CSCD01 Deliverable 3

Songzhif

Zhukevi6

Gaoyife5

xuchunan

Xuxinzhe

wangs314

Table of Contents

Feature Request 1

#14257 [Enhancement] Group aware Time-based cross validation

Link: https://github.com/scikit-learn/scikit-learn/issues/14257

Description:

TimeSeriesSplit is a feature that allows users to split their training data into a sequence of training steps increasing in data size to ensure the model is being trained correctly. However TimeSeriesSplit can only split data into set data sizes (set as an input argument). GroupTimeSeriesSplit will perform the same TimeSeriesSplit but will group by unique values.

- The argument “groups” in the split method should indicate the group number of each element in the input dataset.
- The data which is in the same group should not appear in both the training set and the test set at the same time.

The GroupTimeSeriesSplit class should have the same parameters as TimeSeriesSplit.

Changes:

doc/modules/cross_validation.rst:

- Add documentation of GroupTimeSeriesSplit into the cross_validation’s user guide

doc/modules/group_time_series_split.rst:

- Add user guide for GroupTimeSeriesSplit

examples/model_selection/plot_cv_indices.py:

- Add GroupTimeSeriesSplit to example cross validation strategies

```
137   cvs = [KFold, GroupKFold, ShuffleSplit, StratifiedKFold,
138          GroupShuffleSplit, StratifiedShuffleSplit, TimeSeriesSplit,
139          GroupTimeSeriesSplit]
```

sklearn/model_selection/__init__.py:

- Add GroupTimeSeriesSplit to sklearn.model_selection (import GroupTimeSeriesSplit)

```
45              'train_test_split',
46              'check_cv',
47              'GroupTimeSeriesSplit']
48
```

sklearn/model_selection/tests/test_split.py:

- Add test case

- In _split we create a new class called GroupTimeSeriesSplit built upon existing TimeSeriesSplit functionality.

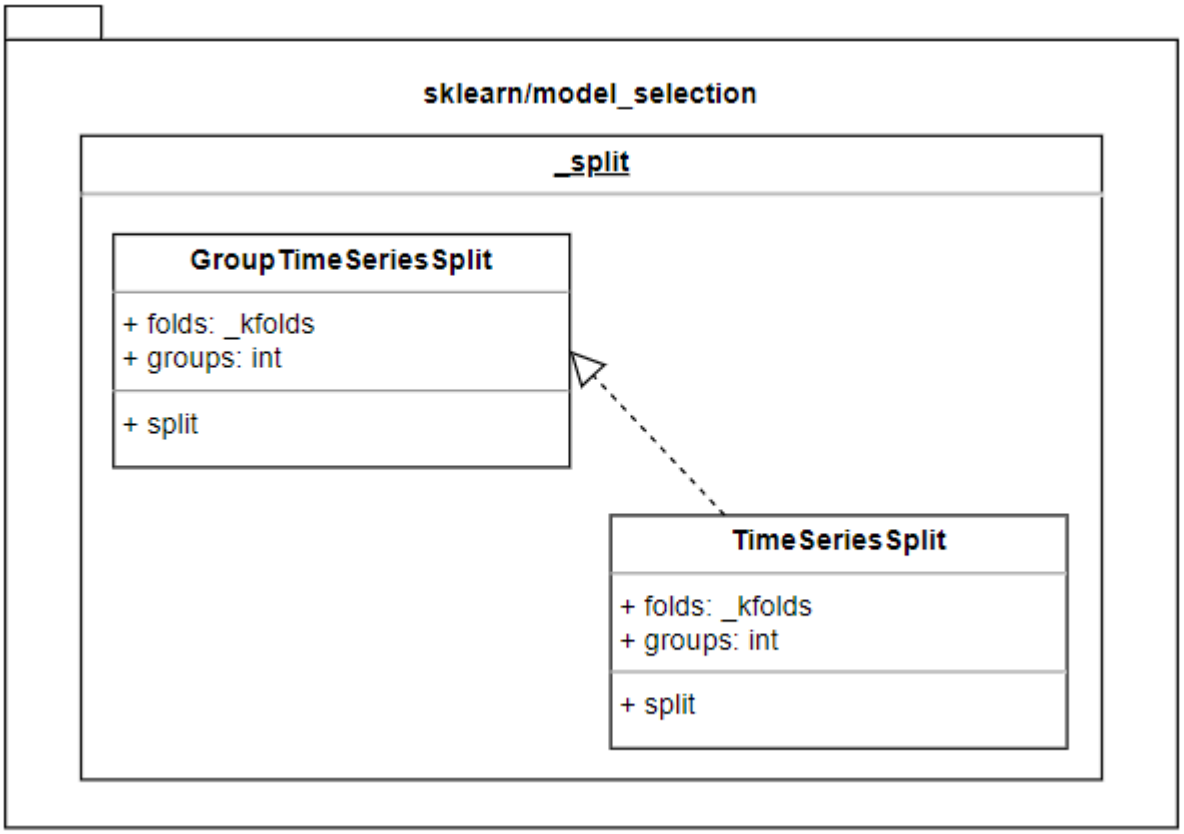
```
892  class GroupTimeSeriesSplit(_BaseKFold):
893      """
894      GroupTimeSeriesSplit
895
```

