CSCD01: Deliverable 3

DEVELOPMENT PROCESS DOCUMENTATION

IMPORTANT LINKS

- Github
- Trello

DEVELOPMENT PROCESS

TASK TRACKING

To keep track of the progress of each task, our team utilizes **`Trello boards`**. We have found that this tool is highly effective in providing real-time updates on the status of each task.

TASK TIME ESTIMATION

We employ **`poker sizing`** for task time estimation, using a **`Fibonacci sequence`** to estimate task complexity, with the numbers 1 through 13 indicating varying degrees of complexity. For example, tasks with a score of 5 or 8 usually represent items such as API design and implementation, bug fixes, and the creation of documentation and test batches for each bug.

ASSIGNMENT OF TASKS

Each task is assigned to two people, with one person responsible for the actual bug fix and the other responsible for writing the validation test batches.

UPDATES IN TASK TRACKING

To ensure we stay on track, we leverage Trello's checklist feature to update our progress and a notification feature to remind us of upcoming deadlines.

CONVENTIONS

Additionally, we have established naming conventions for cutting tickets, using the format "team-name-ticket-number." A description for the ticket and acceptance criteria are also provided to ensure we have a roadmap for development throughout the sprint.

TEAM COMMUNICATION

COMMUNICATION PROCESS

Effective communication is critical to our team's success, which is why we hold regular meetings to discuss progress and address any issues that may arise. Our main meeting is held on **`Discord`**, where we estimate and assign tasks, and cut tickets. To keep things flexible during the sprint, we use a discord text channel to update one another, following the **`Scrum`** framework. By utilizing these tools and frameworks, we can ensure our team stays in sync with one another's progress, and we can quickly address any issues that arise during the development process.

GITHUB REPO USAGE

BRANCHES AND CONVENTIONS

Our team uses **`GitHub`** to manage our codebase. To keep our code organized and modular, we utilize feature branches. We name each branch after the ticket it corresponds to, such as "team-name-ticket-number," to ensure everyone is aware of what feature branch relates to what ticket.

PULL REQUESTS

To ensure a consistent and error-free process, we have established several rules within GitHub. For instance, you cannot merge without making a pull request, and each pull request requires at least one approval before merging.

COMMITS

We also follow the "**`atomic commit message rule`**," using concise and specific commit messages that are on point. If a commit message contains "AND," it indicates the commit is not atomic and should be divided into two separate commits.

ISSUE DOCUMENTATION

ISSUE #5663: Clone Should Not Deepcopy Constructor Parameters

PULL REQUEST: #1
COLLABORATORS: Bassel Ashi & Roozbeh Yadollahi
FILES MODIFIED

- **scikit-learn/sklearn/base.py**
- **scikit-learn/sklearn/tests/test_base.py**

IMPLEMENTATION

The `clone` function being used has been refactored to include a new parameter named `deepcopy` with default value `False`. This allows our update to be backward compatible and prevent performing deep-copy across all estimators that make use of this function. As for the logic of performing shallow vs. deep copy, it has been updated directly in the `_clone_parametrized` through the same concept of adding a default parameter. Within this function, there is a block of code that performs the deep copying by looping through the attributes and making a copy of them, so we added an if statement based on the `deepcopy` parameter.

TESTING

The **`test_clone_shallow_copy`** and **`test_clone_deepcopy`** function aims to test the functionality of the **`clone`** function with the objective of ensuring that the function is returning a shallow copy or deep copy of the input parameters of an estimator object when the deep copy parameter is set to False and True respectively.
The acceptance tests aim to verify that the **`get_params()`** API of the estimator object returns a dictionary with the same key-value pairs and number of parameters. Additionally, the tests ensure that the value of each parameter has the same reference when the deepcopy parameter is set to False, and the opposite if it is set to True.

To achieve this, we implemented two tests that call the clone function on an instance of an estimator object with the deepcopy parameter set to True or False, respectively. In each test, we obtain the parameters of the original estimator object using the **`get_params()`** API, and compare them with the corresponding parameters of the cloned estimator object.

We verify that the number of key-value pairs in both dictionaries is the same. Additionally, for each key-value pair, we check that the value has the same reference as the original object when deepcopy is set to False, and when deepcopy is set to True, we expect that each value has a different reference.

To facilitate this comparison, we have implemented a helper private function named **`_is_immutable()`**.This function determines whether an object is immutable or not, which is critical for distinguishing between shallow and deep copies.

