scikit-validate Documentation

Release 0.4.3

Luke Kreczko

Jul 08, 2021

CONTENTS:

CHAPTER

ONE

SCIKIT-VALIDATE

- Free software: Apache Software License 2.0
- Documentation: https://scikit-validate.readthedocs.io.
- Issues: https://github.com/FAST-HEP/scikit-validate/issues

1.1 Overview

scikit-validate is a validation package for science output developed within F.A.S.T.. This package provides commands for monitoring and comparing analysis outputs, computing resource usage (e.g. CPU time/RAM) as well as commands for summarising findings.

It is meant to provide analysis groups or small experiments with some of the fundamental features needed tovalidate (i.e. compare to a reference) the outcomes of their code and to provide easy access to the results.

1.2 Features

- Collect metrics in JSON output * measure file metrics (e.g. size) * measure execution time and memory usage * compare to previous executions
- compare ROOT files & plot discrepancies
- create validation reports

1.3 Credits

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

1.4 Make a new RELEASE

```
export RELEASE=X.Y.Z
export CHANGELOG_GITHUB_TOKEN=<your github token>
make release
```

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install scikit-validate, run this command in your terminal:

```
$ pip install skvalidate
```

This is the preferred method to install scikit-validate, 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 scikit-validate can be downloaded from the GitLab repo.

You can either clone the public repository:

```
$ git clone https://gitlab.cern.ch/fast-hep/public/scikit-validate.git
```

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

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

CHAPTER

THREE

QUICKSTART

3.1 Storing resource usage

Especially when constrained by available computing resources it is good to keep an eye on the resource usage of your analysis/application. For this purpose we provide the *sv_execute* command which encapsulates the script/executable and periodically checks the memory usage as well as reports the time taken at the end.

The following will execute `stress --cpu 1 --io 1 --vm 1 --vm-bytes 128M --timeout 10s --verbose` and output the resource usage into the `resource_metrics.json` file.

```
sv_execute -m resource_metrics.json \
   -- \
   stress --cpu 1 --io 1 --vm 1 --vm-bytes 128M --timeout 10s --verbose
```

In the first part, `sv_execute -m resource_metrics.json`, we set the parameters for `sv_execute`. `--` marks the end of `sv_execute` parameters, everthing after that is considered as the command (and parameters) to be executed. The standard output of the called command is uneffected:

```
...
stress: dbug: [1844521] allocating 134217728 bytes ...
stress: dbug: [1844521] touching bytes in strides of 4096 bytes ...
...
>>> Ran command: "stress --cpu 1 --io 1 --vm 1 --vm-bytes 128M --timeout 10s␣
↪--verbose"
>>> in 11.424817s and used 93.3 MB of memory.
```

3.2 Storing file information

3.3 Comparing two ROOT files

3.4 Adding high-level information

CHAPTER

FOUR

USAGE

After installation scikit-validate will provide several commands, all starting with *sv_*:

4.1 sv_file_info

The first subcommand will simply record the file size of a given file and record it in a JSON file:

```
sv_file_info --help
Usage: sv_file_info [OPTIONS] [INPUT_FILES]...

Script to record file metrics.

For testing pick or create a file:

  # create 10 MB file     dd if=/dev/zero of=test.file bs=10485760
  count=1     sv_add_file_metrics test.file -m metrics.json

If the output file, default metrics.json, already exists it will be read
first and results will be appended.

Options:
-m, --metrics-file TEXT  file for JSON output
--help                   Show this message and exit.
```

4.2 sv_execute

This subcommand will execute the parameters passed to it as a shell command and monitor its resource usage. At the moment only (simple) CPU time and RAM usage are supported:

```
sv_execute --help
Usage: sv_execute [OPTIONS] COMMAND

  Command that wraps and monitors another command.

  For testing install 'stress' package and run

      sv_execute -m resource_metrics.json -- \
            stress --cpu 1 --io 1 --vm 1 --vm-bytes 128M --timeout 10s --verbose

  If the output file, default resource_metrics.json, already exists it will
```

(continues on next page)

(continued from previous page)

```
  be read first and results will be appended.

  If a single string argument is provided as the command then it will be
  split using white-space, however if multiple arguments are provided then
  no additional splitting is performed.  In this case though, use `--`
  before the command so that options are passed to the command, rather than
  this script.

Options:
    -m, --metrics-file PATH
    --memprof-file PATH
    --sample-interval FLOAT  Sampling period (in seconds), defaults to 0.1
    --help                   Show this message and exit.
```

4.3 sv_get_artifact_url

Reads the ENV variable in a Gitlab CI job and constructs a URL for a given existing file or folder.

e.g.:

```
sv_get_artefact_url output/test_file
```

will return `${CI_PROJECT_URL}/-/jobs/${CI_JOB_ID}/artifacts/file/output/test_file`

while:

```
sv_get_artefact_url output
```

will return `${CI_PROJECT_URL}/-/jobs/${CI_JOB_ID}/artifacts/browse/output`

4.4 sv_get_target_branch

Script to extract the target branch for a given project and commit hash.

