Contributing

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 7)

Unknown directive type "currentmodule".

```
.. currentmodule:: sklearn
```

This project is a community effort, and everyone is welcome to contribute.

The project is hosted on https://github.com/scikit-learn/scikit-learn

The decision making process and governance structure of scikit-learn is laid out in the governance document: :ref:`governance`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 14); *backlink*

Unknown interpreted text role "ref".

Scikit-learn is somewhat :ref:`selective <selectiveness>` when it comes to adding new algorithms, and the best way to contribute and to help the project is to start working on known issues. See :ref:`new_contributors` to get started.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 17); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 17); *backlink*

Unknown interpreted text role "ref".

Our community, our values

We are a community based on openness and friendly, didactic, discussions.

We aspire to treat everybody equally, and value their contributions. We are particularly seeking people from underrepresented backgrounds in Open Source Software and scikit-learn in particular to participate and contribute their expertise and experience.

Decisions are made based on technical merit and consensus.

Code is not the only way to help the project. Reviewing pull requests, answering questions to help others on mailing lists or issues, organizing and teaching tutorials, working on the website, improving the documentation, are all priceless contributions.

We abide by the principles of openness, respect, and consideration of others of the Python Software Foundation: https://www.python.org/psf/codeofconduct/

In case you experience issues using this package, do not hesitate to submit a ticket to the GitHub issue tracker. You are also welcome to post feature requests or pull requests.

Ways to contribute

There are many ways to contribute to scikit-learn, with the most common ones being contribution of code or documentation to the project. Improving the documentation is no less important than improving the library itself. If you find a typo in the documentation, or have made improvements, do not hesitate to send an email to the mailing list or preferably submit a GitHub pull request. Full documentation can be found under the doc/ directory.

But there are many other ways to help. In particular helping to :ref:`improve, triage, and investigate issues <bug_triaging>` and :ref:`reviewing other developers' pull requests <code_review>` are very valuable contributions that decrease the burden on the project maintainers.

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

Unknown interpreted text role "ref".

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

Unknown interpreted text role "ref".

Another way to contribute is to report issues you're facing, and give a "thumbs up" on issues that others reported and that are relevant to you. It also helps us if you spread the word: reference the project from your blog and articles, link to it from your website, or simply star to say "I use it":

In case a contribution/issue involves changes to the API principles or changes to dependencies or supported versions, it must be backed by a :ref:`slep`, where a SLEP must be submitted as a pull-request to enhancement proposals using the SLEP template and follows the decision-making process outlined in :ref:`governance`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 71); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 71); *backlink*

Unknown interpreted text role "ref".

Star

Contributing to related projects

Scikit-learn thrives in an ecosystem of several related projects, which also may have relevant issues to work on, including smaller projects such as:

- scikit-learn-contrib
- joblib
- sphinx-gallery
- numpydoc
- liac-arff

and larger projects:

- numpy
- scipy
- matplotlib
- and so on.

Look for issues marked "help wanted" or similar. Helping these projects may help Scikit-learn too. See also :ref:`related_projects`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 103); *backlink*

Unknown interpreted text role "ref".

Submitting a bug report or a feature request

We use GitHub issues to track all bugs and feature requests; feel free to open an issue if you have found a bug or wish to see a feature implemented.

In case you experience issues using this package, do not hesitate to submit a ticket to the Bug Tracker. You are also welcome to post feature requests or pull requests.

It is recommended to check that your issue complies with the following rules before submitting:

- Verify that your issue is not being currently addressed by other issues or pull requests.
- If you are submitting an algorithm or feature request, please verify that the algorithm fulfills our new algorithm requirements.
- If you are submitting a bug report, we strongly encourage you to follow the guidelines in :ref:`filing_bugs`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 131); *backlink*

Unknown interpreted text role "ref".

How to make a good bug report

When you submit an issue to Github, please do your best to follow these guidelines! This will make it a lot easier to provide you with good feedback:

- The ideal bug report contains a :ref:`short reproducible code snippet <minimal_reproducer>`, this way anyone can try to reproduce the bug easily (see this for more details). If your snippet is longer than around 50 lines, please link to a gist or a github repo.

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 144); *backlink***
  >
  > Unknown interpreted text role "ref".

- If not feasible to include a reproducible snippet, please be specific about what **estimators and/or functions are involved and the shape of the data**.
- If an exception is raised, please **provide the full traceback**.
- Please include your **operating system type and version number**, as well as your **Python, scikit-learn, numpy, and scipy versions**. This information can be found by running the following code snippet:

```
>>> import sklearn
>>> sklearn.show_versions()  # doctest: +SKIP
```

- Please ensure all **code snippets and error messages are formatted in appropriate code blocks**. See Creating and highlighting code blocks for more details.

If you want to help curate issues, read :ref:`the following <bug_triaging>`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 168); *backlink***
>
> Unknown interpreted text role "ref".

Contributing code

> **Note**
>
> To avoid duplicating work, it is highly advised that you search through the issue tracker and the PR list. If in doubt about duplicated work, or if you want to work on a non-trivial feature, it's recommended to first open an issue in the issue tracker to get some feedbacks from core developers.
>
> One easy way to find an issue to work on is by applying the "help wanted" label in your search. This lists all the issues that have been unclaimed so far. In order to claim an issue for yourself, please comment exactly `/take` on it for the CI to automatically assign the issue to you.

Video resources

These videos are step-by-step introductions on how to contribute to scikit-learn, and are a great companion to the following text guidelines. Please make sure to still check our guidelines below, since they describe our latest up-to-date workflow.