This is because for immutable objects in Python, the behavior of shallow and deep copies is the same, and two identical values will have the same reference. However, determining the immutability of an object is not foolproof in python and there is no certain way,, and the implementation of this function is subject to change based on the codebase and input parameters of the estimator object. However, **`_is_immutable()`** is sufficient for now based on the input parameters of the estimator object.

In conclusion, these acceptance tests ensure and validate the correctness of the implementation of the **`clone`** function.

Our acceptance tests batch are implemented in the related tests file for **`clone`** at `scikit-learn/sklearn/tests/test_base.py` to ensure the correctness of our implementation. The entire test suite was run to see if our change caused more tests to fail than before modification as our acceptance test, i.e.:

1. `pytest sklearn/tests`
2. check the number of passes and fails
3. make modification to `scikit-sklearn/sklearn/utils/_random.pyx`

4. `pip install --no-build-isolation -e .` to recompile the cython modules
5. `pytest sklearn/tests`
6. check the number of passes and fails, and see if there are less passes than from step 1.
7. To run the tests that we implemented individually write the following command:
   ```
   pytest sklearn/tests/test_base.py::test_clone_deepcopy
   pytest sklearn/tests/test_base.py::test_clone_shallow_copy
   ```

ISSUE #20435: Incorrect documentation for `warm_start` behavior on BaseForest-derived classes

PULL REQUEST: #2
COLLABORATORS: Raymond Kiguru & Klein Harrigan
FILES MODIFIED

- **scikit-learn/sklearn/ensemble/_forest.py**
- **scikit-learn/sklearn/ensemble/tests/test_forest.py**

IMPLEMENTATION

The `fit` function has been updated to no longer perform an `oob_score` update granted the number of added estimators since the last call to `fit` is zero. These updates have been applied directly to the `BaseForest` implementation in `sklearn/ensemble/_forest.py`. The reason for this is that the `fit` function is implemented in the base class and used in all derived classes. Since we don't wish to re-run the `oob_score_` calculation if there are no more estimators in a successive call to `fit`, we check that `n_more_estimators` is greater than zero. However, in the event that `oob_score` is set to `False` when `fit` is first run, and then set to `True` for successive calls, we also check if the `oob_score_` attribute has been set. In the event that either `oob_score_` is not set or `n_more_estimators` is greater than zero, we will calculate the `oob_score_`.

TESTING

There exist tests to ensure `oob_score_` is not calculated when `oob_score` is set to `False`. Further tests include checks that later setting `oob_score` to `True` will yield a correct `oob_score_`. These can be found in `sklearn/ensemble/tests/test_forest.py:check_warm_start_oob`. A further test case has been added to ensure that if `oob_score_` is set and `n_more_estimators` is zero, that `oob_score_` is not updated: `sklearn/ensemble/tests/test_forest.py:check_no_recompute_oob`.

ISSUE #25484: MAINT Improve scikit-learn reliability by replacing cnp.ndarrays with typed memoryviews

PULL REQUEST: #3 #4
COLLABORATORS: Siavash Yassemi & Vinesh Benny
FILES MODIFIED

- **scikit-learn/sklearn/utils/_random.pyx**
- **scikit-learn/sklearn/_isotonic.pyx**

IMPLEMENTATION

The methods `_sample_without_replacement_with_tracking_selection`, `_sample_without_replacement_with_pool`, `_sample_without_replacement_with_reservoir_sampling` in `scikit-sklearn/sklearn/utils/_random.pyx` have been updated to use typed memoryviews instead of `cnp.ndarray`. This allows for efficient access to memory buffers, such as those underlying NumPy arrays, without incurring any Python overhead. Each function uses `np.asarray()` to coerce the memoryviews into a NumPy `ndarray`, while not having to copy the data. This allows our update to be backwards compatible as the functions still return a `ndarray` as they did before our modification. As for the logic of replacing the cnp.ndarrays to typed memoryviews, every instance of `cnp.ndarray` has been replaced with its equivalent typed memoryview.
Similarly, `_isotonic.pyx` has had its `_make_unique` method replaced so it is using memoryviews instead of `cnp.ndarray`.

TESTING

The unit tests provided at `scikit-sklearn/sklearn/utils/tests/test_random.py` were used to test our modifications. The entire test suite was run to see if our change caused more tests to fail than before modification as our acceptance test, i.e.:

1. `pytest sklearn/tests`
2. check the number of passes and fails
3. make modification to `scikit-sklearn/sklearn/utils/_random.pyx`
4. `pip install --no-build-isolation -e .` to recompile the cython modules
5. `pytest sklearn/tests`
6. check the number of passes and fails, and see if there are less passes than from step 1.