters **observation** – A dict mapping variable indices to their values.

Returns *True* if the right-hand-side is compaible, *False* if not.

class `configurator.assoc_rules.`**AssociationRuleMiner**(*data*)
Association rule mining.

Parameters **data** – A two-dimensional numpy array.

All the arguments are available as instance attributes.

Discover association rules in a two-dimensional numpy array. Each column is expected to represent a categorical variable and each row a multi-variate observation. The frequent item sets are mined using the FP-growth algorithm implementation provided by Christian Borgelt's PyFIM library available at http://www.borgelt.net/pyfim.html.

mine_assoc_rules(*min_support=0.1*, *min_confidence=0.8*, *min_len=2*, *max_len=None*)

Parameters

- **min_support** – Minimum rule support in [0,1].
- **min_confidence** – Minimum confidence of the rules in [0,1].
- **min_len** – Minimum number of items per item set.
- **max_len** – Maximum number of items per item set (default: no limit).

Returns A list of `configurator.assoc_rules.AssociationRule` instances.

configurator.util

Utility functions.

configurator.util.**load_config_sample**(*csv_file*, *dtype=<class 'numpy.uint8'>*)

Load a CSV file with a sample of categorical variables.

Read the CSV file and return an equivalent numpy array with the different categorical values represented by integers.

Parameters

- **csv_file** – Path to a CSV file.
- **dtype** – dtype of the returned numpy array.

Returns A two-dimensional numpy array.

configurator.util.**get_config_values**(*config_sample*)

Get the possible configuration values from the sample.

Parameters **config_sample** – A two-dimensional numpy array containing a sample of the configuration variables.

Returns A list with one entry for each variable, containing a list with all the possible values of the variable.

configurator.util.**simulate_dialog**(*dialog*, *config*)

Simulate a configuration dialog.

Simulate the use of the dialog to predict the given configuration.

Parameters

- **dialog** – An instance of a Dialog subclass.
- **config** – A complete configuration, i.e. a dict mapping variable indices to their values.

Returns A tuple with two elements. The first elements gives the accuracy of the prediction, the second the number of questions that were asked (both normalized in [0,1]).

configurator.util.**cross_validation**(*n_folds*, *builder_class*, *builder_kwargs*, *config_sample*, *config_values=None*)

Measure the performance of a configuration dialog builder.

Use the dialog builder to perform a k-folds cross validation on the given configuration sample. The sample is shuffled before dividing it into batches. The performance is measured in terms of the accuracy of the predicted configuration and the number of questions that were asked.

Parameters

- **n_folds** – Number of folds. Must be at least 2.

- **builder_class** – A `configurator.base.DialogBuilder` subclass.
- **builder_kwargs** – A dict with arguments to pass to builder_class when a new instance is created (except `config_sample` and `config_values`).
- **config_sample** – A two-dimensional numpy array containing a sample of the configuration variables.
- **config_values** – A list with one entry for each variable, containing an enumerable with all the possible values of the variable. If it is not given, it is automatically computed from the columns of `config_sample`.

Returns A *pandas.DataFrame* with one row for each fold and one column for each one of the following statistics: mean and standard deviation of the prediction accuracy, mean and standard deviation of the number of questions that were asked (normalized by the total number of questions).

`configurator.util.`**`measure_scalability`**(*builder_class*, *builder_kwargs*, *config_sample*, *config_values=None*)

Measure the scalability of a configuration dialog builder.

Use the dialog builder to construct a sequence of dialogs with an increasing number of variables from the configuration sample, measuring the CPU time on each case. The variables in the configuration sample are added in a random order.

Parameters

- **builder_class** – A `configurator.base.DialogBuilder` subclass.
- **builder_kwargs** – A dict with arguments to pass to builder_class when a new instance is created (except `config_sample` and `config_values`).
- **config_sample** – A two-dimensional numpy array containing a sample of the configuration variables.
- **config_values** – A list with one entry for each variable, containing an enumerable with all the possible values of the variable. If it is not given, it is automatically computed from the columns of `config_sample`.

Returns A *pandas.DataFrame* with two columns. The first column gives the number of binary variables and the second the corresponding measured CPU time (in seconds). The number of binary variables is computed as the log to the base 2 of the number of possible configurations.

Index

Symbols

A

B

C

D

G

I

L

M

R

S