predeval Documentation

Release 0.0.8

Dan Vatterott

Feb 16, 2019

Contents:

This software is built to identify unexpected changes in a model output before evaluation data becomes available.

For example, if you create a churn model, you will have to wait X number of weeks before learning whether users churned (and can evaluate your churn model predictions). This software will not guarantee that your model is accurate, but it will alert you if your model's outputs (i.e., predictions) are dramatically different from what they have been in the past.

CHAPTER 1

predeval

This software is built to identify changes in a model output before evaluation data becomes available.

For example, if you create a churn model, you will have to wait X number of weeks before learning whether users churned (and can evaluate your churn model predictions).

This software will not guarantee that your model is accurate, but it will alert you if your model's outputs (i.e., predictions) are different from what they have been in the past. A model's output can pass predeval tests and be inaccurate and a model's output can fail predeval and be accurate. That said, unexpected changes in model outputs likely represent a change in accuracy.

- Free software: MIT license
- Documentation: https://predeval.readthedocs.io.

1.1 Installation

Installation is described here: https://predeval.readthedocs.io/en/latest/installation.html

1.2 Example Usage

Examples can be found here: https://predeval.readthedocs.io/en/latest/usage.html

1.3 API Documentation

Documentation of the software can be found here: https://predeval.readthedocs.io/en/latest/api.html

1.4 Contributing

Info about contributing can be found here: https://predeval.readthedocs.io/en/latest/contributing.html

1.5 Changelog

Changelog can be found here: https://predeval.readthedocs.io/en/latest/history.html

1.6 Credits

Info about contributors can be found here: https://predeval.readthedocs.io/en/latest/authors.html

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install predeval, run this command in your terminal:

```
$ pip install predeval
```

This is the preferred method to install predeval, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for predeval can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dvatterott/predeval
```

Or download the tarball:

```
$ curl  -OL https://github.com/dvatterott/predeval/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Examples

Jupyter notebookes with examples using using scikit-learn can be found here: https://github.com/dvatterott/predeval/tree/master/example_notebooks

3.1 ContinuousEvaluator

Example of using the ContinuousEvaluator

```
from predeval import ContinuousEvaluator, evaluate_tests

# create continuous sample.
# this might typically be your model's output from a training data-set
from numpy.random import uniform, seed
seed(1234)
model_output = uniform(0, 100, size=(1000,))

# create evaluator object
ce = ContinuousEvaluator(model_output)
ce.update_param('minimum', 0)  # we know our data should not be less than 0
ce.update_param('maximum', 100) # we also know our data should not be greater than 100

# this might typically be your production model's output
new_model_output = uniform(0, 100, size=(1000,))

# check whether the new output is different than expected
test_results = ce.check_data(new_model_output)
# Passed min check; min observed=0.0227
# Passed max check; max observed=99.8069
# Passed mean check; mean observed=48.2344 (Expected 50.8805 +- 58.9384)
# Passed std check; std observed=29.5791 (Expected 29.4692 +- 14.7346)
# Passed ks check; test statistic=0.0510, p=0.1441

# print test outputs. note we will not generate assertion errors on failure.
```

(continues on next page)

(continued from previous page)

```
from predeval import evaluate_tests
evaluate_tests(test_results, assert_test=False)
# Passed min test.
# Passed max test.
# Passed mean test.
# Passed std test.
# Passed ks test.

changed_model_output = uniform(0, 100, size=(1000,)) + 20
changed_test_results = ce.check_data(changed_model_output)
# Passed min check; min observed=20.0043
# Failed max check; max observed=119.7728
# Passed mean check; mean observed=70.7836 (Expected 50.8805 +- 58.9384)
# Passed std check; std observed=28.9444 (Expected 29.4692 +- 14.7346)
# Failed ks check; test statistic=0.2170, p=0.0000

evaluate_tests(changed_test_results, assert_test=False)
# Passed min test.
# Failed max test.
# Passed mean test.
# Passed std test.
# Failed ks test.
```