Meant to be run within a Gitlab CI job and needs the following ENV variables defined:

- CI_PROJECT_ID (automatic from CI job)
- CI_COMMIT_SHA (automatic from CI job)
- CI_API_TOKEN (to be set in the Gitlab project: settings -> pipelines -> add variable)

Related issue: https://gitlab.com/gitlab-org/gitlab-ce/issues/15280

4.5 sv_merge_json

Merges dictionaries in <N>JSON files into one output file. Uses dict.update() → last occurrence of a key will take precedence. Usage:

```
sv_merge_json [OPTIONS] [INPUT_FILES]... OUTPUT
```

4.6 sv_remove_from_env

Removes a path from an environment variable, e.g.

```
sv_remove_from_env /a/b/c:/a/b/d:/d/b/a /a/b
```

will result in */d/b/a*. Recommended use is to clean up ENV variables:

```
PATH=`sv_remove_from_env /a/b/c:/a/b/d:/d/b/a /a/b`
```

4.7 sv_metric_diff

```
Usage: sv_metric_diff [OPTIONS] FILE_UNDER_TEST REFERENCE_FILE

  Display the difference between two metric (JSON) files.

  Examples:     sv_metric_diff
  skvalidate/data/examples/performance_metrics*.json     sv_metric_diff
  skvalidate/data/examples/file_metrics*.json

Options:
  -o, --output-format [console|csv|markdown]
  --help                          Show this message and exit.
```

Example output:

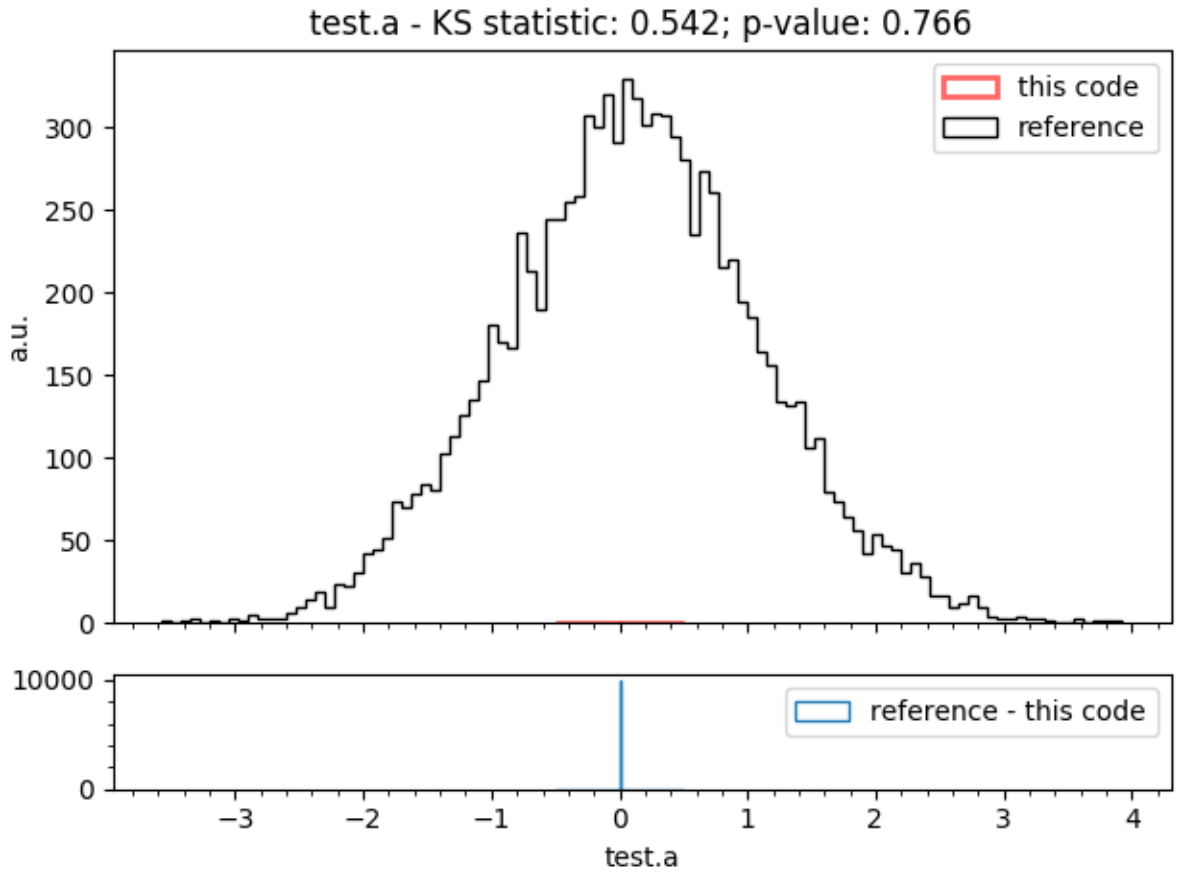
```
sv_metric_diff skvalidate/data/examples/file_metrics*
+---------------------------------------+-----------+--------+-------------+------
↪--+----------+--------+
| file                                  | metric    |   value |   ref value |  ␣
↪diff |   diff_pc | unit   |
|---------------------------------------+-----------+--------+-------------+------
↪--+----------+--------|
| continuous_integration_101.bin        | size_in_mb |    81   |        39.6 |   41.
↪4 |  104.545  | MB     |
| continuous_integration_101.root       | size_in_mb |    14.3 |         9.4 |    4.
↪9 |   52.1277 | MB     |
| continuous_integration_101_mctruth.root | size_in_mb |    90.3 |        31.9 |   58.
↪4 |  183.072  | MB     |
+---------------------------------------+-----------+--------+-------------+------
↪--+----------+--------+
```