sklearn/model_selection/_spilt.py:

- Add a new class called GroupTimeSeriesSplit, which is a group variation of the existing TimeSeriesSplit class. Implement the `split(X, y, groups)` method to split the given data into train and test subsets based on the provided group pattern.
- `groups` needs to be validated with the conditions of 1. Has the same length as X and 2. Repeated elements need to be consecutive

Diagram:

New class GroupTimeSeriesSplit feature builds upon existing TimeSeriesSplit



Feature Request 2

#597 [Performance] Fibonacci Heaps for Ball Tree

Link: https://github.com/scikit-learn/scikit-learn/issues/597

Description:

The Ball Tree algorithm is a method of performing the Nearest Neighbor search. Current implementation relies on Dijkstra's algorithm which achieves its peak theoretical speed with a Fibonacci heap. So Ball Tree Nearest Neighbor search could potentially be sped up using Fibonacci heaps instead of the current implementation.

- an implementation of Fibonacci heap already exists in the sklearn/utils/graph_nearest_neighbor.pyx cython file.
- It would be refactored to fit requirements, and be called whenever Dijkstra's algorithm is used. This solution would "catch" all instances of different classification algorithms involving BallTree.
- Dijkstra's algorithm is only used in the graph_shortest_path.pyx file.

Changes:

BallTree itself requires no change, it is only when performing the Nearest Neighbor search on a constructed BallTree instance, that we want to change the data structures used.

In the graph.py utility file, graph_shortest_path is imported:

```
15   from .graph_shortest_path import graph_shortest_path  # noqa
```

The implementation of Dijkstra's algorithm regarding graph_shortest_path is found in the graph_shortest_path.pyx

File:
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/utils/graph_shortest_path.pyx

In this file we want to change the current implementation of min_heap into fib_heap.

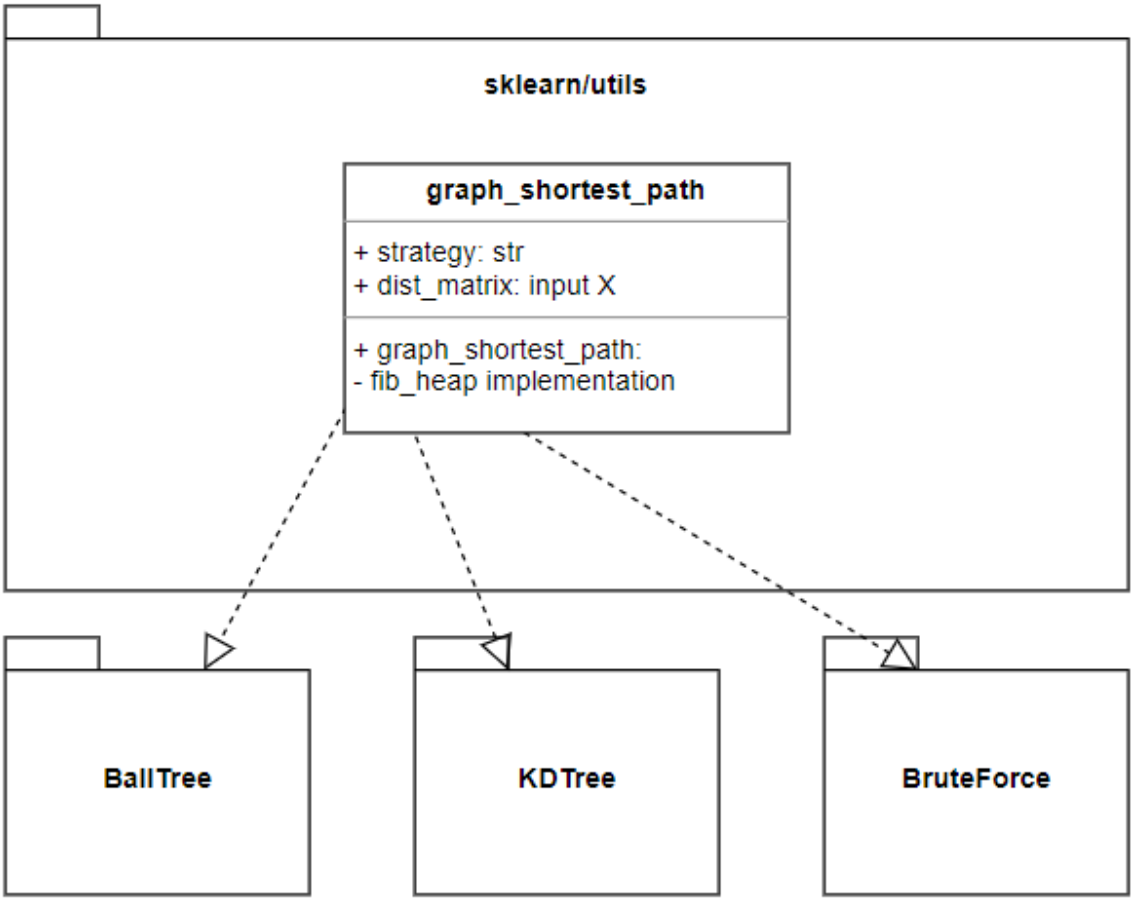
```
30  def graph_shortest_path(dist_matrix, directed=True, method='auto'):
31      """
32      Perform a shortest-path graph search on a positive directed or
33      undirected graph.
34
```

When successfully implemented, the amortized runtime of Nearest Neighbor should be lowered.

```
>>> nbrs = NearestNeighbors(n_neighbors=2, algorithm='ball_tree').fit(X)
>>> distances, indices = nbrs.kneighbors(X)
```

Diagram:

All implementation methods are within the class of graph_shortest_path



Feature Choice:

#14257

Through implementing feature #14257, we could have a better understanding of the module “model_selection” and the mechanism of cross validation in scikit learn. The cross validation is a significant part of scikit learn and more worthwhile for us to spend more time, since it is able to increase the utilization rate of sample data and could be useful in all other estimations of scikit learn. The classes which are already implemented could help us understand how cross validation works and what exactly we should do quickly, and thus provide a higher-quality result.

#597

The feature #597 required some knowledge on the ball tree and Fibonacci Heap, which may take us considerable time. We are definately willing to learn more from the assignment by trying both features, but the time is limited. Additionally, the constants of Fibonacci heap’s complexity are rather high, require complicated implementation, and lose their amortized benefits when used in applications that call delete_min frequently. In practice, the current implementation of min-heap and priority queues works faster and are simpler. Therefore, we decide to work on the more worthy one, feature #14257.

Unit tests are in a3/scikit-learn/sklearn/model_selection/tests/test_split.py

Acceptance Test:

Check number of splits

Steps:

1. Create an array: X = np.array([[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]])
2. Create an array: y = np.array([1, 2, 3, 4, 5])
3. Create groups: groups = np.array([0, 1, 1, 2, 3])
4. Create a GroupTimeSeriesSplit: group_timeseriessplit = GroupTimeSeriesSplit(max_train_size=None, n_splits=3, test_size=None)
5. Print the number of splits: print(group_timeseriessplit.get_n_splits(X, y, groups))

Expected outcome:

- 3 should be printed as the correct output.

Check indices of train and test data

Steps:

1. Follow the steps 1-4 in **Check number of splits.**
2. Print the indices of train and test data:

   for train_index, test_index in group_timeseriessplit.split(X, y, groups):

   print("TRAIN:", train_index, "TEST:", test_index)

   X_train, X_test = X[train_index], X[test_index]

   y_train, y_test = y[train_index], y[test_index]

Expected outcome:

- TRAIN: [0] TEST: [1, 2]

  TRAIN: [0, 1, 2] TEST: [3]

  TRAIN: [0, 1, 2, 3] TEST: [4] should be printed as the correct output.

Check indices of train and test data with different max_train_size, n_splits, and gap

Steps:

1. Follow the steps 1-3 in **Check number of splits**.
2. Create a GroupTimeSeriesSplit with 2 as maximum size for a single training set, 2 as number of splits, 1 as number of samples to exclude from the end of each train set before the test set:

   group_timeseriessplit = GroupTimeSeriesSplit(max_train_size=2, n_splits=2, test_size=None, gap=1)

3. Print the indices of train and test data:

   for train_index, test_index in group_timeseriessplit.split(X, y, groups):

   print("TRAIN:", train_index, "TEST:", test_index)

   X_train, X_test = X[train_index], X[test_index]

   y_train, y_test = y[train_index], y[test_index]

Expected outcome:

- TRAIN: [0] TEST: [3]

  TRAIN: [0, 1, 2] TEST: [4] should be printed as the correct output.

Create a GroupTimeSeriesSplit with no groups

Steps:

1. Make an array: X = [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10], [11, 12], [13, 14]]
2. Try to create a GroupTimeSeriesSplit next(GroupTimeSeriesSplit(n_splits=7).split(X))

Expected outcome:

- ValueError: The 'groups' parameter should not be None should be raised.

Create a GroupTimeSeriesSplit with more folds than groups

Steps:

1. Create groups: groups = np.array([1, 1, 1, 2, 2])
2. Create two arrays: X = y = np.ones(len(groups))
3. Try to create a GroupTimeSeriesSplit with more folds than groups:

   next(GroupTimeSeriesSplit(n_splits=3).split(X, y, groups))

Expected outcome:

- ValueError: Cannot have number of folds=4 greater than the number of samples=2 should be raised.

Create a GroupTimeSeriesSplit with non-continuous groups

Steps:

1. Create non-continuous groups: groups = np.array(['a', 'a', 'a', 'a', 'a', 'a', 'b', 'b', 'b', 'b', 'b', 'c', 'c', 'c', 'c', 'a', 'd', 'd'])
2. Create two arrays: X = y = np.ones(len(groups))
3. Try to create a GroupTimeSeriesSplit with non-continuous groups:

   next(GroupTimeSeriesSplit(n_splits=3).split(X, y, groups))

Expected outcome:

- ValueError: The groups should be continuous. Each unique element in the group should be inside one continuous interval should be raised.

User guides are in a3/scikit-learn/doc/modules/group_time_series_split.rst

And a3/scikit-learn/doc/modules/cross_validation.rst