10. Developer guideline

10.1. Developer utilities

Imbalanced-learn contains a number of utilities to help with development. These are located in `imblearn.utils`, and include tools in a number of categories. All the following functions and classes are in the module `imblearn.utils`.

> **Warning**
>
> These utilities are meant to be used internally within the imbalanced-learn package. They are not guaranteed to be stable between versions of imbalanced-learn. Backports, in particular, will be removed as the imbalanced-learn dependencies evolve.

10.1.1. Validation Tools

These are tools used to check and validate input. When you write a function which accepts arrays, matrices, or sparse matrices as arguments, the following should be used when applicable.

- `check_neighbors_object`: Check the objects is consistent to be a NN.
- `check_target_type`: Check the target types to be conform to the current sam plers.
- `check_sampling_strategy`: Checks that sampling target is onsistent with the type and return a dictionary containing each targeted class with its corresponding number of pixel.

10.1.2. Deprecation

> **Warning**
>
> Apart from `deprecate_parameter` the rest of this section is taken from scikit-learn. Please refer to their original documentation.

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

Edit this page

`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. E.g., renaming an attribute `labels_` to `classes_` can be done as:

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

If a parameter has to be deprecated, use `DeprecationWarning` appropriately. In the following example, k is deprecated and renamed to n_clusters:

```
import warnings

def example_function(n_clusters=8, k=None):
    if k is not None:
        warnings.warn("'k' was renamed to n_clusters in
version 0.13 and "
                      "will be removed in 0.15.",
DeprecationWarning)
        n_clusters = k
```

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). 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.
```

On the top of all the functionality provided by scikit-learn. imbalanced-learn provides `deprecate_parameter`: which is used to deprecate a sampler's parameter (attribute) by another one.

10.1.3. Testing utilities

Currently, imbalanced-learn provide a warning management utility. This feature is going to be merge in pytest and will be removed when the pytest release will have it.

If using Python 2.7 or above, you may use this function as a context manager:

```
>>> import warnings
>>> from imblearn.utils.testing import warns
>>> with warns(RuntimeWarning):
...    warnings.warn("my runtime warning", RuntimeWarning)

>>> with warns(RuntimeWarning):
...    pass
Traceback (most recent call last):
  ...
Failed: DID NOT WARN. No warnings of type
...RuntimeWarning... was emitted...

>>> with warns(RuntimeWarning):
...    warnings.warn(UserWarning)
Traceback (most recent call last):
  ...
Failed: DID NOT WARN. No warnings of type
...RuntimeWarning... was emitted...
```

In the context manager form you may use the keyword argument `match` to assert that the exception matches a text or regex:

```
>>> import warnings
>>> from imblearn.utils.testing import warns
>>> with warns(UserWarning, match='must be 0 or None'):
...     warnings.warn("value must be 0 or None",
UserWarning)

>>> with warns(UserWarning, match=r'must be \d+$'):
...     warnings.warn("value must be 42", UserWarning)

>>> with warns(UserWarning, match=r'must be \d+$'):
...     warnings.warn("this is not here", UserWarning)
Traceback (most recent call last):
  ...
AssertionError: 'must be \d+$' pattern not found in ['this
is not here']
```

10.2. Making a release

This section document the different steps that are necessary to make a new imbalanced-learn release.

10.2.1. Major release

- Update the release note `whats_new/v0.<version number>.rst` by giving a date and removing the status "Under development" from the title.
- Run `bumpversion release`. It will remove the `dev0` tag.
- Commit the change `git commit -am "bumpversion 0.<version number>.0"` (e.g., `git commit -am "bumpversion 0.5.0"`).
- Create a branch for this version (e.g., `git checkout -b 0.<version number>.X`).
- Push the new branch into the upstream remote imbalanced-learn repository.
- Change the `symlink` in the imbalanced-learn website repository such that stable points to the latest release version, i.e, `0.<version number>`. To do this, clone the repository, `run unlink stable`, followed by `ln -s 0.<version number> stable`. To check that this was performed correctly, ensure that stable has the new version number using `ls -l`.
- Return to your imbalanced-learn repository, in the branch `0.<version number>.X`.
- Create the source distribution and wheel: `python setup.py sdist` and `python setup.py bdist_wheel`.
- Upload these file to PyPI using `twine upload dist/*`
- Switch to the `master` branch and run `bumpversion minor`, commit and push on upstream. We are officially at `0.<version number + 1>.0.dev0`.
- Create a GitHub release by clicking on "Draft a new release" here. "Tag version" should be the latest version number (e.g., `0.<version>.0`), "Target" should be the branch for that the release (e.g., `0.<version number>.X`) and "Release title" should be "Version <version number>". Add the notes from the release notes there.
- Add a new `v0.<version number + 1>.rst` file in `doc/whats_new/` and `.. include::` this new file in `doc/whats_new.rst`. Mark the version as the version under development.
- Finally, go to the conda-forge feedstock and a new PR will be created when the feedstock will synchronizing with the PyPI repository. Merge this PR such that we have the binary for `conda` available.

10.2.2. Bug fix release

- Find the commit(s) hash of the bug fix commit you wish to back port using `git log`.
- Checkout the branch for the lastest release, e.g., `git checkout 0.<version number>.X`.
- Append the bug fix commit(s) to the branch using `git cherry-pick <hash>`. Alternatively, you can use interactive rebasing from the `master` branch.
- Bump the version number with bumpversion patch. This will bump the patch version, for example from `0.X.0` to `0.X.* dev0`.
- Mark the current version as a release version (as opposed to `dev` version) with `bumpversion release --allow-dirty`. It will bump the version, for example from `0.X.* dev0` to `0.X.1`.
- Commit the changes with `git commit -am 'bumpversion <new version>'`.
- Push the changes to the release branch in upstream, e.g. `git push <upstream remote> <release branch>`.
- Use the same process as in a major release to upload on PyPI and conda-forge.

<< 9. Dataset loading utilities

11. References >>