- Crash Course in Contributing to Scikit-Learn & Open Source Projects: Video, Transcript
- Example of Submitting a Pull Request to scikit-learn: Video, Transcript
- Sprint-specific instructions and practical tips: Video, Transcript
- 3 Components of Reviewing a Pull Request: Video, Transcript

> **Note**
>
> In January 2021, the default branch name changed from `master` to `main` for the scikit-learn GitHub repository to use more inclusive terms. These videos were created prior to the renaming of the branch. For contributors who are viewing these videos to set up their working environment and submitting a PR, `master` should be replaced to `main`.

How to contribute

The preferred way to contribute to scikit-learn is to fork the main repository on GitHub, then submit a "pull request" (PR).

In the first few steps, we explain how to locally install scikit-learn, and how to set up your git repository:

1. Create an account on GitHub if you do not already have one.
2. Fork the project repository: click on the 'Fork' button near the top of the page. This creates a copy of the code under your account on the GitHub user account. For more details on how to fork a repository see this guide.
3. Clone your fork of the scikit-learn repo from your GitHub account to your local disk:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 245)

Unknown directive type "prompt".

```
.. prompt:: bash $

   git clone git@github.com:YourLogin/scikit-learn.git  # add --depth 1 if your connection is
   cd scikit-learn
```

3. Follow steps 2-7 in :ref:`install_bleeding_edge` to build scikit-learn in development mode and return to this document.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 250); *backlink*

Unknown interpreted text role "ref".

4. Install the development dependencies:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 255)

Unknown directive type "prompt".

```
.. prompt:: bash $

    pip install pytest pytest-cov flake8 mypy numpydoc black==22.3.0
```

5. Add the `upstream` remote. This saves a reference to the main scikit-learn repository, which you can use to keep your repository synchronized with the latest changes:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 265)

Unknown directive type "prompt".

```
.. prompt:: bash $

    git remote add upstream git@github.com:scikit-learn/scikit-learn.git
```

6. Check that the *upstream* and *origin* remote aliases are configured correctly by running *git remote -v* which should display:

```
origin  git@github.com:YourLogin/scikit-learn.git (fetch)
origin  git@github.com:YourLogin/scikit-learn.git (push)
upstream        git@github.com:scikit-learn/scikit-learn.git (fetch)
upstream        git@github.com:scikit-learn/scikit-learn.git (push)
```

You should now have a working installation of scikit-learn, and your git repository properly configured. The next steps now describe the process of modifying code and submitting a PR:

7. Synchronize your `main` branch with the `upstream/main` branch, more details on GitHub Docs:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 284)

Unknown directive type "prompt".

```
.. prompt:: bash $

    git checkout main
    git fetch upstream
    git merge upstream/main
```

8. Create a feature branch to hold your development changes:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 292)

Unknown directive type "prompt".

```
.. prompt:: bash $

    git checkout -b my_feature
```

and start making changes. Always use a feature branch. It's good practice to never work on the `main` branch!

9. (**Optional**) Install pre-commit to run code style checks before each commit:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 302)

Unknown directive type "prompt".

```
.. prompt:: bash $

    pip install pre-commit
    pre-commit install
```

pre-commit checks can be disabled for a particular commit with *git commit -n*.

10. Develop the feature on your feature branch on your computer, using Git to do the version control. When you're done editing, add changed files using `git add` and then `git commit`:

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

Unknown directive type "prompt".

```
.. prompt:: bash $

    git add modified_files
    git commit
```

to record your changes in Git, then push the changes to your GitHub account with:

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

Unknown directive type "prompt".

```
.. prompt:: bash $

   git push -u origin my_feature
```

11. Follow these instructions to create a pull request from your fork. This will send an email to the committers. You may want to consider sending an email to the mailing list for more visibility.

Note

If you are modifying a Cython module, you have to re-compile after modifications and before testing them:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 337)

Unknown directive type "prompt".

```
.. prompt:: bash $

    pip install --no-build-isolation -e .
```

Use the `--no-build-isolation` flag to avoid compiling the whole project each time, only the files you have modified.

It is often helpful to keep your local feature branch synchronized with the latest changes of the main scikit-learn repository:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 347)

Unknown directive type "prompt".

```
.. prompt:: bash $

    git fetch upstream
    git merge upstream/main
```

Subsequently, you might need to solve the conflicts. You can refer to the Git documentation related to resolving merge conflict using the command line.

Learning git:

The Git documentation and http://try.github.io are excellent resources to get started with git, and understanding all of the commands shown here.

Pull request checklist

Before a PR can be merged, it needs to be approved by two core developers. Please prefix the title of your pull request with `[MRG]` if the contribution is complete and should be subjected to a detailed review. An incomplete contribution -- where you expect to do more work before receiving a full review -- should be prefixed `[WIP]` (to indicate a work in progress) and changed to `[MRG]` when it matures. WIPs may be useful to: indicate you are working on something to avoid duplicated work, request broad review of functionality or API, or seek collaborators. WIPs often benefit from the inclusion of a task list in the PR description.

In order to ease the reviewing process, we recommend that your contribution complies with the following rules before marking a PR as `[MRG]`. The **bolded** ones are especially important:

1. **Give your pull request a helpful title** that summarises what your contribution does. This title will often become the commit message once merged so it should summarise your contribution for posterity. In some cases "Fix <ISSUE TITLE>" is enough. "Fix #<ISSUE NUMBER>" is never a good title.