4.8 sv_root_diff

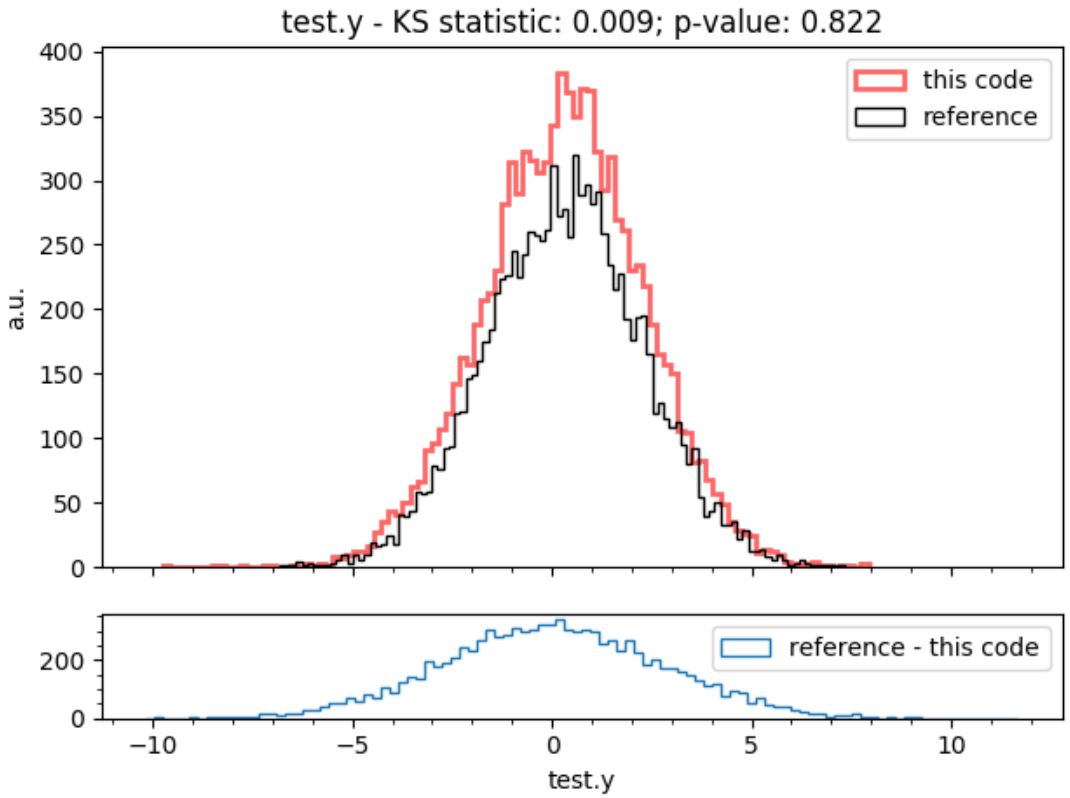
Calculates the difference between two ROOT (https://root.cern.ch/) files. If a difference is present, the command will create plots for the distributions that differ.:

```
sv_root_diff file_under_test reference_file --out-dir <path to output folder (for␣
↪plots etc)>
```

