A1 Report

Scikit-learn

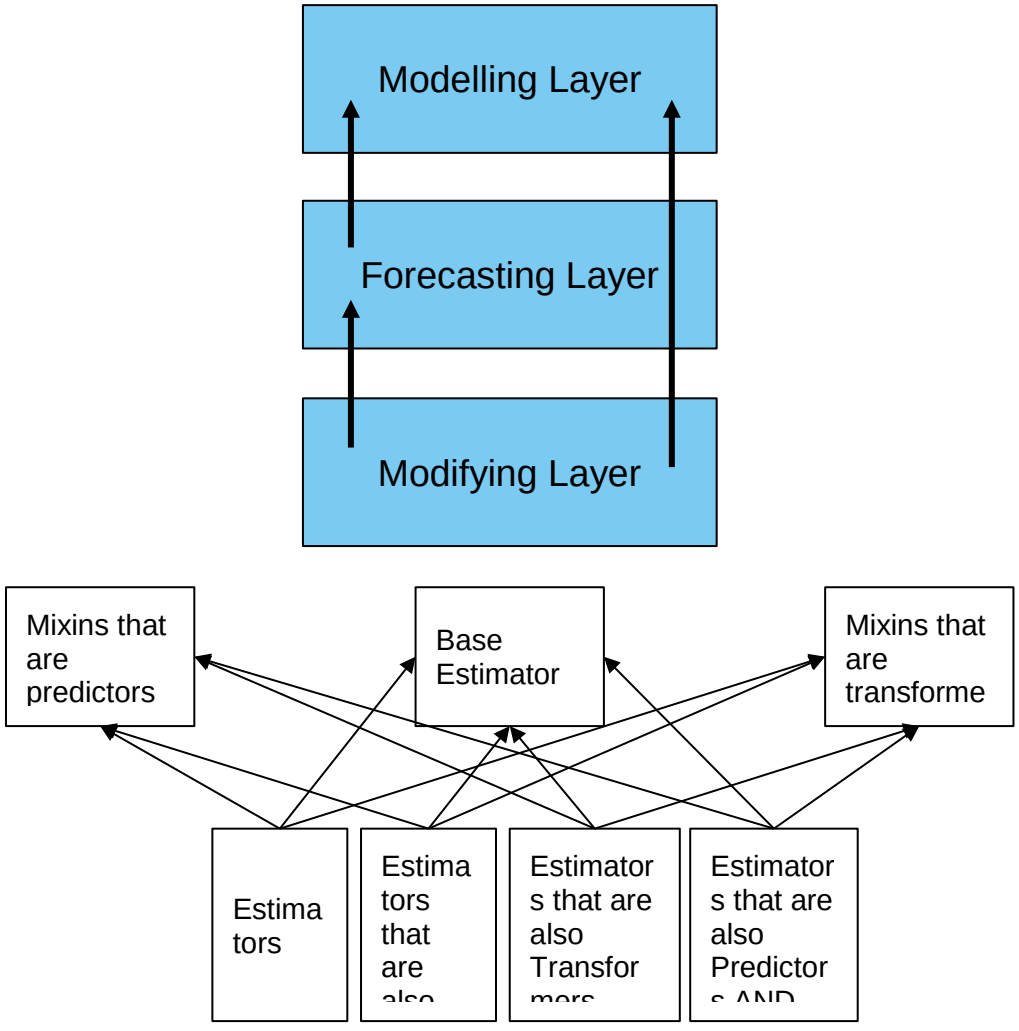
Table of Contents

Overall Architecture

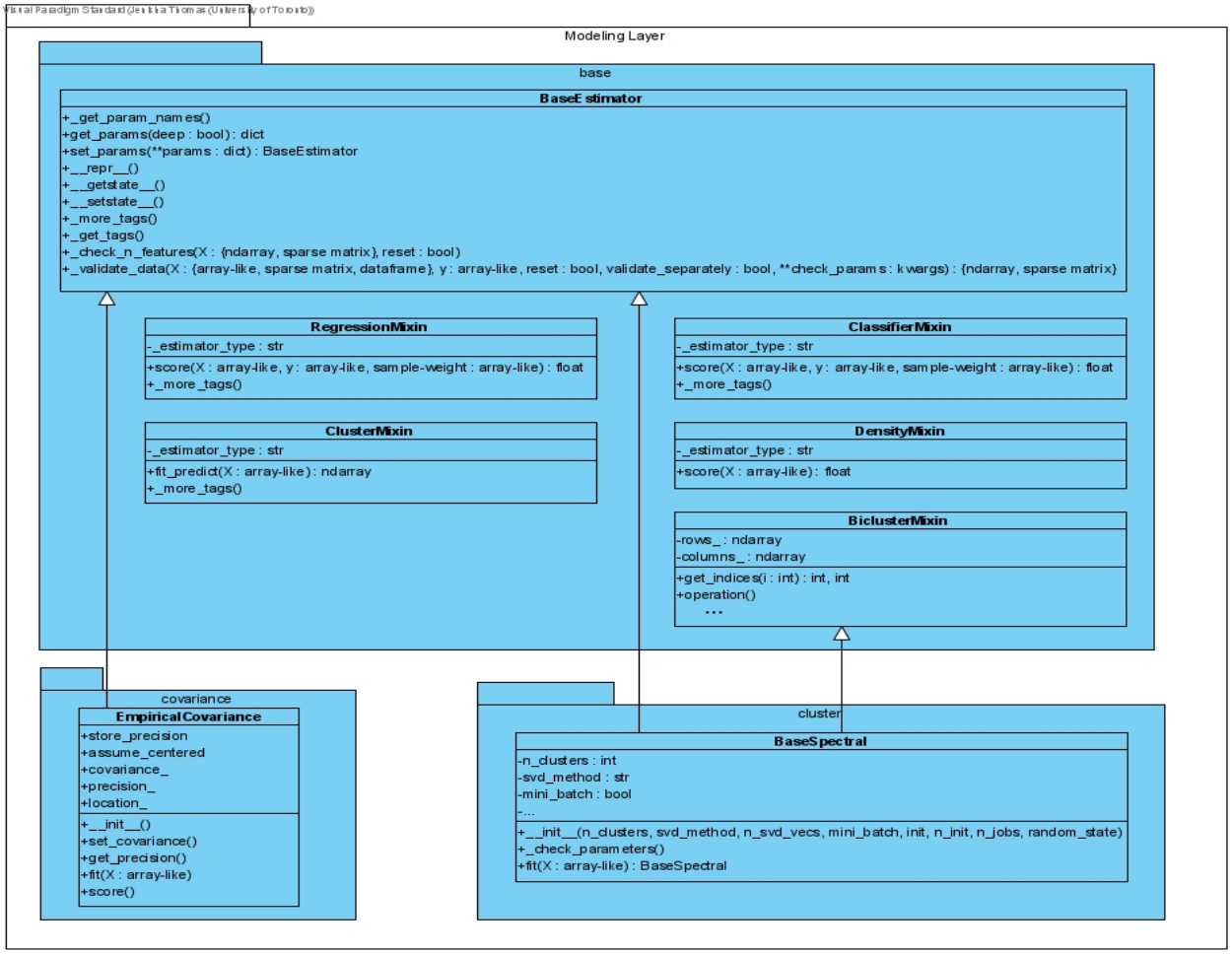
First of all, what is scikit-learn? Scikit-learn is a free, open-source library used with Python to implement machine learning. Among other features, it provides tools for modeling data, such as classification, regression, and clustering algorithms.

One of the recurring themes in the design of scikit-learn is the idea of programming to an interface and not to an implementation.

The overall architecture of scikit-learn can be broken up into three main layers; the modeling layer, forecasting layer, and modifying layer.



Modeling Layer - Estimators



The modeling layer of scikit-learn contains the interfaces and classes responsible for training data and creating models using a variety of learning algorithms. The main class of this layer is the BaseEstimator class, which is the blueprint for all estimators, predictors, and transformers classes in scikit-learn. The modeling layer consists of BaseEstimator and the estimator classes inherited from it. Each of these classes implements the fit method, taking in arguments the sample matrix (X) and target values (y) to fit the data to the estimator. Alongside this, they also implement the set_parameters method provided in BaseEstimator, to set data-independent parameters for each specific type of estimator.

This layer is connected to almost all of the classes in scikit, as all 3 main objects (estimators, predictors, and transformers) of scikit-learn inherit from BaseEstimator. The main package in this level is the base package, which contains BaseEstimator and multiple mixin classes, to allow for duck typing. Thus, to create an estimator of a specific type, scikit utilizes the BaseEstimator class and the related mixin classes.

Recall that almost all of scikit's classes are inherited from the BaseEstimator class. Since we cannot show all of them in a single diagram effectively, we have selected two specific classes to focus on. In the UML above, we can see how the BaseSpectral and EmpiricalCovariance estimators demonstrate the architecture discussed above.

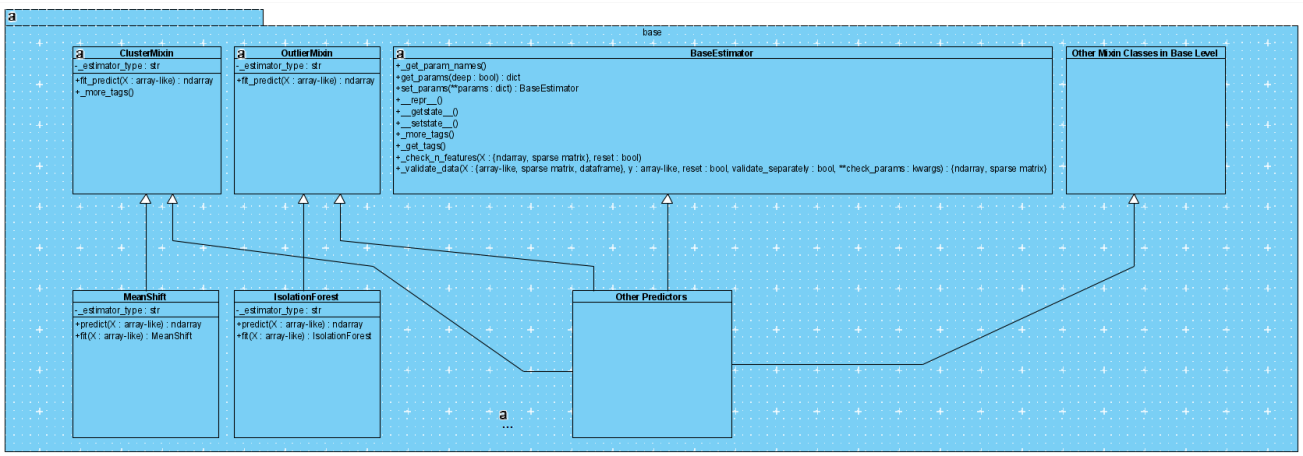
The BaseSpectral class is the base class for biclustering algorithm estimators. Like all estimators, it implements a fit method, taking in only one argument of the sample matrix.

The EmpiricalCovariance class implements a maximum likelihood estimator, which approximates the covariance matrix of a set of data. Its fit method is similar to the BaseSpectral class and takes in only one argument of the sample matrix.

As discussed above, the mixin classes, RegressionMixin, ClusterMixin, ClassifierMixin, DensityMixin, BicLusterMixin are inherited alongside the BaseEstimator to create the different types of estimators used throughout scikit-learn. In the UML above we see this with the BaseSpectral class, inheriting both BaseEstimator and BiclusterMixin.

