HEXAD01

SOFTWARE DEVELOPMENT TEAM

Aidan Zorbas Dawson Brown Jenny Yu

Jingrun Long Kara Autumn Jiang Vanessa Pierre

Scikit-learn Moderate Issues Report

Assignment 3 Deliverable

Table of Contents

Table of Contents 3

Moderate Level Scikit-learn Issues 4

Issue #18057: RandomizedSearchCV 4

Issue Description 4

Related Components 4

Issue #12505: MLPClassifier 5

Related Components 5

RandomizedSearchCV Bugfix 6

Issue Selection Decision 6

Bug Analysis 6

Our Bugfix Solution 7

Organization and UML 9

RandomizedSearchCV Unit Test Suite 11

Summary of Unit Test Coverage 11

Unit Test Acceptance Conditions 11

Running and Reading Unit Tests 11

RandomizedSearchCV Acceptance Testing Suite 12

Summary of Acceptance Test Coverage 12

Acceptance Test Coverage and Acceptance Conditions 12

Running and Reading Acceptance Tests 13

Moderate Level Scikit-learn Issues

During this developmental phase, our team identified two Moderate-Level issues within the scikit-learn source code repository. These issues are as described below:

Issue #18057: RandomizedSearchCV

Issue Description

The RandomizedSearchCV component allows users to perform randomized searches on hyperparameters through its implementation of the fit() and score() methods. Subclasses of this component may also implement additional estimator methods such as score_samples(), “predict(), predict_proba(), transform(), etc. if these additional methods are implemented in the selected estimator.

Depending on the estimator being used, various parameters used to apply these methods are optimized via cross-validation searching over provided parameter settings. Although not all parameters will be used, a fixed number of parameter settings will be sampled from the specified distribution according to the constant provided by the user via the n_iter argument.

To specify the actual data to sample from, the param_distributions argument can be used to input the names and values of the hyperparameters to sample from as a single Python dictionary with the name string as the key and either a Python list of possible hyperparameter values or a distribution with a rvs() method as the value. If all the values are lists, then they are sampled without replacement, while if there are any distributions present, then values are sampled with replacement. Each sample is a random selection of values for each hyperparameter. Multiple sets of parameters can also be provided with a Python list of the previously mentioned dictionaries, and a set is chosen at random uniformly for each iteration.

Given this specified functionality of the RandomizedSearchCV component, it would be expected that if multiple parameter distributions were provided as a list containing multiple dictionaries, then on each iteration within the specified range, first one of the provided dictionaries would be selected at random, then a random set of parameters within that dictionary would be chosen. Thus, as the number of iterations provided through n_iter increases, we would expect to see an even selection of parameters from each provided dictionary. For example, with four parameter distribution dictionaries over 100 iterations, we would expect to see each dictionary selected roughly 25 times.

This is not the case however, as with the current implementation of this component, on every iteration, dictionaries with more possible parameter combinations are favoured over those with fewer combinations, resulting in a sampling imbalance.

Related Components

sklearn/model_selection/_search.py

Issue #12505: MLPClassifier

Scikit-learn’s MLPClassifier is a Multilayer Perceptron Classifier model which optimizes the log-loss function using either a Low-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm (LBFGS), or stochastic gradient descent. The second option supports fitting on multi-label output but does not work correctly when partial_fit() is invoked, specifically as the partial_fit methods breaks after a single iteration rather than performing up to max_iter iterations as specified by the user, and yielding a much lower than expected precision score compared to the fit() method.

Although additional research may be needed to determine the root cause of this bug, it is suspected that it may in part be caused by the fact that the n_iter_no_change variable is disregarded in the partial_fit() method, despite the _no_improvement_count parameter not being reset between partial_fit() calls since the update on the partial_fit batch is performed before this criterion is checked.

Despite not being able to definitively identify a singular point of failure responsible for causing this bug, we can however verify that the partial_fit() method’s low precision score is indeed caused by the method crashing after just one iteration, as by setting the max_iter argument to 1, it behaves identically to the fit() method, and similarly results in a near-identical precision score.

Related Components

sklearn/neural_network/_multilayer_perceptron.py > MLPClassifier

RandomizedSearchCV Bugfix

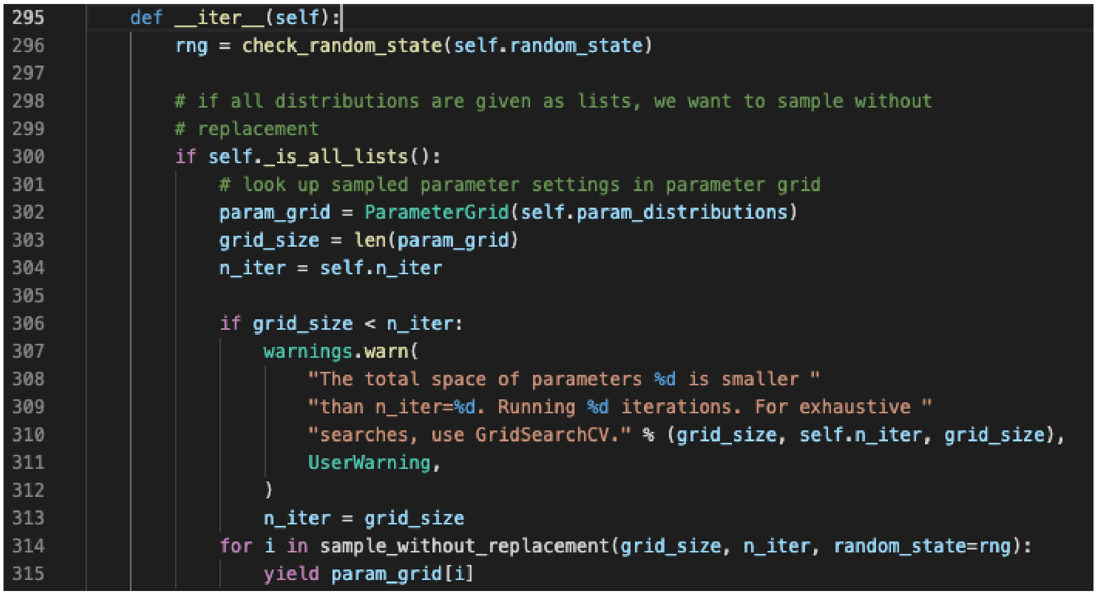
Issue Selection Decision

Having examined a number of moderate-level issues within the scikit-learn repository, we found that many other bugs or proposed features were vaguely defined, or required an extensive machine-learning background to understand. In order to both circumvent these ambiguities and ensure that our development aligned with the project's goals, our issue selection process focused around identifying and working on an issue that was both clearly defined, and whose fix could be verified against a well-defined set of acceptance conditions.

As such, our team ultimately selected Issue #18057 as it was clearly documented, with steps to both reproduce and understand the bug within the RandomizedSearchCV component behaviour, as well as a series of expectations for how the component should behave under normal circumstances. These clearly defined parameters helped us to better understand what our objective would be in designing a bugfix, as well as defining a clear set of test cases and acceptance conditions to verify that our bugfix was appropriate and in-line with both our own objectives and those as defined by the scikit-learn documentation.

Bug Analysis

This bug originates between lines 300-315 in scikit-learn's model_selection/_search.py as (ParameterSampler -> __iter__()), the list of dictionaries of parameters (self.param_distributions) were being flattened into a single grid of parameters. Thus, during sampling, parameters belonging to dictionaries with a larger number of parameter combinations were statistically more likely to be selected.



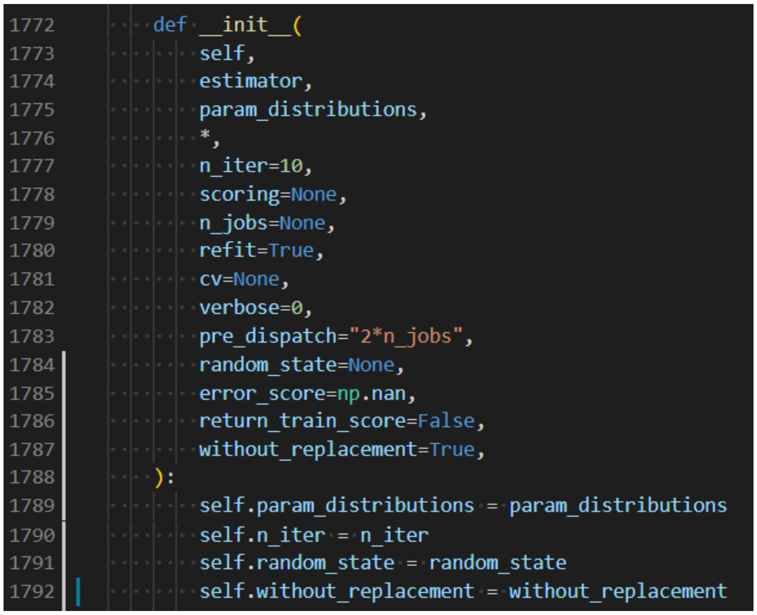
```
295     def __iter__(self):
296         rng = check_random_state(self.random_state)
297
298         # if all distributions are given as lists, we want to sample without
299         # replacement
300         if self._is_all_lists():
301             # look up sampled parameter settings in parameter grid
302             param_grid = ParameterGrid(self.param_distributions)
303             grid_size = len(param_grid)
304             n_iter = self.n_iter
305
306             if grid_size < n_iter:
307                 warnings.warn(
308                     "The total space of parameters %d is smaller "
309                     "than n_iter=%d. Running %d iterations. For exhaustive "
310                     "searches, use GridSearchCV." % (grid_size, self.n_iter, grid_size),
311                     UserWarning,
312                 )
313                 n_iter = grid_size
314             for i in sample_without_replacement(grid_size, n_iter, random_state=rng):
315                 yield param_grid[i]
```

Above: As seen at line 302, the entire list of parameter dictionaries is flattened into a single parameter grid which is then used during the selection process.

Next between lines 314-315, the original source code was randomly selecting n_iter combinations from the provided parameter grid. In the case where one of the parameter dictionaries contained more combinations, their parameter values were far more likely to be selected over those containing fewer combinations instead of yielding even selection behaviour across all provided dictionaries.

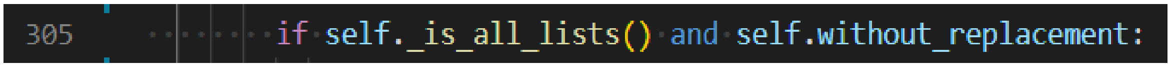
Our Bugfix Solution

To allow for even sampling, we added a new parameter to RandomizedSearchCV and ParameterSampler called `without_replacement`, which defaults to True;



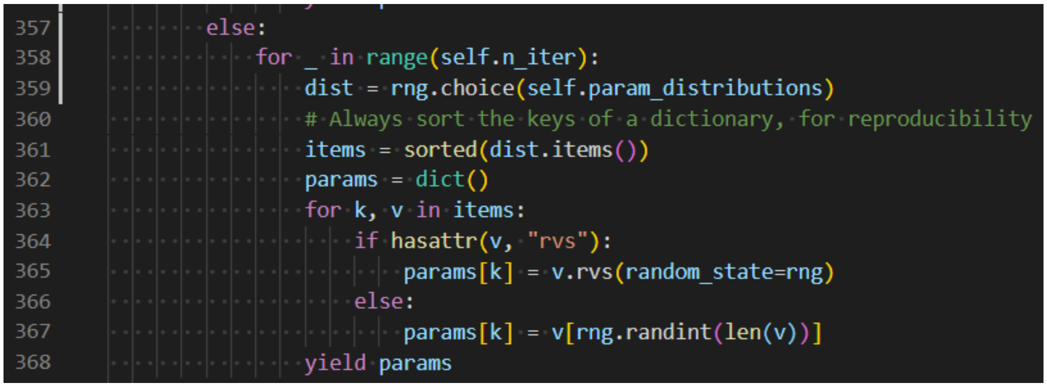
```
    def __init__(
        self,
        estimator,
        param_distributions,
        *,
        n_iter=10,
        scoring=None,
        n_jobs=None,
        refit=True,
        cv=None,
        verbose=0,
        pre_dispatch="2*n_jobs",
        random_state=None,
        error_score=np.nan,
        return_train_score=False,
        without_replacement=True,
    ):
        self.param_distributions = param_distributions
        self.n_iter = n_iter
        self.random_state = random_state
        self.without_replacement = without_replacement
```

Along with the following check on line 305 of _search.py:



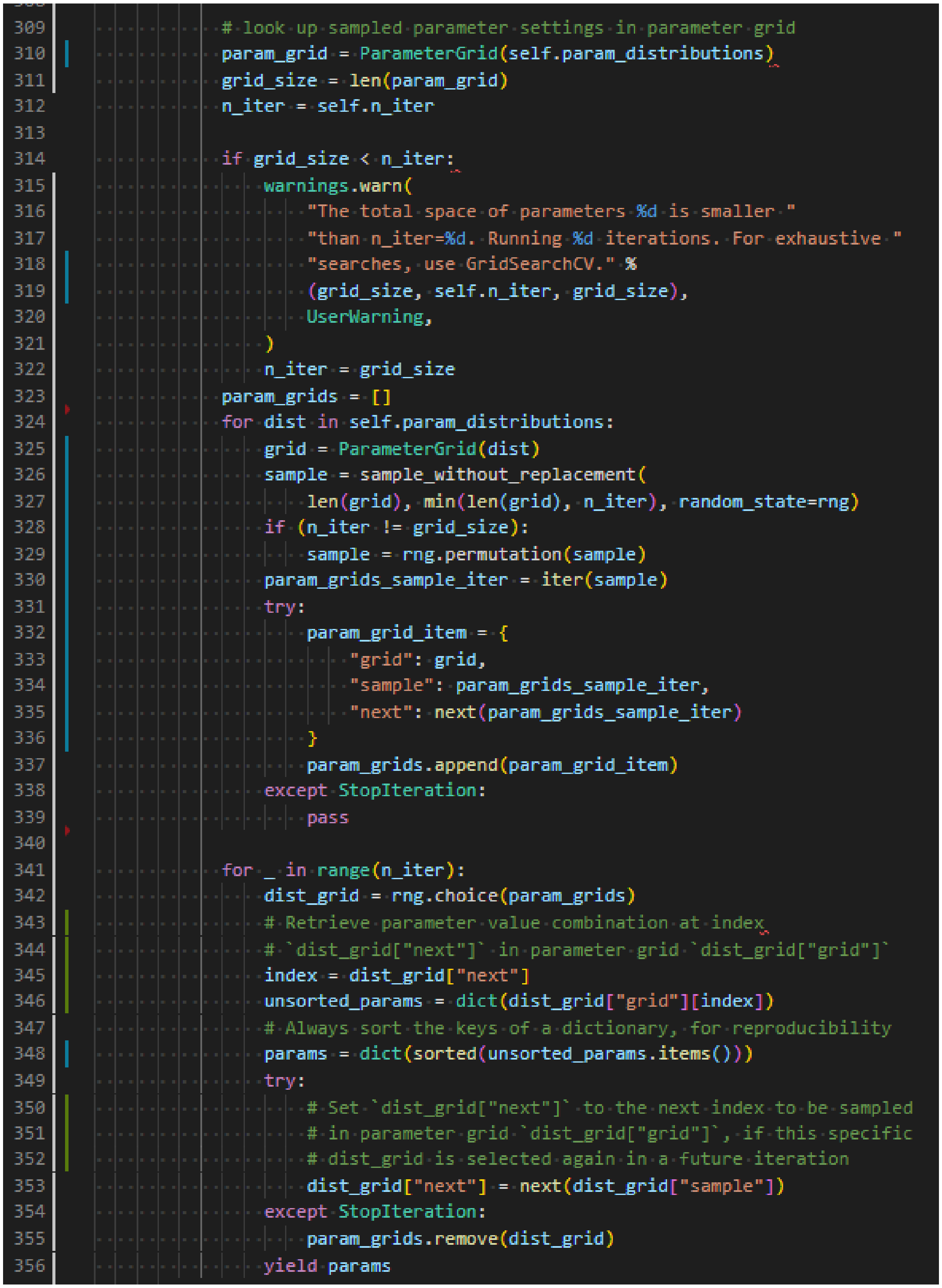
```
305         if self._is_all_lists() and self.without_replacement:
```

Setting `without_replacement=False` causes sampling with replacement to be used, thus allowing a dictionary to be sampled uniformly first with choice() at line 359, and then a parameter being sampled from that selected dictionary at either line 365 or 367.



```
357         else:
358             for _ in range(self.n_iter):
359                 dist = rng.choice(self.param_distributions)
360                 # Always sort the keys of a dictionary, for reproducibility
361                 items = sorted(dist.items())
362                 params = dict()
363                 for k, v in items:
364                     if hasattr(v, "rvs"):
365                         params[k] = v.rvs(random_state=rng)
366                     else:
367                         params[k] = v[rng.randint(len(v))]
368                 yield params
```

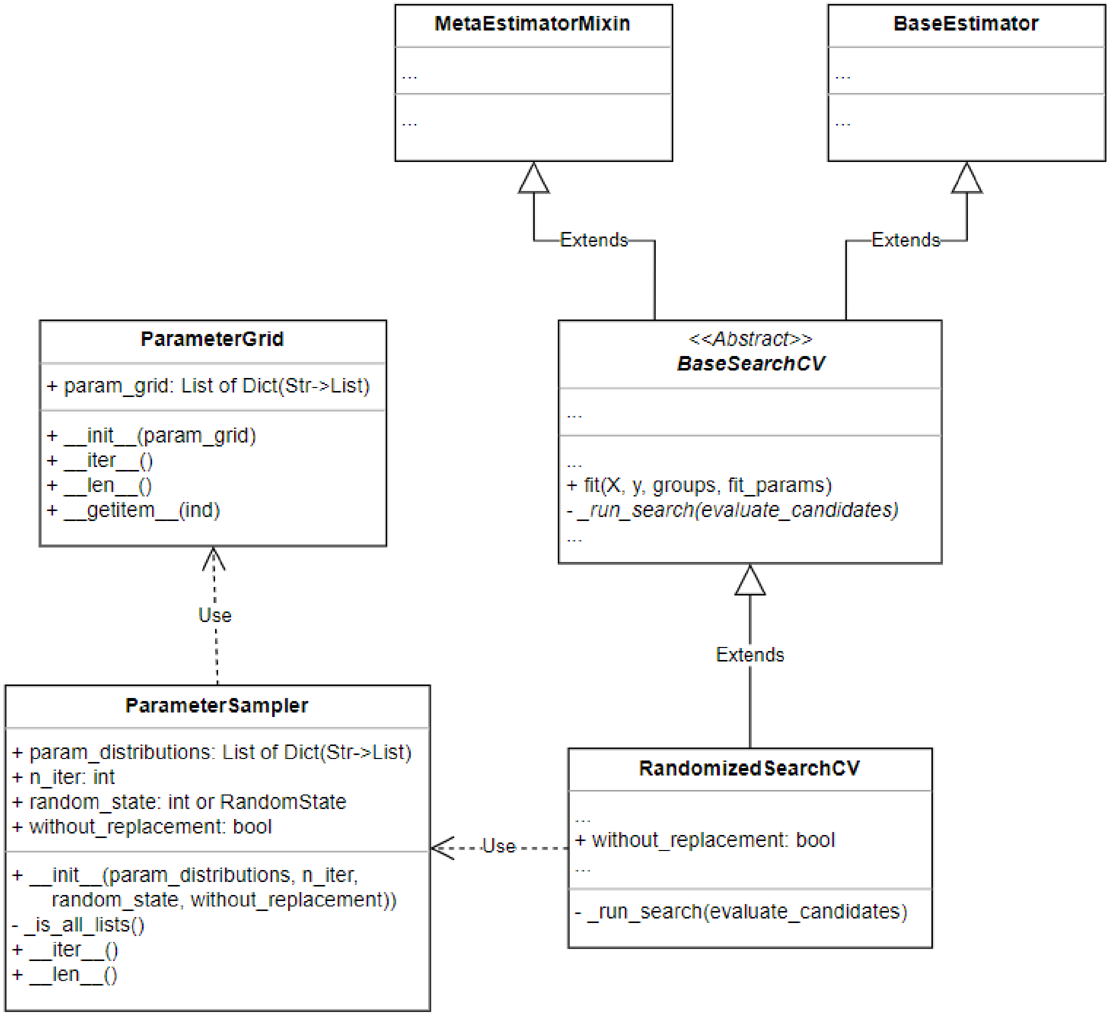
Sampling without replacement is only plausible if all parameters in the provided dictionaries are discrete (ie. presented a list). However, the previous method of sampling without replacement combined all dictionary parameters into a single grid, and selected randomly from that grid. This meant that parameter dictionaries with more possible element combinations were more likely to be sampled from, which did not match the behaviour reflected in the sampling with replacement code.