Example output 1 - *test.a* only exists in the reference file:



Example output 2 - *test.y* exists in both, but different random seed:



4.9 sv_version

```
sv_version
scikit-validate version: 0.3.7

sv_version --plain
0.3.7
```

4.10 run-clang-tidy

From https://github.com/llvm-mirror/clang-tools-extra/blob/master/clang-tidy/tool/run-clang-tidy.py

Runs clang-tidy in parallel for the code base:

```
run-clang-tidy <path to code base>
```

CHAPTER

FIVE

REPORTING

5.1 Report.yml

5.1.1 template

5.1.2 download

The `download` entry for a section allows you to specify files that need to be downloaded from either a web URL or from a pipeline job from the pipeline the report will run in. For the latter, the path is of the form `protocol://<name of CI job>/<path to file>`, e.g. `gitlab://test/output/t.png` will download `output/t.png` from the Gitlab CI pipeline job `test`.

The entry in the Report.yml can then be written as

```
download:
  <output path>: <url>
  images/t.png: gitlab://test/output/t.png
```

CHAPTER

SIX

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:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://gitlab.cern.ch/fast-hep/public/scikit-validate/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.

6.1.2 Fix Bugs

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

6.1.3 Implement Features

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

6.1.4 Write Documentation

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

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://gitlab.cern.ch/fast-hep/public/scikit-validate/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 :)

6.2 Get Started!

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

1. Fork the *skvalidate* repo on GitLab.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/skvalidate.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 skvalidate
$ cd skvalidate/
$ 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 skvalidate 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.

6.3 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/kreczko/skvalidate/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_skvalidate
```

6.5 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

SEVEN

CREDITS

7.1 Development Lead

- Faster Analysis Software Taskforce (FAST) <fast-hep@cern.ch>

7.2 Contributors

None yet. Why not be the first?

CHAPTER

EIGHT

SKVALIDATE

8.1 skvalidate package

Top-level package for scikit-validate.

8.1.1 Subpackages

skvalidate.commands package

Package for skvalidate commands.

All modules in this folder are automatically loaded as commands available through *skvalidate*.

Submodules

skvalidate.commands.absolute_to_relative_path module

skvalidate.commands.cpp_check_format module

skvalidate.commands.detect_software_versions module

skvalidate.commands.execute module

skvalidate.commands.file_info module

skvalidate.commands.get_artifact_url module

skvalidate.commands.get_target_branch module

skvalidate.commands.make_demo_report module

skvalidate.commands.make_report module

skvalidate.commands.merge_json module

8.1.2 Submodules

skvalidate.clang_tidy module

Parallel clang-tidy runner.

Runs clang-tidy over all files in a compilation database. Requires clang-tidy and clang-apply-replacements in $PATH.

Example invocations.

- Run clang-tidy on all files in the current working directory with a default set of checks and show warnings in the cpp files and all project headers.

```
run-clang-tidy.py $PWD
```

- Fix all header guards.

```
run-clang-tidy.py -fix -checks=-*,llvm-header-guard
```

- Fix all header guards included from clang-tidy and header guards for clang-tidy headers.

```
run-clang-tidy.py -fix -checks=-*,llvm-header-guard extra/clang-tidy -header-
↪filter=extra/clang-tidy
```

Compilation database setup: http://clang.llvm.org/docs/HowToSetupToolingForLLVM.html

skvalidate.clang_tidy.**apply_fixes**(*args*, *tmpdir*)
Call clang-apply-fixes on a given directory. Deletes the dir when done.

skvalidate.clang_tidy.**check_clang_apply_replacements_binary**(*args*)
Check if invoking supplied clang-apply-replacements binary works.

skvalidate.clang_tidy.**find_compilation_database**(*path*)
Adjust the directory until a compilation database is found.

skvalidate.clang_tidy.**get_tidy_invocation**(*f*, *clang_tidy_binary*, *checks*, *tmpdir*, *build_path*, *header_filter*, *extra_arg*, *extra_arg_before*, *quiet*)
Get a command line for clang-tidy.

skvalidate.clang_tidy.**main**()
Execute the script.

skvalidate.clang_tidy.**run_tidy**(*args*, *tmpdir*, *build_path*, *queue*)
Take filenames out of queue and runs clang-tidy on them.

skvalidate.cli module

skvalidate.git module

skvalidate.skvalidate module

Main module.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX