CSCD01: Deliverable 4

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 #20804: Revisiting the tags interface

PULL REQUEST: #7
COLLABORATORS: Siavash Yassemi & Bassel Ashi & Roozbeh Yadollahi
DESIGN AND ISSUE

The task we chose involves restructuring how tags work in Scikit-learn. Tags are used to determine the checks that should be applied to estimators and input data. For instance, setting the tag `requires_fit=True` indicates that the estimator will require that the fit method is called beforehand. Currently, tags are handled using the `__get_tags()` and `_more_tags` methods. However, we want to restructure this implementation by creating `__sklearn_tags__` method that uses python inheritance instead of MRO. The current implementation requires developers to import BaseEstimator, as well as any other classes and mixins, to use their tags and add new ones. By creating this new method,any class or mixin will just call the parent's class's `__sklearn_tags__` method. Moreover, since any other tags included in Scikit-learn can be used via inheritance, developers will not need to import their classes. We decided to take on this task as it has a larger scope with a huge impact on the entire codebase.

IMPLEMENTATION

The `_more_tags` method for each estimator mixin was replaced with `__sklearn_tags__` method such that for any existing method, they would need to implement this new method. When they implement the new method, they inherit all the tags from their super class, and then add their own tags before passing it along. Besides `_more_tags` method, `_get_tags` method will also be replaced by our new method in all instances. This is important since most classes inheriting from BaseEstimator will have to change their behavior to account for this change.
For instance consider the case for `KernelRidge`

```
def _more_tags(self):
    return {"pairwise": self.kernel == "precomputed"}
```

To:

```
def __sklearn_tags__(self):
    tags =  super().__sklearn_tags__()
    tags["pairwise"] = self.kernel == "precomputed"
    return tags
```

The parent class for `KernelRidge` had set the tag for `tags["multioutput"] = True`

The perpetual add to the tags can be tracked all the way back up to the base class where the tags are just a deep_copy of some default tags. Because of the use of inheritance, any new class in most cases can just inherit its parents tags and does not need to go all the way up to the base estimator and then modify and add its own tags which is the whole point of this change.

For any developer to use tags, the classes they create can either extend BaseEstimator or any child of BaseEstimator. **`__sklearn_tags__`** will then have to be implemented with a call to the parent, while making sure that the new tags dictionary will have every single tag available still (the key:value pair should exist for any instance).
The changes made to a specific tag are similar to the example above.
BaseEstimator is using `_DEFAULT_TAGS` which can be found at `_tags.py` .

TESTING

Unit tests:

Given that the issue we are working on at its core is a refactor, the unit tests we designed aimed to account for proper implementation of the **`__sklearn_tags__`** for the base class, as well as the instances where the exceptions could occur namely the `Pipeline` and `_GeneralizedLinearRegressor` class. The BaseEstimator unit test was designed to cross check that the default we are providing are consistent with the default tags we would like to provide. Additionally, the unit test cases were mocked to account for Type error, Value error, and Attribute errors that the mentioned classes could encounter. We used these unit test cases to guide our refactor to make sure we are approaching the issue correctly.

Acceptance tests:

Due to the refactoring nature of the issue, the existing acceptance tests were also updated to account for the change so we made sure they are using **`__sklearn_tags__`** in appropriate places in test cases like the following:

```
    # did the pipeline set the pairwise attribute?
-   assert pipeline._get_tags()["pairwise"]
+   assert pipeline.__sklearn_tags__()["pairwise"]
```

These tests account for tags with different types, and were applied to all the test cases using tags for their unit tests.
The new test cases could all be found at:
`\scikit-learn\acceptanceTests\issue20804.py`
`\scikit-learn\unitTests\issue20804.py`

To run all the tests do the following:

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_tags.py`

FILES MODIFIED

- **scikit-learn/acceptanceTests/issue20804.py**
- **scikit-learn/doc/developers/develop.rst**
- **scikit-learn/doc/sphinxext/allow_nan_estimators.py**
- **scikit-learn/sklearn/_isotonic.pyx**
- **scikit-learn/sklearn/base.py**
- **scikit-learn/sklearn/calibration.py**
- **scikit-learn/sklearn/cluster/_affinity_propagation.py**
- **scikit-learn/sklearn/cluster/_bicluster.py**
- **scikit-learn/sklearn/cluster/_birch.py**
- **scikit-learn/sklearn/cluster/_bisect_k_means.py**
- **scikit-learn/sklearn/cluster/_dbscan.py**
- **scikit-learn/sklearn/cluster/_kmeans.py**
- **scikit-learn/sklearn/cluster/_spectral.py**
- **scikit-learn/sklearn/compose/_target.py**
- **scikit-learn/sklearn/cross_decomposition/_pls.py**
- **scikit-learn/sklearn/decomposition/_dict_learning.py**
- **scikit-learn/sklearn/decomposition/_fastica.py**
- **scikit-learn/sklearn/decomposition/_kernel_pca.py**
- **scikit-learn/sklearn/decomposition/_lda.py**
- **scikit-learn/sklearn/decomposition/_nmf.py**
- **scikit-learn/sklearn/decomposition/_pca.py**
- **scikit-learn/sklearn/decomposition/_sparse_pca.py**
- **scikit-learn/sklearn/decomposition/_truncated_svd.py**
- **scikit-learn/sklearn/dummy.py**
- **scikit-learn/sklearn/ensemble/_bagging.py**
- **scikit-learn/sklearn/ensemble/_forest.py**
- **scikit-learn/sklearn/ensemble/_hist_gradient_boosting/gradient_boosting.py**

- scikit-learn/sklearn/ensemble/_iforest.py
- scikit-learn/sklearn/ensemble/_voting.py
- scikit-learn/sklearn/ensemble/tests/test_bagging.py
- scikit-learn/sklearn/feature_extraction/_dict_vectorizer.py
- scikit-learn/sklearn/feature_extraction/_hash.py
- scikit-learn/sklearn/feature_extraction/image.py
- scikit-learn/sklearn/feature_extraction/text.py
- scikit-learn/sklearn/feature_selection/_base.py
- scikit-learn/sklearn/feature_selection/_from_model.py
- scikit-learn/sklearn/feature_selection/_rfe.py
- scikit-learn/sklearn/feature_selection/_sequential.py
- scikit-learn/sklearn/feature_selection/_univariate_selection.py
- scikit-learn/sklearn/feature_selection/_variance_threshold.py
- scikit-learn/sklearn/feature_selection/tests/test_from_model.py
- scikit-learn/sklearn/feature_selection/tests/test_rfe.py
- scikit-learn/sklearn/gaussian_process/_gpr.py
- scikit-learn/sklearn/impute/_base.py
- scikit-learn/sklearn/impute/tests/test_knn.py
- scikit-learn/sklearn/isotonic.py
- scikit-learn/sklearn/kernel_approximation.py
- scikit-learn/sklearn/kernel_ridge.py
- scikit-learn/sklearn/linear_model/_base.py
- scikit-learn/sklearn/linear_model/_coordinate_descent.py
- scikit-learn/sklearn/linear_model/_glm/glm.py
- scikit-learn/sklearn/linear_model/_glm/tests/test_glm.py
- scikit-learn/sklearn/linear_model/_least_angle.py
- scikit-learn/sklearn/linear_model/_logistic.py
- scikit-learn/sklearn/linear_model/_ransac.py
- scikit-learn/sklearn/linear_model/_ridge.py
- scikit-learn/sklearn/linear_model/_stochastic_gradient.py
- scikit-learn/sklearn/linear_model/tests/test_coordinate_descent.py
- scikit-learn/sklearn/manifold/_isomap.py
- scikit-learn/sklearn/manifold/_mds.py
- scikit-learn/sklearn/manifold/_spectral_embedding.py
- scikit-learn/sklearn/manifold/_t_sne.py
- scikit-learn/sklearn/model_selection/_search.py
- scikit-learn/sklearn/model_selection/_search_successive_halving.py
- scikit-learn/sklearn/model_selection/tests/test_search.py
- scikit-learn/sklearn/multiclass.py