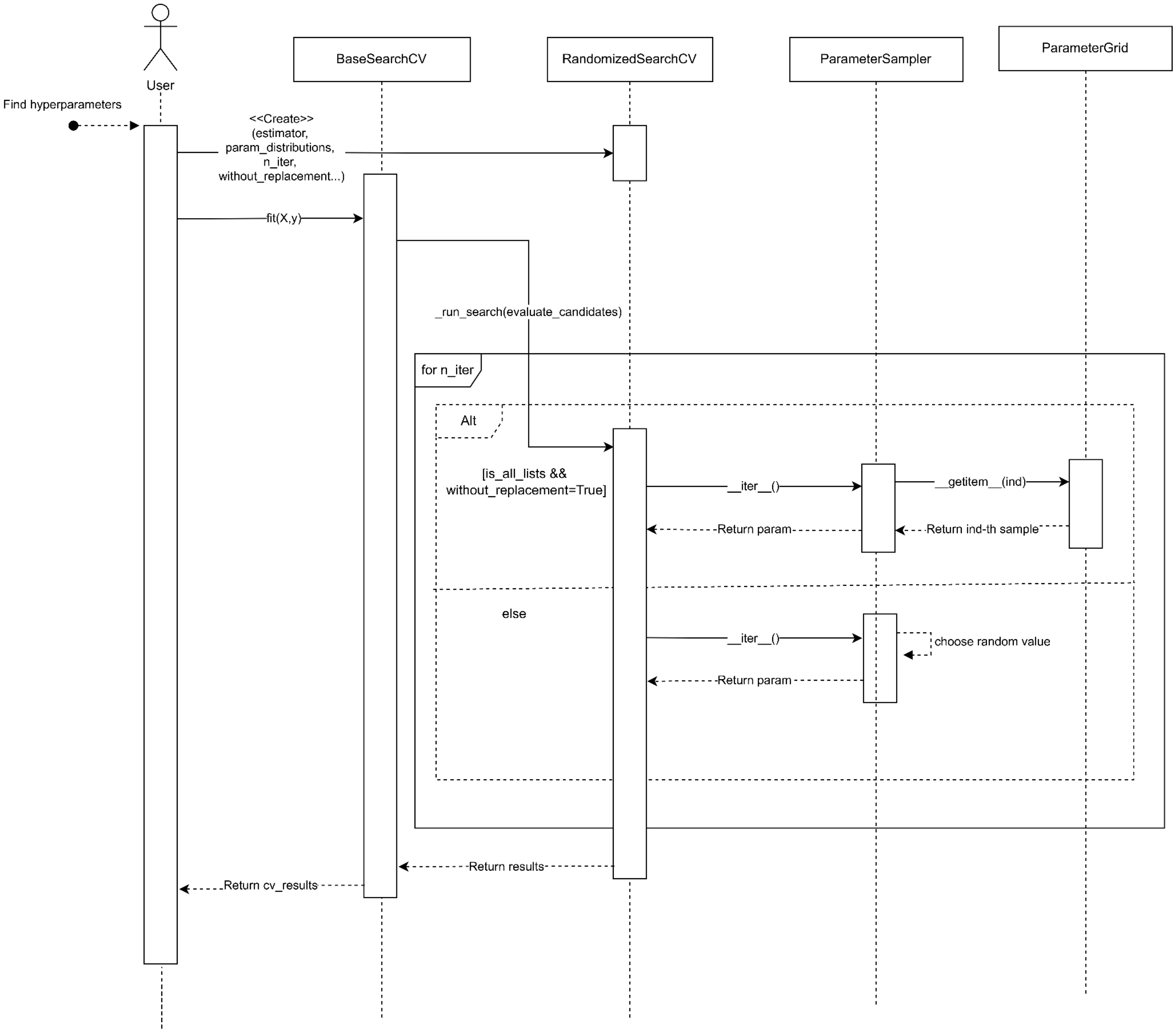
```
            # look up sampled parameter settings in parameter grid
            param_grid = ParameterGrid(self.param_distributions)
            grid_size = len(param_grid)
            n_iter = self.n_iter

            if grid_size < n_iter:
                warnings.warn(
                    "The total space of parameters %d is smaller "
                    "than n_iter=%d. Running %d iterations. For exhaustive "
                    "searches, use GridSearchCV." %
                    (grid_size, self.n_iter, grid_size),
                    UserWarning,
                )
                n_iter = grid_size
            param_grids = []
            for dist in self.param_distributions:
                grid = ParameterGrid(dist)
                sample = sample_without_replacement(
                    len(grid), min(len(grid), n_iter), random_state=rng)
                if (n_iter != grid_size):
                    sample = rng.permutation(sample)
                param_grids_sample_iter = iter(sample)
                try:
                    param_grid_item = {
                        "grid": grid,
                        "sample": param_grids_sample_iter,
                        "next": next(param_grids_sample_iter)
                    }
                    param_grids.append(param_grid_item)
                except StopIteration:
                    pass

            for _ in range(n_iter):
                dist_grid = rng.choice(param_grids)
                # Retrieve parameter value combination at index
                # `dist_grid["next"]` in parameter grid `dist_grid["grid"]`
                index = dist_grid["next"]
                unsorted_params = dict(dist_grid["grid"][index])
                # Always sort the keys of a dictionary, for reproducibility
                params = dict(sorted(unsorted_params.items()))
                try:
                    # Set `dist_grid["next"]` to the next index to be sampled
                    # in parameter grid `dist_grid["grid"]`, if this specific
                    # dist_grid is selected again in a future iteration
                    dist_grid["next"] = next(dist_grid["sample"])
                except StopIteration:
                    param_grids.remove(dist_grid)
                yield params
```

In the screenshot above, a new method of sampling without replacement is implemented. This time, by creating as many parameter grids as there are dictionaries in `self.param_distributions`. This allows us to uniformly sample a dictionary first, then sample a parameter combination from the parameter grid related to the selected dictionary. This means all parameter dictionaries have an equal probability of being selected, regardless of how many elements they contain. The exception is if a dictionary runs out of elements to sample from, it is removed from the list of options on line 355, and the code now uniformly samples from the remaining dictionaries on line 342.

Organization and UML


MetaEstimatorMixin
...
...
BaseEstimator
...
...
Extends
Extends
ParameterGrid
+ param_grid: List of Dict(Str->List)
+ __init__(param_grid)
+ __iter__()
+ __len__()
+ __getitem__(ind)
<<Abstract>>
BaseSearchCV
...
...
+ fit(X, y, groups, fit_params)
- _run_search(evaluate_candidates)
...
Use
Extends
ParameterSampler
+ param_distributions: List of Dict(Str->List)
+ n_iter: int
+ random_state: int or RandomState
+ without_replacement: bool
+ __init__(param_distributions, n_iter, random_state, without_replacement))
- _is_all_lists()
+ __iter__()
+ __len__()
Use
RandomizedSearchCV
...
+ without_replacement: bool
...
- _run_search(evaluate_candidates)

Our fix adds a without_replacement parameter to RandomizedSearchCV and ParameterSampler, the class used by RandomizedSearchCV to randomly sample hyperparameters. This option was chosen to allow previous behaviour to still be used, and the parameter defaults to without_replacement=True to maintain old behaviour, but the updated functionality can be used by setting without_replacement=False so that sampling will be done with replacement even for input dictionaries that are all lists.



Shown above is a sequence diagram for a user running fit() with a RandomizedSearchCV to find hyperparameters. The main method involved in this bug is __iter__() in ParameterSampler which determines the sampled parameters to return. If without_replacement is True and all inputs are lists, then ParameterSampler's __iter__() makes use of ParameterGrid to build the set of values to sample from. Otherwise, the values are randomly sampled with replacement. ParameterSampler is used in RandomizedSearchCV's _run_search() which is called by its parent class BaseSearchCV in fit().

