Maintainer / core-developer information

Releasing

This section is about preparing a major release, incrementing the minor version, or a bug fix release incrementing the patch version. Our convention is that we release one or more release candidates (0.RRrcN) before releasing the final distributions. We follow the PEP101 to indicate release candidates, post, and minor releases.

Before a release

1. Update authors table:

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 20)

   Unknown directive type "prompt".

   ```
   .. prompt:: bash $

       cd build_tools; make authors; cd ..
   ```

   and commit. This is only needed if the authors have changed since the last release. This step is sometimes done independent of the release. This updates the maintainer list and is not the contributor list for the release.
2. Confirm any blockers tagged for the milestone are resolved, and that other issues tagged for the milestone can be postponed.
3. Ensure the change log and commits correspond (within reason!), and that the change log is reasonably well curated. Some tools for these tasks include:
   - `maint_tools/sort_whats_new.py` can put what's new entries into sections. It's not perfect, and requires manual checking of the changes. If the what's new list is well curated, it may not be necessary.
   - The `maint_tools/whats_missing.sh` script may be used to identify pull requests that were merged but likely missing from What's New.
4. Make sure the deprecations, FIXME and TODOs tagged for the release have been taken care of.

Permissions

The release manager requires a set of permissions on top of the usual permissions given to maintainers, which includes:

- *maintainer* role on `scikit-learn` projects on `pypi.org` and `test.pypi.org`, separately.
- become a member of the *scikit-learn* team on conda-forge by editing the `recipe/meta.yaml` file on `https://github.com/conda-forge/scikit-learn-feedstock`

Preparing a release PR

Major version release

Prior to branching please do not forget to prepare a Release Highlights page as a runnable example and check that its HTML rendering looks correct. These release highlights should be linked from the `doc/whats_new/v0.99.rst` file for the new version of scikit-learn.

Releasing the first RC of e.g. version *0.99.0* involves creating the release branch *0.99.X* directly on the main repo, where *X* really is the letter X, **not a placeholder**. The development for the major and minor releases of *0.99* should **also** happen under *0.99.X*. Each release (rc, major, or minor) is a tag under that branch.

This is done only once, as the major and minor releases happen on the same branch:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 77)

Unknown directive type "prompt".

```
.. prompt:: bash $

    # Assuming upstream is an alias for the main scikit-learn repo:
    git fetch upstream main
    git checkout upstream/main
    git checkout -b 0.99.X
    git push --set-upstream upstream 0.99.X
```

Again, *X* is literal here, and *99* is replaced by the release number. The branches are called `0.19.X`, `0.20.X`, etc.

In terms of including changes, the first RC ideally counts as a *feature freeze*. Each coming release candidate and the final release afterwards will include only minor documentation changes and bug fixes. Any major enhancement or feature should be excluded.

Then you can prepare a local branch for the release itself, for instance: `release-0.99.0rc1`, push it to your github fork and open a PR **to the** ***scikit-learn/0.99.X*** **branch**. Copy the :ref:`release_checklist` templates in the description of the Pull Request to track progress.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 93); *backlink*

Unknown interpreted text role "ref".

This PR will be used to push commits related to the release as explained in :ref:`making_a_release`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 98); *backlink*

Unknown interpreted text role "ref".

You can also create a second PR from main and targeting main to increment the `__version__` variable in *sklearn/__init__.py* to increment the dev version. This means while we're in the release candidate period, the latest stable is two versions behind the main branch, instead of one. In this PR targeting main you should also include a new file for the matching version under the `doc/whats_new/` folder so PRs that target the next version can contribute their changelog entries to this file in parallel to the release process.

Minor version release

The minor releases should include bug fixes and some relevant documentation changes only. Any PR resulting in a behavior change which is not a bug fix should be excluded.

First, create a branch, **on your own fork** (to release e.g. *0.99.3*):

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 119)

Unknown directive type "prompt".

```
.. prompt:: bash $

    # assuming main and upstream/main are the same
    git checkout -b release-0.99.3 main
```

Then, create a PR **to the** *scikit-learn/0.99.X* **branch** (not to main!) with all the desired changes:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 127)

Unknown directive type "prompt".

```
.. prompt:: bash $

        git rebase -i upstream/0.99.2
```

Copy the :ref:`release_checklist` templates in the description of the Pull Request to track progress.

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

Unknown interpreted text role "ref".

Do not forget to add a commit updating `sklearn.__version__`.

It's nice to have a copy of the `git rebase -i` log in the PR to help others understand what's included.

Making a release

0. Ensure that you have checked out the branch of the release PR as explained in :ref:`preparing_a_release_pr` above.

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 144); *backlink*

   Unknown interpreted text role "ref".

1. Update docs. Note that this is for the final release, not necessarily for the RC releases. These changes should be made in main and cherry-picked into the release branch, only before the final release.

   - Edit the `doc/whats_new/v0.99.rst` file to add release title and list of contributors. You can retrieve the list of contributor names with:

     ```
     $ git shortlog -s 0.98.33.. | cut -f2- | sort --ignore-case | tr '\n' ';' | sed 's/;/, /g;s/, $//'
     ```

        - For major releases, link the release highlights example from the `doc/whats_new/v0.99.rst` file.
    - Update the release date in `whats_new.rst`
    - Edit the `doc/templates/index.html` to change the 'News' entry of the front page (with the release month as well).
2. On the branch for releasing, update the version number in `sklearn/__init__.py`, the `__version__`.

   For major releases, please add a 0 at the end: *0.99.0* instead of *0.99*.

   For the first release candidate, use the *rc1* suffix on the expected final release number: *0.99.0rc1*.
3. Trigger the wheel builder with the `[cd build]` commit marker using the command:

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 177)

   Unknown directive type "prompt".

   ```
   .. prompt:: bash $

    git commit --allow-empty -m "Trigger wheel builder workflow: [cd build]"
   ```

   The wheel building workflow is managed by GitHub Actions and the results be browsed at: https://github.com/scikit-learn/scikit-learn/actions?query=workflow%3A%22Wheel+builder%22

Note

Before building the wheels, make sure that the `pyproject.toml` file is up to date and using the oldest version of `numpy` for each Python version to avoid ABI incompatibility issues. Moreover, a new line have to be included in the `pyproject.toml` file for each new supported version of Python.

Note

The acronym CD in *[cd build]* stands for Continuous Delivery and refers to the automation used to generate the release artifacts (binary and source packages). This can be seen as an extension to CI which stands for Continuous Integration. The CD workflow on GitHub Actions is also used to automatically create nightly builds and publish packages for the development branch of scikit-learn. See :ref:`install_nightly_builds`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 194); *backlink*

Unknown interpreted text role "ref".

4. Once all the CD jobs have completed successfully in the PR, merge it, again with the *[cd build]* marker in the commit message. This time the results will be uploaded to the staging area.

   You should then be able to upload the generated artifacts (.tar.gz and .whl files) to https://test.pypi.org using the "Run workflow" form for the following GitHub Actions workflow:

   https://github.com/scikit-learn/scikit-learn/actions?query=workflow%3A%22Publish+to+Pypi%22

4.1 You can test the conda-forge builds by submitting a PR to the feedstock repo: https://github.com/conda-forge/scikit-learn-feedstock. If you want to publish an RC release on conda-forge, the PR should target the *rc* branch as opposed to the *master* branch. The two branches need to be kept sync together otherwise.

5. If this went fine, you can proceed with tagging. Proceed with caution. Ideally, tags should be created when you're almost certain that the release is ready, since adding a tag to the main repo can trigger certain automated processes.

   Create the tag and push it (if it's an RC, it can be `0.xx.0rc1` for instance):

   System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 228)

   Unknown directive type "prompt".

   ```
   .. prompt:: bash $

     git tag -a 0.99.0  # in the 0.99.X branch
     git push git@github.com:scikit-learn/scikit-learn.git 0.99.0
   ```
6. Trigger the GitHub Actions workflow again but this time to upload the artifacts to the real https://pypi.org (replace "testpypi" by "pypi" in the "Run workflow" form).
7. Alternatively, it's possible to collect locally the generated binary wheel packages and source tarball and upload them all to PyPI by running the following commands in the scikit-learn source folder (checked out at the release tag):

   **System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst, line 242)

Unknown directive type "prompt".

```
.. prompt:: bash $

    rm -r dist
    pip install -U wheelhouse_uploader twine
    python -m wheelhouse_uploader fetch \
      --version 0.99.0 \
      --local-folder dist \
      scikit-learn \
      https://pypi.anaconda.org/scikit-learn-wheels-staging/simple/scikit-learn/
```

This command will download all the binary packages accumulated in the staging area on the anaconda.org hosting service and put them in your local *./dist* folder.

Check the content of the *./dist* folder: it should contain all the wheels along with the source tarball ("scikit-learn-RRR.tar.gz").

Make sure that you do not have developer versions or older versions of the scikit-learn package in that folder.

Before uploading to pypi, you can test upload to test.pypi.org:

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

Unknown directive type "prompt".

```
.. prompt:: bash $

    twine upload --verbose --repository-url https://test.pypi.org/legacy/ dist/*
```

Upload everything at once to https://pypi.org:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst, line 271)

Unknown directive type "prompt".

```
.. prompt:: bash $

    twine upload dist/*
```

8. For major/minor (not bug-fix release), update the symlink for `stable` and the `latestStable` variable in https://github.com/scikit-learn/scikit-learn.github.io:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst, line 279)

Unknown directive type "prompt".

```
.. prompt:: bash $

    cd /tmp
    git clone --depth 1 --no-checkout git@github.com:scikit-learn/scikit-learn.github.io.git
    cd scikit-learn.github.io
    echo stable > .git/info/sparse-checkout
    git checkout main
    rm stable
    ln -s 0.999 stable
    sed -i "s/latestStable = '.*/latestStable = '0.999';/" versionwarning.js
    git add stable versionwarning.js
    git commit -m "Update stable to point to 0.999"
    git push origin master
```