2. **Make sure your code passes the tests**. The whole test suite can be run with *pytest*, but it is usually not recommended since it takes a long time. It is often enough to only run the test related to your changes: for example, if you changed something in *sklearn/linear_model/_logistic.py*, running the following commands will usually be enough:

   - *pytest sklearn/linear_model/_logistic.py* to make sure the doctest examples are correct
   - *pytest sklearn/linear_model/tests/test_logistic.py* to run the tests specific to the file
   - *pytest sklearn/linear_model* to test the whole :mod:`~sklearn.linear_model` module

     > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 401); *backlink***
     >
     > Unknown interpreted text role "mod".

   - *pytest doc/modules/linear_model.rst* to make sure the user guide examples are correct.
   - *pytest sklearn/tests/test_common.py -k LogisticRegression* to run all our estimator checks (specifically for *LogisticRegression*, if that's the estimator you changed).

   There may be other failing tests, but they will be caught by the CI so you don't need to run the whole test suite locally. For guidelines on how to use `pytest` efficiently, see the :ref:`pytest_tips`.

   > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 409); *backlink***
   >
   > Unknown interpreted text role "ref".

3. **Make sure your code is properly commented and documented**, and **make sure the documentation renders properly**. To build the documentation, please refer to our :ref:`contribute_documentation` guidelines. The CI will also build the docs: please refer to :ref:`generated_doc_CI`.

   > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 413); *backlink***
   >
   > Unknown interpreted text role "ref".

   > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 413); *backlink***
   >
   > Unknown interpreted text role "ref".

4. **Tests are necessary for enhancements to be accepted**. Bug-fixes or new features should be provided with non-regression tests. These tests verify the correct behavior of the fix or feature. In this manner, further modifications on the code

base are granted to be consistent with the desired behavior. In the case of bug fixes, at the time of the PR, the non-regression tests should fail for the code base in the `main` branch and pass for the PR code.

5. Run *black* to auto-format your code.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 430)

   Unknown directive type "prompt".

   ```
   .. prompt:: bash $

       black .
   ```

   See black's editor integration documentation to configure your editor to run *black*.

6. **Make sure that your PR does not add PEP8 violations**. To check the code that you changed, you can run the following command (see :ref:`above <upstream>` to set up the `upstream` remote):

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 438); *backlink*

   Unknown interpreted text role "ref".

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 442)

   Unknown directive type "prompt".

   ```
   .. prompt:: bash $

       git diff upstream/main -u -- "*.py" | flake8 --diff
   ```

   or *make flake8-diff* which should work on unix-like system.

7. Follow the :ref:`coding-guidelines`.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 448); *backlink*

   Unknown interpreted text role "ref".

8. When applicable, use the validation tools and scripts in the `sklearn.utils` submodule. A list of utility routines available for developers can be found in the :ref:`developers-utils` page.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 451); *backlink*

   Unknown interpreted text role "ref".

9. Often pull requests resolve one or more other issues (or pull requests). If merging your pull request means that some other issues/PRs should be closed, you should use keywords to create link to them (e.g., `Fixes #1234`; multiple issues/PRs are allowed as long as each one is preceded by a keyword). Upon merging, those issues/PRs will automatically be closed by GitHub. If your pull request is simply related to some other issues/PRs, create a link to them without using the keywords (e.g., `See also #1234`).

10. PRs should often substantiate the change, through benchmarks of performance and efficiency (see :ref:`monitoring_performances`) or through examples of usage. Examples also illustrate the features and intricacies of the library to users. Have a look at other examples in the examples/ directory for reference. Examples should demonstrate why the new functionality is useful in practice and, if possible, compare it to other methods available in scikit-learn.

    System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 465); *backlink*

    Unknown interpreted text role "ref".

11. New features have some maintenance overhead. We expect PR authors to take part in the maintenance for the code they

submit, at least initially. New features need to be illustrated with narrative documentation in the user guide, with small code snippets. If relevant, please also add references in the literature, with PDF links when possible.

12. The user guide should also include expected time and space complexity of the algorithm and scalability, e.g. "this algorithm can scale to a large number of samples > 100000, but does not scale in dimensionality: n_features is expected to be lower than 100".

You can also check our :ref:`code_review` to get an idea of what reviewers will expect.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 486); *backlink*

Unknown interpreted text role "ref".

You can check for common programming errors with the following tools:

- Code with a good unittest coverage (at least 80%, better 100%), check with:

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 494)

  Unknown directive type "prompt".

  ```
  .. prompt:: bash $

    pip install pytest pytest-cov
    pytest --cov sklearn path/to/tests_for_package
  ```

  see also :ref:`testing_coverage`

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 499); *backlink*

  Unknown interpreted text role "ref".

  Run static analysis with *mypy*:

  System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 503)

  Unknown directive type "prompt".

  ```
  .. prompt:: bash $

      mypy sklearn
  ```

  must not produce new errors in your pull request. Using *# type: ignore* annotation can be a workaround for a few cases that are not supported by mypy, in particular,

  - when importing C or Cython modules
  - on properties with decorators

Bonus points for contributions that include a performance analysis with a benchmark script and profiling output (see :ref:`monitoring_performances`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 514); *backlink*

Unknown interpreted text role "ref".

Also check out the :ref:`performance-howto` guide for more details on profiling and Cython optimizations.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 517); *backlink*

Unknown interpreted text role "ref".

Note

The current state of the scikit-learn code base is not compliant with all of those guidelines, but we expect that enforcing those constraints on all new contributions will get the overall code base quality in the right direction.

