Developers' Tips and Tricks

Productivity and sanity-preserving tips

In this section we gather some useful advice and tools that may increase your quality-of-life when reviewing pull requests, running unit tests, and so forth. Some of these tricks consist of userscripts that require a browser extension such as TamperMonkey or GreaseMonkey; to set up userscripts you must have one of these extensions installed, enabled and running. We provide userscripts as GitHub gists; to install them, click on the "Raw" button on the gist page.

Folding and unfolding outdated diffs on pull requests

GitHub hides discussions on PRs when the corresponding lines of code have been changed in the mean while. This userscript provides a shortcut (Control-Alt-P at the time of writing but look at the code to be sure) to unfold all such hidden discussions at once, so you can catch up.

Checking out pull requests as remote-tracking branches

In your local fork, add to your `.git/config`, under the `[remote "upstream"]` heading, the line:

```
fetch = +refs/pull/*/head:refs/remotes/upstream/pr/*
```

You may then use `git checkout pr/PR_NUMBER` to navigate to the code of the pull-request with the given number. (Read more in this gist.)

Display code coverage in pull requests

To overlay the code coverage reports generated by the CodeCov continuous integration, consider this browser extension. The coverage of each line will be displayed as a color background behind the line number.

Useful pytest aliases and flags

The full test suite takes fairly long to run. For faster iterations, it is possibly to select a subset of tests using pytest selectors. In particular, one can run a single test based on its node ID:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers)tips.rst`, line 60)

Unknown directive type "prompt".

```
.. prompt:: bash $

  pytest -v sklearn/linear_model/tests/test_logistic.py::test_sparsify
```

or use the -k pytest parameter to select tests based on their name. For instance,:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers)tips.rst`, line 68)

Unknown directive type "prompt".

```
.. prompt:: bash $

  pytest sklearn/tests/test_common.py -v -k LogisticRegression
```

will run all :term:`common tests` for the `LogisticRegression` estimator.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers)tips.rst`, line 72); *backlink*

Unknown interpreted text role "term".

When a unit test fails, the following tricks can make debugging easier:

1. The command line argument `pytest -l` instructs pytest to print the local variables when a failure occurs.
2. The argument `pytest --pdb` drops into the Python debugger on failure. To instead drop into the rich IPython debugger `ipdb`, you may set up a shell alias to:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers)tips.rst`, line 83)

Unknown directive type "prompt".

```
.. prompt:: bash $

    pytest --pdbcls=IPython.terminal.debugger:TerminalPdb --capture no
```

Other *pytest* options that may become useful include:

- `-x` which exits on the first failed test
- `--lf` to rerun the tests that failed on the previous run
- `--ff` to rerun all previous tests, running the ones that failed first
- `-s` so that pytest does not capture the output of `print()` statements
- `--tb=short` or `--tb=line` to control the length of the logs
- `--runxfail` also run tests marked as a known failure (XFAIL) and report errors.

Since our continuous integration tests will error if `FutureWarning` isn't properly caught, it is also recommended to run `pytest` along with the `-Werror::FutureWarning` flag.

Standard replies for reviewing

It may be helpful to store some of these in GitHub's saved replies for reviewing:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers)tips.rst`, line 111)

Unknown directive type "highlight".

```
.. highlight:: none
```

Issue: Usage questions

```
You are asking a usage question. The issue tracker is for bugs and new features. For usage questions, it is recommended to try [St

Unfortunately, we need to close this issue as this issue tracker is a communication tool used for the development of scikit-learn
```

Issue: You're welcome to update the docs

```
Please feel free to offer a pull request updating the documentation if you feel it could be improved.
```

Issue: Self-contained example for bug

```
Please provide [self-contained example code](https://stackoverflow.com/help/mcve), including imports and data (if possible), so th
```

Issue: Software versions

~~~
To help diagnose your issue, please paste the output of:
```py
import sklearn; sklearn.show_versions()
```
Thanks.
~~~

Issue: Code blocks

~~~
Readability can be greatly improved if you [format](https://help.github.com/articles/creating-and-highlighting-code-blocks/) your

    ```python
    print(something)
    ```
generates:
```python
print(something)
```
And:

    ```pytb
    Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
    ImportError: No module named 'hello'
    ```
generates:
```pytb
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'hello'
```
You can edit your issue descriptions and comments at any time to improve readability. This helps maintainers a lot. Thanks!
~~~

Issue/Comment: Linking to code

```
Friendly advice: for clarity's sake, you can link to code like [this](https://help.github.com/articles/creating-a-permanent-link-t
```

Issue/Comment: Linking to comments

```
Please use links to comments, which make it a lot easier to see what you are referring to, rather than just linking to the issue.
```

PR-NEW: Better description and title

```
Thanks for the pull request! Please make the title of the PR more descriptive. The title will become the commit message when this
```