3.2 CategoricalEvaluator

Example of using the CategoricalEvaluator

```
from predeval import CategoricalEvaluator, evaluate_tests

# create categorical sample.
# this might typically be your model's output from a training data-set
from numpy.random import choice, seed
seed(1234)
model_output = choice([0, 1, 2], size=(1000,))

# create evaluator object
ce = CategoricalEvaluator(model_output)

# this might typically be your production model's output
new_model_output = choice([0, 1, 2], size=(1000,))

# check whether the new output is different than expected
test_results = ce.check_data(new_model_output)
# Passed chi2 check; test statistic=0.7317, p=0.6936
# Passed min check; observed=[0 1 2] (Expected [0, 1, 2])

# print test outputs. note we will not generate assertion errors on failure.
from predeval import evaluate_tests
evaluate_tests(test_results, assert_test=False)
# Passed chi2 test.
# Passed exist test.

changed_model_output = choice([0, 1, 2], size=(1000,))
changed_model_output[:200] = 0
```

(continues on next page)

(continued from previous page)

```
changed_test_results = ce.check_data(changed_model_output)
# Failed chi2 check; test statistic=59.0655, p=0.0000
# Passed min check; observed=[0 1 2] (Expected [0, 1, 2])

evaluate_tests(changed_test_results, assert_test=False)
# Failed chi2 test.
# Passed exist test.
```

3.3 Updating test parameters

Example of changing the minimum expected value to 0. I demonstrate the three different ways this can be done.

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)
ce.update_param('minimum', 0)

# or

ce.assertion_params['minimum'] = 0

# or

ce.update_min([0])
```

Example of changing the maximum expected value to 100.

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)
ce.update_param('maximum', 100)
```

Example of changing the expected mean to 50.

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)
ce.update_param('mean', 50)
```

Example of changing expected standard-deviation to 10.

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)
ce.update_param('std', 10)
```

Example of changing Kolmogorov-Smirnov test threshold to 1.

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)
ce.update_param('ks_stat', 1)
```

Example of changing Kolmogorov-Smirnov test.

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)
ce.update_ks_test(new_model_output)
```

Example of changing Chi-square test of independence threshold to 3.

```
from predeval import CategoricalEvaluator
ce = CategoricalEvaluator(model_output)
ce.update_chi2_test(new_model_output)
```

Example of changing Chi-square test.

```
from predeval import CategoricalEvaluator
ce = CategoricalEvaluator(model_output)
ce.update_param('chi2_stat', 3)
```

Example of changing expected categories to 1, 2, and 3.

```
from predeval import CategoricalEvaluator
ce = CategoricalEvaluator(model_output)
ce.update_param('cat_exists', [1, 2, 3])
```

3.4 Changing evaluation tests

You might not want to run the entire test suite. Here's some examples of how to change what tests are run.

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output, assertions=['min', 'max'])

# or you can run the tests one at a time.

ce.check_min(new_model_output)
ce.check_max(new_model_output)
```

3.5 Saving and Loading your evaluator

Here's an example of how to save and load your evaluator in python3 (remember to import your evaluator before loading the object).

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)

from joblib import dump, load
dump(ce, 'con_eval.joblib')  # save evaluator
ce = load('con_eval.joblib')  # load evaluator
```

Here's an example of how to save and load your evaluator in python2 (remember to import your evaluator before loading the object).

```
from predeval import ContinuousEvaluator
ce = ContinuousEvaluator(model_output)

import cloudpickle

# save evaluator
with open('con_eval.pkl', 'wb') as f:
    cloudpickle.dump(ce, f)
```

(continues on next page)

(continued from previous page)

```
# load evaluator
with open('con_eval.pkl', 'rb') as f:
    ce = cloudpickle.load(f)
```

CHAPTER 4

API

This page contains a comprehensive list of all classes and functions in predeval.

4.1 ContinuousEvaluator

Library of classes for evaluating continuous model outputs.

class predeval.continuous.**ContinuousEvaluator**(*ref_data*, *assertions=None*, *verbose=True*, ***kwargs*)

Bases: predeval.parent.ParentPredEval

Evaluator for continuous model outputs (e.g., regression models).

By default, this will run the tests listed in the assertions attribute (['min', 'max', 'mean', 'std', 'ks_test']). You can change the tests that will run by listing the desired tests in the assertions parameter.

The available tests are min, max, mean, std, and ks_test.

...