Note

For two very well documented and more detailed guides on development workflow, please pay a visit to the Scipy Development Workflow - and the Astropy Workflow for Developers sections.

Continuous Integration (CI)

- Azure pipelines are used for testing scikit-learn on Linux, Mac and Windows, with different dependencies and settings.
- CircleCI is used to build the docs for viewing, for linting with flake8, and for testing with ARM64 / aarch64 on Linux

Please note that if one of the following markers appear in the latest commit message, the following actions are taken.

Commit Message Marker	Action Taken by CI
[ci skip]	CI is skipped completely
[cd build]	CD is run (wheels and source distribution are built)
[cd build gh]	CD is run only for GitHub Actions
[lint skip]	Azure pipeline skips linting
[scipy-dev]	Build & test with our dependencies (numpy, scipy, etc ...) development builds
[icc-build]	Build & test with the Intel C compiler (ICC)
[pypy]	Build & test with PyPy
[doc skip]	Docs are not built
[doc quick]	Docs built, but excludes example gallery plots
[doc build]	Docs built including example gallery plots (very long)

Note that, by default, the documentation is built but only the examples that are directly modified by the pull request are executed.

Stalled pull requests

As contributing a feature can be a lengthy process, some pull requests appear inactive but unfinished. In such a case, taking them over is a great service for the project.

A good etiquette to take over is:

- **Determine if a PR is stalled**
  - A pull request may have the label "stalled" or "help wanted" if we have already identified it as a candidate for other contributors.
  - To decide whether an inactive PR is stalled, ask the contributor if she/he plans to continue working on the PR in the near future. Failure to respond within 2 weeks with an activity that moves the PR forward suggests that the PR is stalled and will result in tagging that PR with "help wanted".

    Note that if a PR has received earlier comments on the contribution that have had no reply in a month, it is safe to assume that the PR is stalled and to shorten the wait time to one day.

    After a sprint, follow-up for un-merged PRs opened during sprint will be communicated to participants at the sprint, and those PRs will be tagged "sprint". PRs tagged with "sprint" can be reassigned or declared stalled by sprint leaders.
- **Taking over a stalled PR**: To take over a PR, it is important to comment on the stalled PR that you are taking over and to link from the new PR to the old one. The new PR should be created by pulling from the old one.

Stalled and Unclaimed Issues

Generally speaking, issues which are up for grabs will have a "help wanted". tag. However, not all issues which need contributors will have this tag, as the "help wanted" tag is not always up-to-date with the state of the issue. Contributors can find issues which are still up for grabs using the following guidelines:

- First, to **determine if an issue is claimed**:
  - Check for linked pull requests
  - Check the conversation to see if anyone has said that they're working on creating a pull request
- If a contributor comments on an issue to say they are working on it, a pull request is expected within 2 weeks (new contributor) or 4 weeks (contributor or core dev), unless an larger time frame is explicitly given. Beyond that time, another contributor can take the issue and make a pull request for it. We encourage contributors to comment directly on the stalled or unclaimed issue to let community members know that they will be working on it.
- If the issue is linked to a :ref:`stalled pull request <stalled_pull_request>`, we recommend that contributors follow the procedure described in the :ref:`stalled_pull_request` section rather than working directly on the issue.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst`, **line 625**); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 625); *backlink*

Unknown interpreted text role "ref".

Issues for New Contributors

New contributors should look for the following tags when looking for issues. We strongly recommend that new contributors tackle "easy" issues first: this helps the contributor become familiar with the contribution workflow, and for the core devs to become acquainted with the contributor; besides which, we frequently underestimate how easy an issue is to solve!

good first issue tag

A great way to start contributing to scikit-learn is to pick an item from the list of good first issues in the issue tracker. Resolving these issues allow you to start contributing to the project without much prior knowledge. If you have already contributed to scikit-learn, you should look at Easy issues instead.

Easy tag

If you have already contributed to scikit-learn, another great way to contribute to scikit-learn is to pick an item from the list of Easy issues in the issue tracker. Your assistance in this area will be greatly appreciated by the more experienced developers as it helps free up their time to concentrate on other issues.

help wanted tag

We often use the help wanted tag to mark issues regardless of difficulty. Additionally, we use the help wanted tag to mark Pull Requests which have been abandoned by their original contributor and are available for someone to pick up where the original contributor left off. The list of issues with the help wanted tag can be found here.

Note that not all issues which need contributors will have this tag.

Documentation

We are glad to accept any sort of documentation: function docstrings, reStructuredText documents (like this one), tutorials, etc. reStructuredText documents live in the source code repository under the `doc/` directory.

You can edit the documentation using any text editor, and then generate the HTML output by typing `make` from the `doc/` directory. Alternatively, `make html` may be used to generate the documentation **with** the example gallery (which takes quite some time). The resulting HTML files will be placed in `_build/html/stable` and are viewable in a web browser.

Building the documentation

First, make sure you have :ref:`properly installed <install_bleeding_edge>` the development version.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 688); *backlink*

Unknown interpreted text role "ref".

Building the documentation requires installing some additional packages:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 696)

Unknown directive type "prompt".

```
.. prompt:: bash $

    pip install sphinx sphinx-gallery numpydoc matplotlib Pillow pandas \
                scikit-image packaging seaborn sphinx-prompt \
                sphinxext-opengraph
```

To build the documentation, you need to be in the `doc` folder:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 704)

Unknown directive type "prompt".

```
.. prompt:: bash $

    cd doc
```

In the vast majority of cases, you only need to generate the full web site, without the example gallery:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, **line 711**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    make
```

The documentation will be generated in the `_build/html/stable` directory. To also generate the example gallery you can use:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, **line 718**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    make html
```

This will run all the examples, which takes a while. If you only want to generate a few examples, you can use:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, **line 725**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    EXAMPLES_PATTERN=your_regex_goes_here make html
```

This is particularly useful if you are modifying a few examples.

Set the environment variable *NO_MATHJAX=1* if you intend to view the documentation in an offline setting.

To build the PDF manual, run:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, **line 736**)

Unknown directive type "prompt".