RandomizedSearchCV Unit Test Suite

Summary of Unit Test Coverage

Test Number	Description
Test Set 1	Repetition Off and Even Distribution With 1 Param Dictionary
Test Set 2	Repetition Off and Even Distribution With 2 Param Dictionaries
Test Set 3	Repetition Off and Even Distribution With 3 Param Dictionaries
Test Set 4	Repetition Off and Uneven Distribution With 2 Param Dictionaries
Test Set 5	Repetition Off and Uneven Distribution With 4 Param Dictionaries
Test Set 6	Repetition Off and Uneven Distribution With 3 Param Dictionaries
Test Set 7	Repetition On With 1 Param Dictionary (Distribution must be Equal)
Test Set 8	Repetition On With 2 Param Dictionaries (Distribution must be Equal)
Test Set 9	Repetition On With 4 Param Dictionaries (Distribution must be Equal)
Test Set 10	Repetition On with Non-List Parameters* With 4 Param Dictionaries

* In this test case, a script distribution is used as the corresponding min_sample_leaf value rather than a list.

When building scikit-learn from source with the bugfix applied, all test sets defined above pass, and additionally, the existing scikit-learn unit test suite also passes.

Unit Test Acceptance Conditions

Each test set provides a list of param_distributions, the amount of selection iterations and a repetition flag. These attributes are passed into testRandomizedSearchCV and used to create a RandomizedSearchCV estimator. The estimator is then fitted and the testing application verifies 3 to 4 acceptance conditions based on its result. First, the distribution of selected parameters for each dictionary must be close to the identified even distribution percentage. Next, the number of selected parameters is verified; in which each selected parameter must belong to one of the original param_distribution dictionaries. Finally if the repetition flag is set to false, then the selected params must not contain duplicates.

Running and Reading Unit Tests

Our Unit Test suite has been included within a dedicated testing folder within our A3 directory. To run these unit tests, please first build scikit-learn from source using the main branch of our scikit-learn project fork, and then use ‘python3 -W ignore testRandomizedSearchCVUnit.py’ to run the Unit test suite. (This ignores some unrelated, built in deprecation warnings which are unrelated to the feature)

Note in the original scikit-learn source code without our bugfix implemented, the RandomizedSearchCV component does not have a without_replacement argument. Since our bugfix implements this new class argument, this component can be instantiated slightly differently. As such, please select whether or not the bugfix has been applied to the scikit-learn source code when prompted by the testing script.

RandomizedSearchCV Acceptance Testing Suite

Summary of Acceptance Test Coverage

Test Number	**Description**
Test Set 1	Few Iterations, With Replacement
Test Set 2	Many Iterations, with Replacement
Test Set 3	Different List Values for params2, With Replacement
Test Set 4	Different List Values for params1, With Replacement
Test Set 5	Few Iterations, Without Replacement
Test Set 6	Many Iterations, Without Replacement
Test Set 7	Different List Values for params2, Without Replacement
Test Set 8	Different List Values for params1, Without Replacement

Acceptance Test Coverage and Acceptance Conditions

Our acceptance test suite centres around four main test cases and demonstrates most importantly that the RandomizedSearchCV now samples from each provided dictionary roughly an equal number of times, and secondarily that the values provided in these dictionaries do not impact the selection process. As such, our first two acceptance test cases are performed over a small (50) and larger (100) number of iterations, while the second and third cases provide parameter dictionaries with alternate values. The second set of four test cases (5-8) perform the same tests, but use the option to sample with replacement.

This script loads an iris dummy dataset, creates a randomForestClassifier, and a RandomizedSearchCV which takes a list of multiple dictionaries as param_distributions. It then fits the dummy data, and demonstrates how often values from each dict were selected during fitting. After the fix is applied, the split between each dict is roughly equal (50%), as we would statistically expect to see.

Running and Reading Acceptance Tests

Our Acceptance Test suite has been included within a dedicated testing folder within our A3 directory. To run these acceptance tests, please first build scikit-learn from source using the main branch of our scikit-learn project fork, and then use ‘python3 -W ignore testRandomizedSearchCVAcceptance.py’ to run the acceptance testing script. (This ignores some unrelated, built in deprecation warnings which are unrelated to the feature)

Note in the original scikit-learn source code without our bugfix implemented, the RandomizedSearchCV component does not have a without_replacement argument. Since our bugfix implements this new class argument, this component can be instantiated slightly differently. As such, please select whether or not the bugfix has been applied to the scikit-learn source code when prompted by the testing script.