Parameters

- **ref_data** (*list of int or float or np.array*) – This the reference data for all tests. All future data will be compared to this data.
- **assertions** (*list of str, optional*) – These are the assertion tests that will be created. Defaults is ['chi2_test', 'exist'].
- **verbose** (*bool, optional*) – Whether tests should print their output. Default is true

Variables

- **assertion_params** (*dict*) – dictionary of test names and values defining these tests.
  - **minimum** [float] Expected minimum.
  - **maximum** [float] Expected maximum.
  - **mean** [float] Expected mean.

- **std** [float] Expected standard-deviation.
- **ks_stat: float** ks-test-statistic. When this value is exceeded. The test 'failed'.
- **ks_test** [func] Partially evaluated ks test.

• **assertions** (*list of str*) – This list of strings describes the tests that will be run on comparison data. Defaults to ['min', 'max', 'mean', 'std', 'ks_test']

check_data(*test_data*)

Check whether test_data is as expected.

Run threw all tests in assertions and return whether the data passed these tests.

Parameters **test_data** (*list or np.array*) – This the data that will be compared to the reference data.

Returns **output** – Each tuple has a string a boolean. The string describes the test. The boolean describes the outcome. True is a pass and False is a fail.

Return type list of tuples

check_ks(*test_data*)

Test whether test_data is similar to reference data.

If the returned ks-test-statistic is greater than the threshold (default 0.2), the test failed.

The threshold is set by assertion_params['ks_test'].

Uses Kolmogorov-Smirnov test from scipy.

Parameters **comparison_data** (*list or np.array, optional*) – This the data that will be compared to the reference data.

Returns 2 item tuple with test name and boolean expressing whether passed test.

Return type (string, bool)

check_max(*test_data*)

Check whether test_data has any larger values than expected.

The expected max is controlled by assertion_params['max'].

Parameters **comparison_data** (*list or np.array, optional*) – This the data that will be compared to the reference data.

Returns 2 item tuple with test name and boolean expressing whether passed test.

Return type (string, bool)

check_mean(*test_data*)

Check whether test_data has a different mean than expected.

If the observed mean is more than 2 standard deviations from the expected mean, the test fails.

The expected mean is controlled by assertion_params['mean'].

The expected standard deviation is controlled by assertion_params['std'].

Parameters **comparison_data** (*list or np.array, optional*) – This the data that will be compared to the reference data.

Returns 2 item tuple with test name and boolean expressing whether passed test.

Return type (string, bool)

check_min(*test_data*)
Check whether test_data has any smaller values than expected.

The expected min is controlled by assertion_params['min'].

Parameters **comparison_data** (*list or np.array, optional*) – This the data that will be compared to the reference data.

Returns 2 item tuple with test name and boolean expressing whether passed test.

Return type (string, bool)

check_std(*test_data*)
Check whether test_data has any larger values than expected.

If the observed standard deviation is less than 1/2 the expected std or greater than 1.5 times the expected std, then the test fails.

The expected standard deviation is controlled by assertion_params['std'].

Parameters **comparison_data** (*list or np.array, optional*) – This the data that will be compared to the reference data.

Returns 2 item tuple with test name and boolean expressing whether passed test.

Return type (string, bool)

update_ks_test(*input_data*)
Create partially evaluated ks_test.

Uses Kolmogorov-Smirnov test from scipy.

Parameters **input_data** (*list or np.array*) – This the reference data for the ks-test. All future data will be compared to this data.

Returns

Return type None

update_max(*input_data*)
Find max of input data.

Parameters **input_data** (*list or np.array*) – This the reference data for the max-test. All future data will be compared to this data.

Returns

Return type None

update_mean(*input_data*)
Find mean of input data.

Parameters **input_data** (*list or np.array*) – This the reference data for the max-test. All future data will be compared to this data.

Returns

Return type None

update_min(*input_data*)
Find min of input_data.

Parameters **input_data** (*list or np.array*) – This the reference data for the min-test. All future data will be compared to this data.

Returns

Return type None

update_param(*param_key*, *param_value*)
Update value in assertion param dictionary attribute.

Parameters

- **param_key** (*string*) – This is the assertion param that we want to update.
- **param_value** (*real number or partially evaluated test.*) – This is the updated value.

Returns

Return type None

update_std(*input_data*)
Find standard deviation of input data.

Parameters **input_data** (*list or np.array*) – This the reference data for the max-test. All future data will be compared to this data.