- scikit-learn/sklearn/multioutput.py
- scikit-learn/sklearn/naive_bayes.py
- scikit-learn/sklearn/neighbors/_base.py
- scikit-learn/sklearn/neighbors/_classification.py
- scikit-learn/sklearn/neighbors/_graph.py
- scikit-learn/sklearn/neighbors/_kde.py
- scikit-learn/sklearn/neighbors/_lof.py
- scikit-learn/sklearn/neighbors/_nca.py
- scikit-learn/sklearn/neighbors/_regression.py
- scikit-learn/sklearn/neural_network/_multilayer_perceptron.py
- scikit-learn/sklearn/neural_network/_rbm.py
- scikit-learn/sklearn/pipeline.py
- scikit-learn/sklearn/preprocessing/_data.py
- scikit-learn/sklearn/preprocessing/_encoders.py
- scikit-learn/sklearn/preprocessing/_function_transformer.py
- scikit-learn/sklearn/preprocessing/_label.py
- scikit-learn/sklearn/preprocessing/_polynomial.py
- scikit-learn/sklearn/preprocessing/tests/test_data.py
- scikit-learn/sklearn/preprocessing/tests/test_encoders.py
- scikit-learn/sklearn/random_projection.py
- scikit-learn/sklearn/svm/_base.py
- scikit-learn/sklearn/svm/_classes.py
- scikit-learn/sklearn/tests/test_base.py
- scikit-learn/sklearn/tests/test_docstring_parameters.py
- scikit-learn/sklearn/tests/test_multiclass.py
- scikit-learn/sklearn/tests/test_pipeline.py
- scikit-learn/sklearn/tests/test_tags.py
- scikit-learn/sklearn/tree/_classes.py
- scikit-learn/sklearn/utils/_mocking.py
- scikit-learn/sklearn/utils/_tags.py
- scikit-learn/sklearn/utils/estimator_checks.py
- scikit-learn/sklearn/utils/tests/test_estimator_checks.py
- scikit-learn/sklearn/utils/tests/test_tags.py
- scikit-learn/unitTests/issue20804.py

ISSUE #26015: GaussianMixture is runnning too many computation when parameters are alredy provided

PULL REQUEST: #6

COLLABORATORS: Raymond Kiguru

FILES MODIFIED

- **scikit-learn/sklearn/mixture/_gaussian_mixture.py**
- **scikit-learn/sklearn/mixture/tests/test_gaussian_mixture.py**

DESIGN and IMPLEMENTATION

As things stand, the current implementation of `GaussianMixture` will always compute weights, means, and covariances for the `GaussianMixture` regardless of whether these values have been provided manually. This leads to a great degree of computational overhead that could be otherwise prevented by changing how `GaussianMixtures` are initialized in Scikit-Learn.

The design of this feature revolves around the different parameters that are needed to instantiate a `GaussianMixture`. Among the various data needed by a GaussianMixture are mixture weights (the probability of each Gaussian distribution in the mixture), means (the center points of each Gaussian distribution in the mixture), and precisions (the inverses of the covariance matrices of the Gaussian distributions in the mixture). These values can be manually dictated by the user, or can be calculated through an estimation process that makes use of some `X` (an input data matrix) and `resp` (a reference of responsibilities for each sample to each Gaussian). As things stand, each of these is being approximated through the aforementioned estimation process and then being overwritten later in execution with user-defined values `weights_init`, `means_init`, and `precisions_init`.

With this being a Performance-tagged issue, the focus was on refactoring for greater efficiency while avoiding changing the code's functionality. It can be noted that if any of the three user-defined values is missing, the estimation process need be completed anyway to estimate the missing value. The initial approach was to consolidate conditional expressions throughout the `GaussianMixture`'s initialization such that if the user defined all three values, that each value would be instantiated with the user-provided initial values and that execution of the approximation would take place otherwise with each value assigned the corresponding estimated value. This approach, however, fails given the fact that the existing implementation sets the values of any of the three which has a defined initial user-provided value to said value.

The primary technique applied was instead to consolidate duplicate conditional fragments and replace nested conditional with guard clauses. We recognized that while each value would be set to the corresponding estimate only when no initial value was provided, the estimation process would be done regardless if any one initial value was missing. As such, we perform the estimation process only once if any of the initial values is missing (as we'd need perform it anyway). From here, we can perform the piecemeal checks for definition on each of the initial values and set the corresponding value (the given initial value if defined or the estimate if not) in successive checks without having to rerun the estimate. This approach minimizes the repetition and interdependency of all conditional checks during initialization with only one duplicated conditional statement evaluation.

TESTING

Unit Tests:

This issue was primarily a refactor to increase performance of the `GaussianMixture` class by preventing superfluous computation. Our unit tests aim to ensure that `GaussianMixture` remains initialized correctly. That is, we make sure that all values (`weights_`, `means_`, and `precisions_cholesky_`) remain correctly defined across classes and workloads that use them.

Acceptance Tests:

There exist tests to ensure that the Gaussian estimator `_estimate_gaussian_parameters` is only called when any of `weights_init`, `means_init`, and `precisions_init` are missing. This is done by checking for values initialized only in the case that any of the three initial values is missing (`covariances_`), initialized in the event that `precisions_init` is not provided.

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 more failures than step 1.