Release checklist

The following GitHub checklist might be helpful in a release PR:

```
* [ ] update news and what's new date in release branch
* [ ] update news and what's new date and sklearn dev0 version in main branch
* [ ] check that the for the release wheels can be built successfully
* [ ] merge the PR with `[cd build]` commit message to upload wheels to the staging repo
* [ ] upload the wheels and source tarball to https://test.pypi.org
* [ ] create tag on the main github repo
* [ ] confirm bot detected at
  https://github.com/conda-forge/scikit-learn-feedstock and wait for merge
* [ ] upload the wheels and source tarball to PyPI
* [ ] https://github.com/scikit-learn/scikit-learn/releases publish
* [ ] announce on mailing list and on Twitter, and LinkedIn
```

Merging Pull Requests

Individual commits are squashed when a Pull Request (PR) is merged on Github. Before merging,

- the resulting commit title can be edited if necessary. Note that this will rename the PR title by default.
- the detailed description, containing the titles of all the commits, can be edited or deleted.
- for PRs with multiple code contributors care must be taken to keep the *Co-authored-by: name <name@example.com>* tags in the detailed description. This will mark the PR as having multiple co-authors. Whether code contributions are significanly enough to merit co-authorship is left to the maintainer's discretion, same as for the "what's new" entry.

The scikit-learn.org web site

The scikit-learn web site (http://scikit-learn.org) is hosted at GitHub, but should rarely be updated manually by pushing to the https://github.com/scikit-learn/scikit-learn.github.io repository. Most updates can be made by pushing to master (for /dev) or a release branch like 0.99.X, from which Circle CI builds and uploads the documentation automatically.

Travis Cron jobs

From https://docs.travis-ci.com/user/cron-jobs: Travis CI cron jobs work similarly to the cron utility, they run builds at regular scheduled intervals independently of whether any commits were pushed to the repository. Cron jobs always fetch the most recent commit on a particular branch and build the project at that state. Cron jobs can run daily, weekly or monthly, which in practice means up to an hour after the selected time span, and you cannot set them to run at a specific time.

For scikit-learn, Cron jobs are used for builds that we do not want to run in each PR. As an example the build with the dev versions of numpy and scipy is run as a Cron job. Most of the time when this numpy-dev build fail, it is related to a numpy change and not a scikit-learn one, so it would not make sense to blame the PR author for the Travis failure.

The definition of what gets run in the Cron job is done in the .travis.yml config file, exactly the same way as the other Travis jobs. We use a `if: type = cron` filter in order for the build to be run only in Cron jobs.

The branch targeted by the Cron job and the frequency of the Cron job is set via the web UI at https://www.travis-ci.org/scikit-learn/scikit-learn/settings.

Experimental features

The :mod:`sklearn.experimental` module was introduced in 0.21 and contains experimental features / estimators that are subject to change without deprecation cycle.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 367); *backlink*

Unknown interpreted text role "mod".

To create an experimental module, you can just copy and modify the content of enable_hist_gradient_boosting.py, or enable_iterative_imputer.py.

Note

These are permalink as in 0.24, where these estimators are still experimental. They might be stable at the time of reading - hence the permalink. See below for instructions on the transition from experimental to stable.

Note that the public import path must be to a public subpackage (like `sklearn/ensemble` or `sklearn/impute`), not just a `.py` module. Also, the (private) experimental features that are imported must be in a submodule/subpackage of the public subpackage, e.g. `sklearn/ensemble/_hist_gradient_boosting/` or `sklearn/impute/_iterative.py`. This is needed so that pickles still work in the future when the features aren't experimental anymore.

To avoid type checker (e.g. mypy) errors a direct import of experimental estimators should be done in the parent module, protected by the `if typing.TYPE_CHECKING` check. See sklearn/ensemble/__init__.py, or sklearn/impute/__init__.py for an example.

Please also write basic tests following those in test_enable_hist_gradient_boosting.py.

Make sure every user-facing code you write explicitly mentions that the feature is experimental, and add a `# noqa` comment to avoid pep8-related warnings:

```
# To use this experimental feature, we need to explicitly ask for it:
from sklearn.experimental import enable_hist_gradient_boosting  # noqa
from sklearn.ensemble import HistGradientBoostingRegressor
```

For the docs to render properly, please also import `enable_my_experimental_feature` in `doc/conf.py`, else sphinx won't be able to import the corresponding modules. Note that using `from sklearn.experimental import *` **does not work**.

Note that some experimental classes / functions are not included in the :mod:`sklearn.experimental` module: `sklearn.datasets.fetch_openml`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]maintainer.rst`, line 418); *backlink*

Unknown interpreted text role "mod".

Once the feature become stable, remove all *enable_my_experimental_feature* in the scikit-learn code (even feature highlights etc.) and make the *enable_my_experimental_feature* a no-op that just raises a warning: enable_hist_gradient_boosting.py. The file should stay there indefinitely as we don't want to break users code: we just incentivize them to remove that import with the warning.

Also update the tests accordingly: test_enable_hist_gradient_boosting.py.