PR-NEW: Fix #

```
Please use "Fix #issueNumber" in your PR description (and you can do it more than once). This way the associated issue gets closed
```

PR-NEW or Issue: Maintenance cost

```
Every feature we include has a [maintenance cost](http://scikit-learn.org/dev/faq.html#why-are-you-so-selective-on-what-algorithms
```

PR-WIP: What's needed before merge?

```
Please clarify (perhaps as a TODO list in the PR description) what work you believe still needs to be done before it can be review
```

PR-WIP: Regression test needed

```
Please add a [non-regression test](https://en.wikipedia.org/wiki/Non-regression_testing) that would fail at main but pass in this
```

PR-WIP: PEP8

```
You have some [PEP8](https://www.python.org/dev/peps/pep-0008/) violations, whose details you can see in the Circle CI `lint` job
```

PR-MRG: Patience

```
Before merging, we generally require two core developers to agree that your pull request is desirable and ready. [Please be patien
```

PR-MRG: Add to what's new

```
Please add an entry to the change log at `doc/whats_new/v*.rst`. Like the other entries there, please reference this pull request
```

PR: Don't change unrelated

```
Please do not change unrelated lines. It makes your contribution harder to review and may introduce merge conflicts to other pull
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)tips.rst`, line 225)

Unknown directive type "highlight".

```
.. highlight:: default
```

Debugging memory errors in Cython with valgrind

While python/numpy's built-in memory management is relatively robust, it can lead to performance penalties for some routines. For this reason, much of the high-performance code in scikit-learn is written in cython. This performance gain comes with a tradeoff, however: it is very easy for memory bugs to crop up in cython code, especially in situations where that code relies heavily on pointer arithmetic.

Memory errors can manifest themselves a number of ways. The easiest ones to debug are often segmentation faults and related glibc errors. Uninitialized variables can lead to unexpected behavior that is difficult to track down. A very useful tool when debugging these sorts of errors is valgrind.

Valgrind is a command-line tool that can trace memory errors in a variety of code. Follow these steps:

1. Install valgrind on your system.
2. Download the python valgrind suppression file: valgrind-python.supp.
3. Follow the directions in the README.valgrind file to customize your python suppressions. If you don't, you will have spurious output coming related to the python interpreter instead of your own code.
4. Run valgrind as follows:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)tips.rst`, line 257)

Unknown directive type "prompt".

```
.. prompt:: bash $

  valgrind -v --suppressions=valgrind-python.supp python my_test_script.py
```

The result will be a list of all the memory-related errors, which reference lines in the C-code generated by cython from your .pyx file. If you examine the referenced lines in the .c file, you will see comments which indicate the corresponding location in your .pyx source file. Hopefully the output will give you clues as to the source of your memory error.

For more information on valgrind and the array of options it has, see the tutorials and documentation on the valgrind web site.

Building and testing for the ARM64 platform on a x86_64 machine

ARM-based machines are a popular target for mobile, edge or other low-energy deployments (including in the cloud, for instance on Scaleway or AWS Graviton).

Here are instructions to setup a local dev environment to reproduce ARM-specific bugs or test failures on a x86_64 host laptop or workstation. This is based on QEMU user mode emulation using docker for convenience (see https://github.com/multiarch/qemu-user-static).

Note

The following instructions are illustrated for ARM64 but they also apply to ppc64le, after changing the Docker image and Miniforge paths appropriately.

Prepare a folder on the host filesystem and download the necessary tools and source code:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers) tips.rst`, **line 296**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    mkdir arm64
    pushd arm64
    wget https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-aarch64.sh
    git clone https://github.com/scikit-learn/scikit-learn.git
```

Use docker to install QEMU user mode and run an ARM64v8 container with access to your shared folder under the */io* mount point:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers) tips.rst`, **line 306**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
    docker run -v`pwd`:/io --rm -it arm64v8/ubuntu /bin/bash
```

In the container, install miniforge3 for the ARM64 (a.k.a. aarch64) architecture:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers) tips.rst`, **line 314**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    bash Miniforge3-Linux-aarch64.sh
    # Choose to install miniforge3 under: `/io/miniforge3`
```

Whenever you restart a new container, you will need to reinit the conda env previously installed under */io/miniforge3*:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main) (doc) (developers) tips.rst`, **line 322**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    /io/miniforge3/bin/conda init
    source /root/.bashrc
```

as the */root* home folder is part of the ephemeral docker container. Every file or directory stored under */io* is persistent on the other hand.

You can then build scikit-learn as usual (you will need to install compiler tools and dependencies using apt or conda as usual). Building scikit-learn takes a lot of time because of the emulation layer, however it needs to be done only once if you put the scikit-learn folder under the */io* mount point.

Then use pytest to run only the tests of the module you are interested in debugging.