```
.. prompt:: bash $

    make latexpdf
```

Warning

Sphinx version

While we do our best to have the documentation build under as many versions of Sphinx as possible, the different versions tend to behave slightly differently. To get the best results, you should use the same version as the one we used on CircleCI. Look at this github search to know the exact version.

Guidelines for writing documentation

It is important to keep a good compromise between mathematical and algorithmic details, and give intuition to the reader on what the algorithm does.

Basically, to elaborate on the above, it is best to always start with a small paragraph with a hand-waving explanation of what the method does to the data. Then, it is very helpful to point out why the feature is useful and when it should be used - the latter also including "big O" ($O(g(n))$) complexities of the algorithm, as opposed to just *rules of thumb*, as the latter can be very machine-dependent. If those complexities are not available, then rules of thumb may be provided instead.

Secondly, a generated figure from an example (as mentioned in the previous paragraph) should then be included to further provide some intuition.

Next, one or two small code examples to show its use can be added.

Next, any math and equations, followed by references, can be added to further the documentation. Not starting the documentation with the maths makes it more friendly towards users that are just interested in what the feature will do, as opposed to how it works "under the hood".

Finally, follow the formatting rules below to make it consistently good:

- Add "See Also" in docstrings for related classes/functions.
- "See Also" in docstrings should be one line per reference, with a colon and an explanation, for example:

```
See Also
--------
SelectKBest : Select features based on the k highest scores.
SelectFpr : Select features based on a false positive rate test.
```

- When documenting the parameters and attributes, here is a list of some well-formatted examples:

```
n_clusters : int, default=3
    The number of clusters detected by the algorithm.

some_param : {'hello', 'goodbye'}, bool or int, default=True
    The parameter description goes here, which can be either a string
    literal (either `hello` or `goodbye`), a bool, or an int. The default
    value is True.

array_parameter : {array-like, sparse matrix} of shape (n_samples, n_features) or (n_samples,)
    This parameter accepts data in either of the mentioned forms, with one
    of the mentioned shapes. The default value is
    `np.ones(shape=(n_samples,))`.

list_param : list of int

typed_ndarray : ndarray of shape (n_samples,), dtype=np.int32

sample_weight : array-like of shape (n_samples,), default=None

multioutput_array : ndarray of shape (n_samples, n_classes) or list of such arrays
```

  In general have the following in mind:

  1. Use Python basic types. (`bool` instead of `boolean`)
  2. Use parenthesis for defining shapes: `array-like of shape (n_samples,)` or `array-like of shape (n_samples, n_features)`
  3. For strings with multiple options, use brackets: `input: {'log', 'squared', 'multinomial'}`
  4. 1D or 2D data can be a subset of `{array-like, ndarray, sparse matrix, dataframe}`. Note that `array-like` can also be a `list`, while `ndarray` is explicitly only a `numpy.ndarray`.
  5. Specify `dataframe` when "frame-like" features are being used, such as the column names.
  6. When specifying the data type of a list, use `of` as a delimiter: `list of int`. When the parameter supports arrays giving details about the shape and/or data type and a list of such arrays, you can use one of `array-like of shape (n_samples,) or list of such arrays`.
  7. When specifying the dtype of an ndarray, use e.g. `dtype=np.int32` after defining the shape: `ndarray of shape (n_samples,), dtype=np.int32`. You can specify multiple dtype as a set: `array-like of shape (n_samples,), dtype={np.float64, np.float32}`. If one wants to mention arbitrary precision, use *integral* and *floating* rather than the Python dtype *int* and *float*. When both *int* and *floating* are supported, there is no need to specify the dtype.
  8. When the default is `None`, `None` only needs to be specified at the end with `default=None`. Be sure to include in the docstring, what it means for the parameter or attribute to be `None`.

- For unwritten formatting rules, try to follow existing good works:

  - When bibliographic references are available with arxiv or Digital Object Identifier identification numbers, use the sphinx directives *:arxiv:* or *:doi:*. For example, see references in :ref:`Spectral Clustering Graphs <spectral_clustering_graph>`.

    System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 841); *backlink*

    Unknown interpreted text role "ref".

  - For "References" in docstrings, see the Silhouette Coefficient (:func:`sklearn.metrics.silhouette_score`).

    System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 845); *backlink*

    Unknown interpreted text role "func".

- When editing reStructuredText (`.rst`) files, try to keep line length under 80 characters when possible (exceptions include links and tables).
- Do not modify sphinx labels as this would break existing cross references and external links pointing to specific sections in the scikit-learn documentation.
- Before submitting your pull request check if your modifications have introduced new sphinx warnings and try to fix them.

Generated documentation on CircleCI

When you change the documentation in a pull request, CircleCI automatically builds it. To view the documentation generated by CircleCI, simply go at the bottom of your PR page and look for the "ci/circleci: doc artifact" link.

Testing and improving test coverage

High-quality unit testing is a corner-stone of the scikit-learn development process. For this purpose, we use the pytest package. The tests are functions appropriately named, located in *tests* subdirectories, that check the validity of the algorithms and the different options of the code.

Running *pytest* in a folder will run all the tests of the corresponding subpackages. For a more detailed *pytest* workflow, please refer to the :ref:`pr_checklist`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 879); *backlink***
>
> Unknown interpreted text role "ref".

We expect code coverage of new features to be at least around 90%.

Writing matplotlib related tests

Test fixtures ensure that a set of tests will be executing with the appropriate initialization and cleanup. The scikit-learn test suite implements a fixture which can be used with `matplotlib`.

`pyplot`
: The `pyplot` fixture should be used when a test function is dealing with `matplotlib`. `matplotlib` is a soft dependency and is not required. This fixture is in charge of skipping the tests if `matplotlib` is not installed. In addition, figures created during the tests will be automatically closed once the test function has been executed.

To use this fixture in a test function, one needs to pass it as an argument:

```
def test_requiring_mpl_fixture(pyplot):
    # you can now safely use matplotlib
```

Workflow to improve test coverage

To test code coverage, you need to install the coverage package in addition to pytest.

1. Run 'make test-coverage'. The output lists for each file the line numbers that are not tested.
2. Find a low hanging fruit, looking at which lines are not tested, write or adapt a test specifically for these lines.
3. Loop.

Monitoring performance

This section is heavily inspired from the pandas documentation.

When proposing changes to the existing code base, it's important to make sure that they don't introduce performance regressions. Scikit-learn uses asv benchmarks to monitor the performance of a selection of common estimators and functions. You can view these benchmarks on the scikit-learn benchmark page. The corresponding benchmark suite can be found in the *scikit-learn/asv_benchmarks* directory.

To use all features of asv, you will need either *conda* or *virtualenv*. For more details please check the asv installation webpage.

First of all you need to install the development version of asv:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 941)**
>
> Unknown directive type "prompt".
>
> ```
> .. prompt:: bash $
>
>     pip install git+https://github.com/airspeed-velocity/asv
> ```

and change your directory to *asv_benchmarks/*:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 947)**
>
> Unknown directive type "prompt".

```
.. prompt:: bash $

  cd asv_benchmarks/
```

The benchmark suite is configured to run against your local clone of scikit-learn. Make sure it is up to date:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 954)

Unknown directive type "prompt".

```
.. prompt:: bash $

  git fetch upstream
```

In the benchmark suite, the benchmarks are organized following the same structure as scikit-learn. For example, you can compare the performance of a specific estimator between `upstream/main` and the branch you are working on:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 962)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv continuous -b LogisticRegression upstream/main HEAD
```

The command uses conda by default for creating the benchmark environments. If you want to use virtualenv instead, use the *-E* flag:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 969)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv continuous -E virtualenv -b LogisticRegression upstream/main HEAD
```

You can also specify a whole module to benchmark:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 975)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv continuous -b linear_model upstream/main HEAD
```

You can replace *HEAD* by any local branch. By default it will only report the benchmarks that have change by at least 10%. You can control this ratio with the *-f* flag.

To run the full benchmark suite, simply remove the *-b* flag :

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 985)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv continuous upstream/main HEAD
```

However this can take up to two hours. The *-b* flag also accepts a regular expression for a more complex subset of benchmarks to run.

To run the benchmarks without comparing to another branch, use the *run* command:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 995)

Unknown directive type "prompt".

```
.. prompt:: bash $
```

```
asv run -b linear_model HEAD^!
```

You can also run the benchmark suite using the version of scikit-learn already installed in your current Python environment:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1002)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv run --python=same
```

It's particularly useful when you installed scikit-learn in editable mode to avoid creating a new environment each time you run the benchmarks. By default the results are not saved when using an existing installation. To save the results you must specify a commit hash:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1011)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv run --python=same --set-commit-hash=<commit hash>
```

Benchmarks are saved and organized by machine, environment and commit. To see the list of all saved benchmarks:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1018)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv show
```

and to see the report of a specific run:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1024)

Unknown directive type "prompt".

```
.. prompt:: bash $

  asv show <commit hash>
```

When running benchmarks for a pull request you're working on please report the results on github.

The benchmark suite supports additional configurable options which can be set in the *benchmarks/config.json* configuration file. For example, the benchmarks can run for a provided list of values for the *n_jobs* parameter.

More information on how to write a benchmark and how to use asv can be found in the asv documentation.

Issue Tracker Tags

All issues and pull requests on the GitHub issue tracker should have (at least) one of the following tags:

Bug / Crash: Something is happening that clearly shouldn't happen. Wrong results as well as unexpected errors from estimators go here.

Cleanup / Enhancement: Improving performance, usability, consistency.

Documentation: Missing, incorrect or sub-standard documentations and examples.

New Feature: Feature requests and pull requests implementing a new feature.

There are four other tags to help new contributors:

good first issue: This issue is ideal for a first contribution to scikit-learn. Ask for help if the formulation is unclear. If you have already contributed to scikit-learn, look at Easy issues instead.

Easy: This issue can be tackled without much prior experience.

Moderate: Might need some knowledge of machine learning or the package, but is still approachable for someone new to the project.

help wanted: This tag marks an issue which currently lacks a contributor or a PR that needs another contributor to take over the work. These issues can range in difficulty, and may not be approachable for new contributors. Note that not all issues which need contributors will have this tag.

Maintaining backwards compatibility

Deprecation

If any publicly accessible method, function, attribute or parameter is renamed, we still support the old one for two releases and issue a deprecation warning when it is called/passed/accessed. E.g., if the function `zero_one` is renamed to `zero_one_loss`, we add the decorator `deprecated` (from `sklearn.utils`) to `zero_one` and call `zero_one_loss` from that function:

```
from ..utils import deprecated

def zero_one_loss(y_true, y_pred, normalize=True):
    # actual implementation
    pass

@deprecated("Function 'zero_one' was renamed to 'zero_one_loss' "
            "in version 0.13 and will be removed in release 0.15. "
            "Default behavior is changed from 'normalize=False' to "
            "'normalize=True'")
def zero_one(y_true, y_pred, normalize=False):
    return zero_one_loss(y_true, y_pred, normalize)
```

If an attribute is to be deprecated, use the decorator `deprecated` on a property. Please note that the `property` decorator should be placed before the `deprecated` decorator for the docstrings to be rendered properly. E.g., renaming an attribute `labels_` to `classes_` can be done as:

```
@deprecated("Attribute `labels_` was deprecated in version 0.13 and "
            "will be removed in 0.15. Use `classes_` instead")
@property
def labels_(self):
    return self.classes_
```

If a parameter has to be deprecated, a `FutureWarning` warning must be raised too. In the following example, k is deprecated and renamed to n_clusters:

```
import warnings

def example_function(n_clusters=8, k='deprecated'):
    if k != 'deprecated':
        warnings.warn("'k' was renamed to n_clusters in version 0.13 and "
                      "will be removed in 0.15.",
                      FutureWarning)
        n_clusters = k
```

When the change is in a class, we validate and raise warning in `fit`:

```
import warnings

class ExampleEstimator(BaseEstimator):
    def __init__(self, n_clusters=8, k='deprecated'):
        self.n_clusters = n_clusters
        self.k = k

    def fit(self, X, y):
        if self.k != 'deprecated':
            warnings.warn("'k' was renamed to n_clusters in version 0.13 and "
                          "will be removed in 0.15.",
                          FutureWarning)
            self._n_clusters = self.k
        else:
            self._n_clusters = self.n_clusters
```

As in these examples, the warning message should always give both the version in which the deprecation happened and the version in which the old behavior will be removed. If the deprecation happened in version 0.x-dev, the message should say deprecation occurred in version 0.x and the removal will be in 0.(x+2), so that users will have enough time to adapt their code to the new behaviour. For example, if the deprecation happened in version 0.18-dev, the message should say it happened in version 0.18 and the old behavior will be removed in version 0.20.

In addition, a deprecation note should be added in the docstring, recalling the same information as the deprecation warning as explained above. Use the `.. deprecated::` directive:

```
.. deprecated:: 0.13
   ``k`` was renamed to ``n_clusters`` in version 0.13 and will be removed
   in 0.15.
```

What's more, a deprecation requires a test which ensures that the warning is raised in relevant cases but not in other cases. The warning should be caught in all other tests (using e.g., `@pytest.mark.filterwarnings`), and there should be no warning in the examples.

Change the default value of a parameter

If the default value of a parameter needs to be changed, please replace the default value with a specific value (e.g., `warn`) and raise `FutureWarning` when users are using the default value. The following example assumes that the current version is 0.20 and that we change the default value of `n_clusters` from 5 (old default for 0.20) to 10 (new default for 0.22):

```
import warnings

def example_function(n_clusters='warn'):
    if n_clusters == 'warn':
        warnings.warn("The default value of n_clusters will change from "
                      "5 to 10 in 0.22.", FutureWarning)
        n_clusters = 5
```

When the change is in a class, we validate and raise warning in `fit`:

```
import warnings

class ExampleEstimator:
    def __init__(self, n_clusters='warn'):
        self.n_clusters = n_clusters

    def fit(self, X, y):
        if self.n_clusters == 'warn':
          warnings.warn("The default value of n_clusters will change from "
                        "5 to 10 in 0.22.", FutureWarning)
          self._n_clusters = 5
```

Similar to deprecations, the warning message should always give both the version in which the change happened and the version in which the old behavior will be removed. The docstring needs to be updated accordingly. We need a test which ensures that the warning is raised in relevant cases but not in other cases. The warning should be caught in all other tests (using e.g., `@pytest.mark.filterwarnings`), and there should be no warning in the examples.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1217)

Unknown directive type "currentmodule".

```
.. currentmodule:: sklearn
```

Code Review Guidelines

Reviewing code contributed to the project as PRs is a crucial component of scikit-learn development. We encourage anyone to start reviewing code of other developers. The code review process is often highly educational for everybody involved. This is particularly appropriate if it is a feature you would like to use, and so can respond critically about whether the PR meets your needs. While each pull request needs to be signed off by two core developers, you can speed up this process by providing your feedback.

> **Note**
>
> The difference between an objective improvement and a subjective nit isn't always clear. Reviewers should recall that code review is primarily about reducing risk in the project. When reviewing code, one should aim at preventing situations which may require a bug fix, a deprecation, or a retraction. Regarding docs: typos, grammar issues and disambiguations are better addressed immediately.

Here are a few important aspects that need to be covered in any code review, from high-level questions to a more detailed check-list.

