

Python Script

Extends functionalities through Python scripting.

Inputs

- Data (Orange.data.Table): input dataset bound to `in_data` variable
- Learner (Orange.classification.Learner): input learner bound to `in_learner` variable
- Classifier (Orange.classification.Learner): input classifier bound to `in_classifier` variable
- Object: input Python object bound to `in_object` variable

Outputs

- Data (Orange.data.Table): dataset retrieved from `out_data` variable
- Learner (Orange.classification.Learner): learner retrieved from `out_learner` variable
- Classifier (Orange.classification.Learner): classifier retrieved from `out_classifier` variable
- Object: Python object retrieved from `out_object` variable

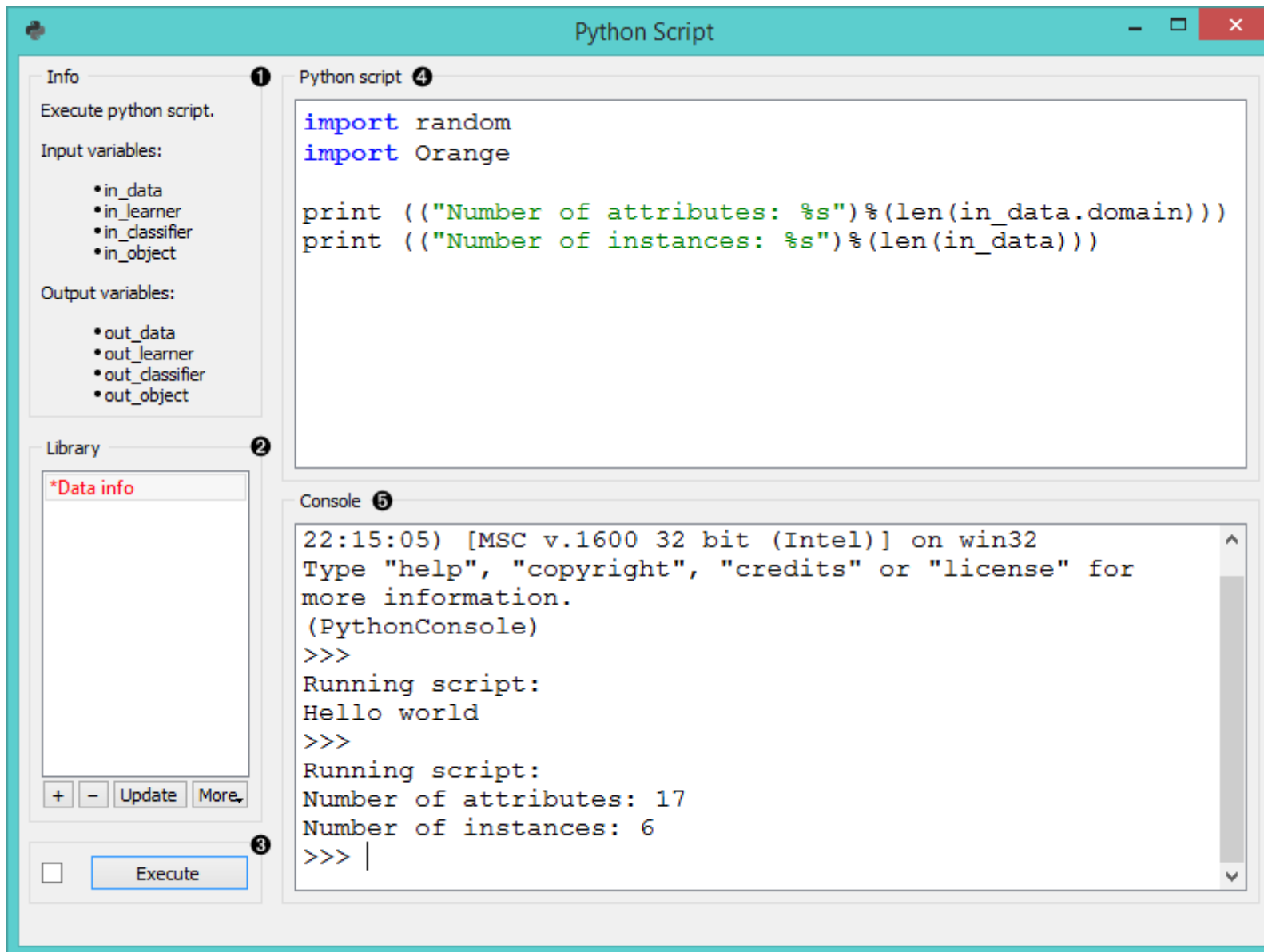
Python Script widget can be used to run a python script in the input, when a suitable functionality is not implemented in an existing widget. The script has `in_data`, `in_distance`, `in_learner`, `in_classifier` and `in_object` variables (from input signals) in its local namespace. If a signal is not connected or it did not yet receive any data, those variables contain `None`.

After the script is executed variables from the script's local namespace are extracted and used as outputs of the widget. The widget can be further connected to other widgets for visualizing the output.

For instance the following script would simply pass on all signals it receives:

```
out_data = in_data
out_distance = in_distance
out_learner = in_learner
out_classifier = in_classifier
out_object = in_object
```

Note: You should not modify the input objects in place.



1. Info box contains names of basic operators for Orange Python script.
2. The *Library* control can be used to manage multiple scripts. Pressing "+" will add a new entry and open it in the *Python script* editor. When the script is modified, its entry in the *Library* will change to indicate it has unsaved changes. Pressing *Update* will save the script (keyboard shortcut "Ctrl+S"). A script can be removed by selecting it and pressing the "-" button.
3. Pressing *Execute* in the *Run* box executes the script (keyboard shortcut "Ctrl+R"). Any script output (from `print`) is captured and displayed in the *Console* below the script.

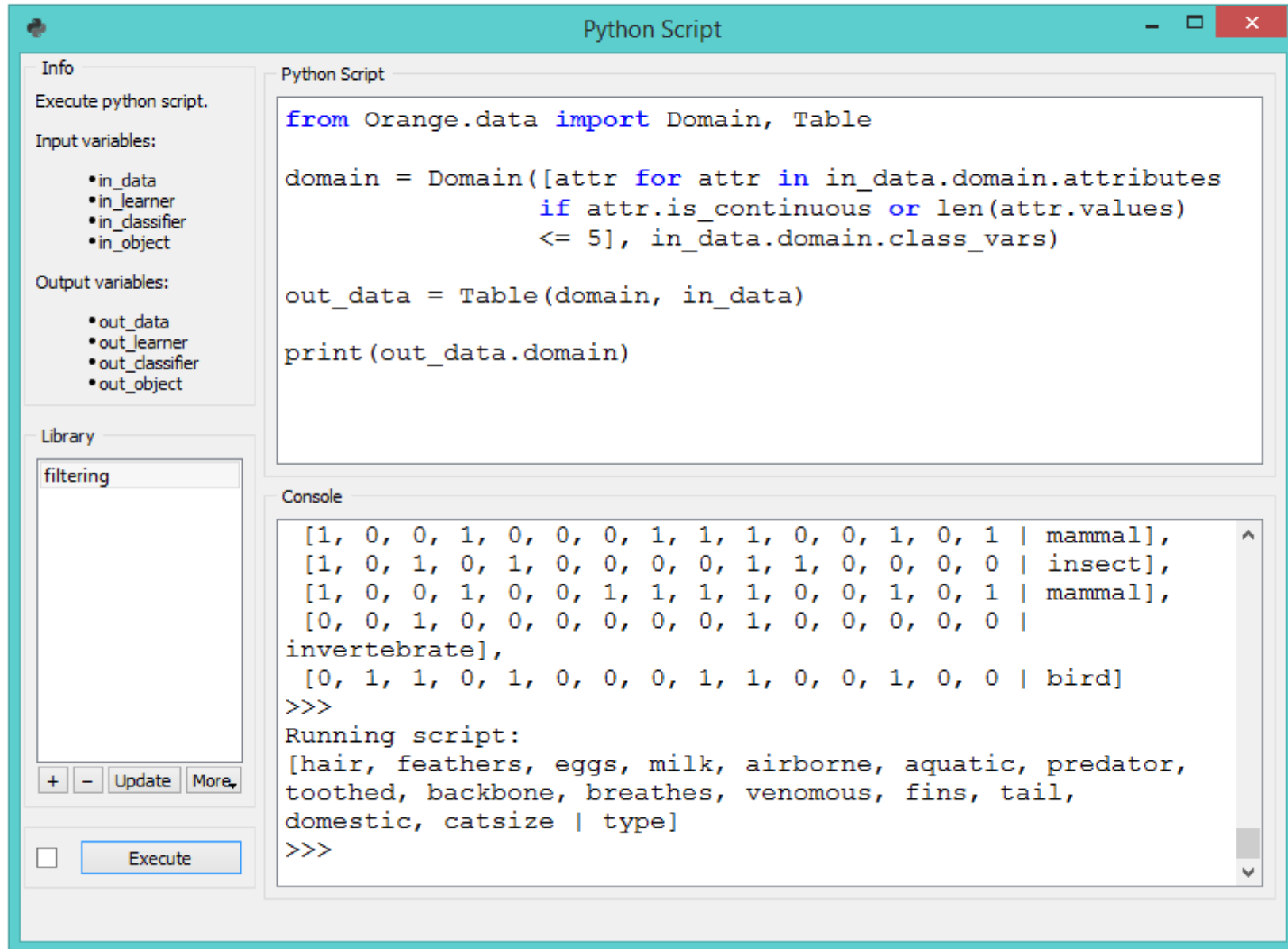
4. The *Python script* editor on the left can be used to edit a script (it supports some rudimentary syntax highlighting).
5. Console displays the output of the script.

Examples

Python Script widget is intended to extend functionalities for advanced users. Classes from Orange library are described in the [documentation](#). To find further information about orange Table class see [Table](#), [Domain](#), and [Variable](#) documentation.

One can, for example, do batch filtering by attributes. We used zoo.tab for the example and we filtered out all the attributes that have more than 5 discrete values. This in our case removed only 'leg' attribute, but imagine an example where one would have many such attributes.

```
from Orange.data import Domain, Table
domain = Domain([attr for attr in in_data.domain.attributes
                  if attr.is_continuous or len(attr.values) <= 5],
               in_data.domain.class_vars)
out_data = Table(domain, in_data)
```



The second example shows how to round all the values in a few lines of code. This time we used wine.tab and rounded all the values to whole numbers.

```
import numpy as np
out_data = in_data.copy()
#copy, otherwise input data will be overwritten
np.round(out_data.X, 0, out_data.X)
```



Python Script

Info

Execute python script.

Input variables:

- in_data
- in_learner
- in_classifier
- in_object

Output variables:

- out_data
- out_learner
- out_classifier
- out_object

Library

round

+ - Update More

☐ Execute

Python Script

```
import numpy as np

out_data = in_data.copy()
#copy, otherwise input data will be overwritten

np.round(out_data.X, 0, out_data.X)

print(out_data)
```

Console

```
[14.000, 6.000, 2.000, 20.000, 95.000, 2.000, 1.000,
1.000, 1.000, 8.000, 1.000, 2.000, 740.000 | 3],
[13.000, 4.000, 2.000, 23.000, 102.000, 2.000, 1.000,
0.000, 1.000, 7.000, 1.000, 2.000, 750.000 | 3],
[13.000, 4.000, 2.000, 20.000, 120.000, 2.000, 1.000,
0.000, 1.000, 10.000, 1.000, 2.000, 835.000 | 3],
[13.000, 3.000, 2.000, 20.000, 120.000, 2.000, 1.000,
1.000, 1.000, 9.000, 1.000, 2.000, 840.000 | 3],
[14.000, 4.000, 3.000, 24.000, 96.000, 2.000, 1.000,
1.000, 1.000, 9.000, 1.000, 2.000, 560.000 | 3]
>>>
```