Returns

Return type None

4.2 CategoricalEvaluator

Library of classes for evaluating categorical model outputs.

class predeval.categorical.**CategoricalEvaluator**(*ref_data*, *assertions=None*, *verbose=True*, ***kwargs*)
Bases: predeval.parent.ParentPredEval

Evaluator for categorical model outputs (e.g., classification models).

By default, this will run the tests listed in the assertions attribute ([‘chi2_test’, ‘exist’]). You can change the tests that will run by listing the desired tests in the assertions parameter.

The available tests are chi2_test and exist.

. . .

Parameters

- **ref_data** (*list of int or float or np.array*) – This the reference data for all tests. All future data will be compared to this data.
- **assertions** (*list of str, optional*) – These are the assertion tests that will be created. Defaults is [‘chi2_test’, ‘exist’].
- **verbose** (*bool, optional*) – Whether tests should print their output. Default is true

Variables

- **assertion_params** (*dict*) – dictionary of test names and values defining these tests.
  - **chi2_stat** [float] Chi2-test-statistic. When this value is exceeded. The test ‘failed’.
  - **chi2_test** [func] Partially evaluated chi2 test.
  - **cat_exists** [list of int or str] This is a list of the expected model outputs
- **assertions** (*list of str*) – This list of strings describes the tests that will be run on comparison data. Defaults to [‘chi2_test’, ‘exist’]

check_chi2(*test_data*)
Test whether test_data is similar to reference data.

If the returned chi2-test-statistic is greater than the threshold (default 2), the test failed.

The threshold is set by assertion_params['chi2_test'].

Uses chi2_contingency test from scipy.

Parameters **test_data** (*list or np.array*) – This the data that will be compared to the reference data.

Returns 2 item tuple with test name and boolean expressing whether passed test.

Return type (string, bool)

check_data(*test_data*)
Check whether test_data is as expected.

Run threw all tests in assertions and return whether the data passed these tests.

Parameters **test_data** (*list or np.array*) – This the data that will be compared to the reference data.

Returns **output** – Each tuple has a string a boolean. The string describes the test. The boolean describes the outcome. True is a pass and False is a fail.

Return type list of tuples

check_exist(*test_data*)
Check that all distinct values present in test_data.

If any values missing, then the function will return a False (rather than true).

The expected values is controlled by assertion_params['cat_exists'].

Parameters **test_data** (*list or np.array*) – This the data that will be compared to the reference data.

Returns 2 item tuple with test name and boolean expressing whether passed test.

Return type (string, bool)

update_chi2_test(*input_data*)
Create partially evaluated chi2 contingency test.

Uses chi2_contingency test from scipy.

Parameters **input_data** (*list or np.array*) – This the reference data for the ks-test. All future data will be compared to this data.

Returns

Return type None

update_exist(*input_data*)
Create input data for test checking whether all categorical outputs exist.

Parameters **input_data** (*list or np.array*) – This the reference data for the check_exist. All future data will be compared to it.

Returns

Return type None

update_param(*param_key*, *param_value*)
Update value in assertion param dictionary attribute.

Parameters

- **param_key** (*string*) – This is the assertion param that we want to update.
- **param_value** (*real number or partially evaluated test.*) – This is the updated value.

Returns

Return type None

4.3 Utilities

Helper functions for the predeval module.

predeval.utilities.**evaluate_tests**(*test_ouputs*, *assert_test=False*, *verbose=True*)

Check whether the data passed evaluation tests.

Parameters

- **test_ouputs** (*list of tuples*) – Each tuple has a string a boolean. The string describes the test. The boolean describes the outcome. True is a pass and False is a fail. This is the output of the check_data method.
- **assert_test** (*bool*) – Whether to assert the test passed. Default is False.
- **verbose** (*bool*) – Whether to print whether each test was passed or not.

Returns

Return type None

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/dvatterott/predeval/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

predeval could always use more documentation, whether as part of the official predeval docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dvatterott/predeval/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *predeval* for local development.

1. Fork the *predeval* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/predeval.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv predeval
$ cd predeval/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 predeval tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/dvatterott/predeval/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_predeval
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Dan Vatterott <dvatterott@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.0.1 (2019-01-06)

- First release on PyPI.

7.2 0.0.2 (2019-01-06)

- Adding pypi deploy to travis.

7.3 0.0.3 (2019-01-06)

- Only one travis version publishes to pypi.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index