- Do we want this in the library? Is it likely to be used? Do you, as a scikit-learn user, like the change and intend to use it? Is it in the scope of scikit-learn? Will the cost of maintaining a new feature be worth its benefits?
- Is the code consistent with the API of scikit-learn? Are public functions/classes/parameters well named and intuitively designed?
- Are all public functions/classes and their parameters, return types, and stored attributes named according to scikit-learn conventions and documented clearly?
- Is any new functionality described in the user-guide and illustrated with examples?
- Is every public function/class tested? Are a reasonable set of parameters, their values, value types, and combinations tested? Do the tests validate that the code is correct, i.e. doing what the documentation says it does? If the change is a bug-fix, is a non-regression test included? Look at this to get started with testing in Python.
- Do the tests pass in the continuous integration build? If appropriate, help the contributor understand why tests failed.
- Do the tests cover every line of code (see the coverage report in the build log)? If not, are the lines missing coverage good exceptions?
- Is the code easy to read and low on redundancy? Should variable names be improved for clarity or consistency? Should comments be added? Should comments be removed as unhelpful or extraneous?
- Could the code easily be rewritten to run much more efficiently for relevant settings?

- Is the code backwards compatible with previous versions? (or is a deprecation cycle necessary?)
- Will the new code add any dependencies on other libraries? (this is unlikely to be accepted)
- Does the documentation render properly (see the :ref:`contribute_documentation` section for more details), and are the plots instructive?

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1283); *backlink***
>
> Unknown interpreted text role "ref".

:ref:`saved_replies` includes some frequent comments that reviewers may make.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1287); *backlink***
>
> Unknown interpreted text role "ref".

Communication Guidelines

Reviewing open pull requests (PRs) helps move the project forward. It is a great way to get familiar with the codebase and should motivate the contributor to keep involved in the project. [1]

- Every PR, good or bad, is an act of generosity. Opening with a positive comment will help the author feel rewarded, and your subsequent remarks may be heard more clearly. You may feel good also.
- Begin if possible with the large issues, so the author knows theyâ€™ve been understood. Resist the temptation to immediately go line by line, or to open with small pervasive issues.
- Do not let perfect be the enemy of the good. If you find yourself making many small suggestions that don't fall into the :ref:`code_review`, consider the following approaches:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1304); *backlink***
>
> Unknown interpreted text role "ref".

  - refrain from submitting these;
  - prefix them as "Nit" so that the contributor knows it's OK not to address;
  - follow up in a subsequent PR, out of courtesy, you may want to let the original contributor know.
- Do not rush, take the time to make your comments clear and justify your suggestions.
- You are the face of the project. Bad days occur to everyone, in that occasion you deserve a break: try to take your time and stay offline.

[1] Adapted from the numpy communication guidelines.

Reading the existing code base

Reading and digesting an existing code base is always a difficult exercise that takes time and experience to main. Even though we try to write simple code in general, understanding the code can seem overwhelming at first, given the sheer size of the project. Here is a list of tips that may help make this task easier and faster (in no particular order).

- Get acquainted with the :ref:`api_overview`: understand what :term:`fit`, :term:`predict`, :term:`transform`, etc. are used for.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1330); *backlink***
>
> Unknown interpreted text role "ref".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1330); *backlink***
>
> Unknown interpreted text role "term".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1330); *backlink***

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1330); *backlink*

Unknown interpreted text role "term".

- Before diving into reading the code of a function / class, go through the docstrings first and try to get an idea of what each parameter / attribute is doing. It may also help to stop a minute and think *how would I do this myself if I had to?*
- The trickiest thing is often to identify which portions of the code are relevant, and which are not. In scikit-learn **a lot** of input checking is performed, especially at the beginning of the :term:`fit` methods. Sometimes, only a very small portion of the code is doing the actual job. For example looking at the `fit()` method of :class:`~linear_model.LinearRegression`, what you're looking for might just be the call the `scipy.linalg.lstsq`, but it is buried into multiple lines of input checking and the handling of different kinds of parameters.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1336); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1336); *backlink*

Unknown interpreted text role "class".

- Due to the use of Inheritance, some methods may be implemented in parent classes. All estimators inherit at least from :class:`~base.BaseEstimator`, and from a `Mixin` class (e.g. :class:`~base.ClassifierMixin`) that enables default behaviour depending on the nature of the estimator (classifier, regressor, transformer, etc.).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1345); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1345); *backlink*

Unknown interpreted text role "class".

- Sometimes, reading the tests for a given function will give you an idea of what its intended purpose is. You can use `git grep` (see below) to find all the tests written for a function. Most tests for a specific function/class are placed under the `tests/` folder of the module
- You'll often see code looking like this: `out = Parallel(...)(delayed(some_function)(param) for param in some_iterable)`. This runs `some_function` in parallel using Joblib. `out` is then an iterable containing the values returned by `some_function` for each call.
- We use Cython to write fast code. Cython code is located in `.pyx` and `.pxd` files. Cython code has a more C-like flavor: we use pointers, perform manual memory allocation, etc. Having some minimal experience in C / C++ is pretty much mandatory here.
- Master your tools.
  - With such a big project, being efficient with your favorite editor or IDE goes a long way towards digesting the code base. Being able to quickly jump (or *peek*) to a function/class/attribute definition helps a lot. So does being able to quickly see where a given name is used in a file.
  - git also has some built-in killer features. It is often useful to understand how a file changed over time, using e.g. `git blame` (manual). This can also be done directly on GitHub. `git grep` (examples) is also extremely useful to see every occurrence of a pattern (e.g. a function call or a variable) in the code base.
- Configure *git blame* to ignore the commit that migrated the code style to *black*.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\(scikit-learn-main)(doc)(developers)contributing.rst, line 1383)

Unknown directive type "prompt".

```
.. prompt:: bash $

    git config blame.ignoreRevsFile .git-blame-ignore-revs
```

Find out more information in black's documentation for avoiding ruining git blame.