The third example introduces some Gaussian noise to the data. Again we make a copy of the input data, then walk through all the values with a double for loop and add random noise.

```
import random
from Orange.data import Domain, Table
new_data = in_data.copy()
for inst in new_data:
    for f in inst.domain.attributes:
        inst[f] += random.gauss(0, 0.02)
out_data = new_data
```

The screenshot shows the 'Python Script' widget in the Orange Data Mining software. The window has a teal title bar with the text 'Python Script' and standard window controls. On the left, there is a sidebar with three sections: 'Info', 'Library', and 'Execute'. The 'Info' section contains instructions to 'Execute python script.' and lists input variables (in_data, in_learner, in_classifier, in_object) and output variables (out_data, out_learner, out_classifier, out_object). The 'Library' section shows a list with 'noise' and buttons for '+', '-', 'Update', and 'More...'. The 'Execute' section has a checkbox and an 'Execute' button. The main area is divided into two panes: 'Python Script' and 'Console'. The 'Python Script' pane contains the following code:

```
import random

new_data = in_data.copy()

for inst in new_data:
    for f in inst.domain.attributes:
        inst[f] += random.gauss(0, 0.02)

out_data = new_data
print(out_data)
```

The 'Console' pane shows the output of the script, which is a list of 15 data points, each with 4 numerical features and a class label 'Iris-virginica':

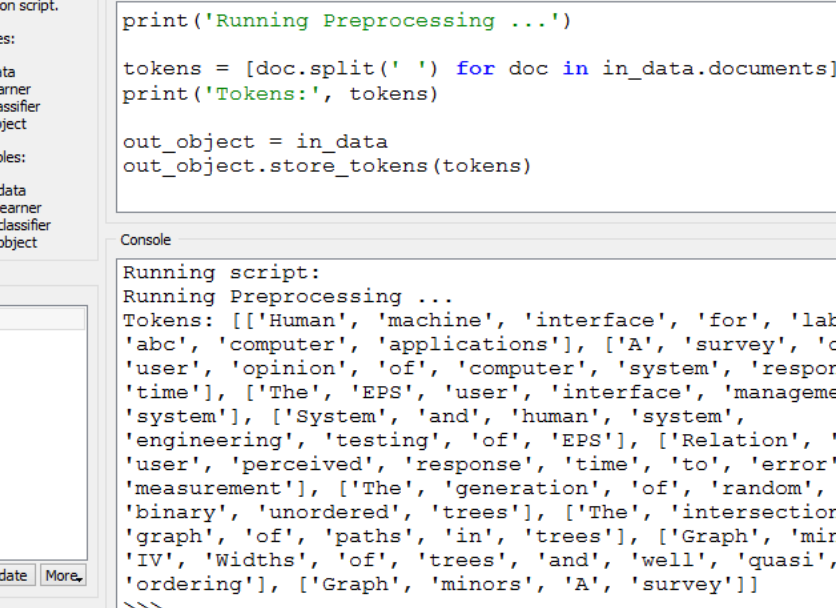
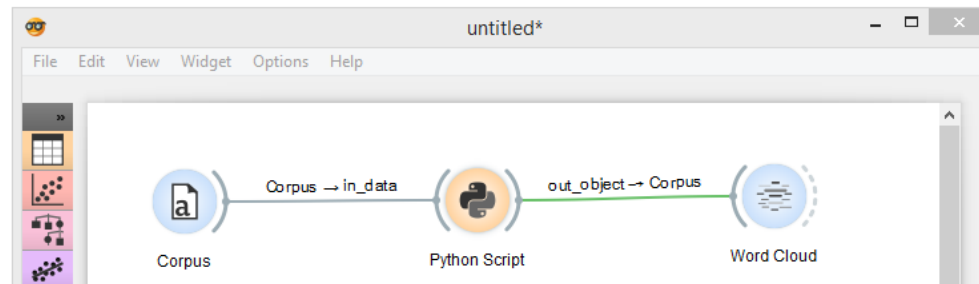
```
[6.386, 3.137, 5.473, 1.809 | Iris-virginica],
[6.010, 3.005, 4.783, 1.820 | Iris-virginica],
[6.880, 3.068, 5.396, 2.115 | Iris-virginica],
[6.702, 3.117, 5.602, 2.390 | Iris-virginica],
[6.885, 3.122, 5.073, 2.275 | Iris-virginica],
[5.813, 2.717, 5.089, 1.905 | Iris-virginica],
[6.816, 3.181, 5.918, 2.319 | Iris-virginica],
[6.717, 3.322, 5.701, 2.476 | Iris-virginica],
[6.720, 2.993, 5.184, 2.322 | Iris-virginica],
[6.318, 2.536, 4.983, 1.853 | Iris-virginica],
[6.512, 3.006, 5.202, 2.039 | Iris-virginica],
[6.188, 3.393, 5.377, 2.300 | Iris-virginica],
[5.888, 3.009, 5.118, 1.787 | Iris-virginica]
```

Below the list, there are two prompt lines: >>> and >>>.

The final example uses Orange3-Text add-on. **Python Script** is very useful for custom preprocessing in text mining, extracting new features from strings, or utilizing advanced *nltk* or *gensim* functions. Below, we simply tokenized our input data from *deerwester.tab* by splitting them by whitespace.


```
print('Running Preprocessing ...')
tokens = [doc.split(' ') for doc in in_data.documents]
print('Tokens:', tokens)
out_object = in_data
out_object.store_tokens(tokens)
```

You can add a lot of other preprocessing steps to further adjust the output. The output of **Python Script** can be used with any widget that accepts the type of output your script produces. In this case, connection is green, which signalizes the right type of input for Word Cloud widget.



The screenshot displays the Orange3 Python Script widget interface. The top bar is teal with the title "Python Script" and standard window controls. The interface is divided into four main sections:

- Info:** Contains a button "Execute python script." and two lists of variables:
 - Input variables:
 - in_data
 - in_learner
 - in_classifier
 - in_object
 - Output variables:
 - out_data
 - out_learner
 - out_classifier
 - out_object
- Library:** A text box containing the word "text". Below it are buttons for "+", "-", "Update", and "More...". At the bottom is a checkbox labeled "Auto Execute" which is checked.
- Python Script:** A large text area containing the following Python code:

```
print('Running Preprocessing ...')

tokens = [doc.split(' ') for doc in in_data.documents]
print('Tokens:', tokens)

out_object = in_data
out_object.store_tokens(tokens)
```
- Console:** A text area showing the execution output:

```
Running script:
Running Preprocessing ...
Tokens: [['Human', 'machine', 'interface', 'for', 'lab',
'abc', 'computer', 'applications'], ['A', 'survey', 'of',
'user', 'opinion', 'of', 'computer', 'system', 'response',
'time'], ['The', 'EPS', 'user', 'interface', 'management',
'system'], ['System', 'and', 'human', 'system',
'engineering', 'testing', 'of', 'EPS'], ['Relation', 'of',
'user', 'perceived', 'response', 'time', 'to', 'error',
'measurement'], ['The', 'generation', 'of', 'random',
'binary', 'unordered', 'trees'], ['The', 'intersection',
'graph', 'of', 'paths', 'in', 'trees'], ['Graph', 'minors',
'IV', 'Widths', 'of', 'trees', 'and', 'well', 'quasi',
'ordering'], ['Graph', 'minors', 'A', 'survey']]
>>>
```

[illegible]