The links to the files discussed above:
BaseEstimator: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L141
RegressionMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L506
ClusterMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L560
ClassifierMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L470
DensityMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L705
BiclusterMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L590
EmpiricalCovariance: https://github.com/scikit-learn/scikit-learn/.../empirical_covariance.py#L102
BaseSpectral: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/.../_bicluster.py#L86

Forecasting Layer - Predictors

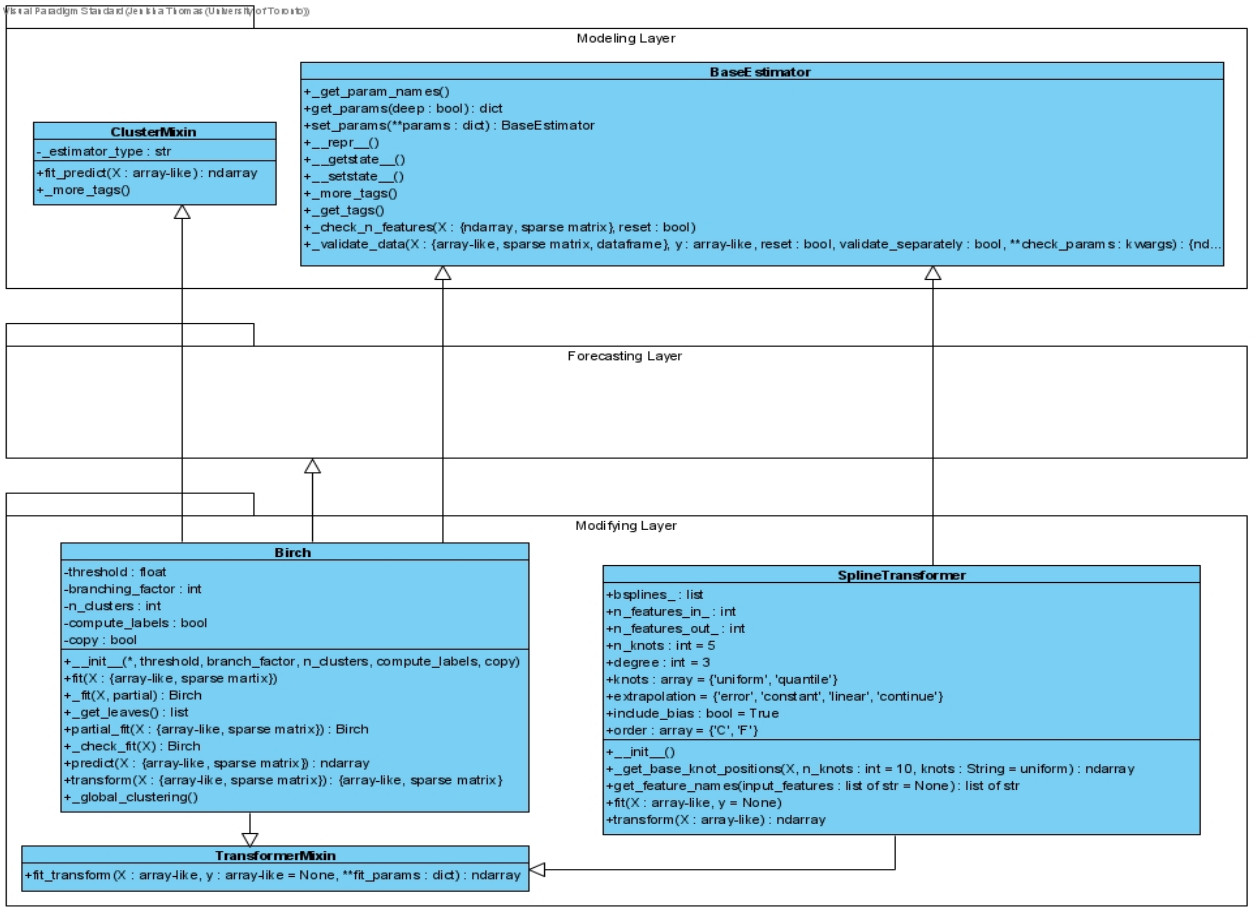


The forecasting layer of scikit-learn does the work for predicting results. For any model to work, it must implement a forecasting method called "predict" or "fit_predict". The predictor classes are actually just estimators that have the predict and/or fit_predict methods. The predict method takes test features and predicts their results. It can also take an input feature vector and return predicted labels.

Similar to the modeling layer, at the topmost level, of the forecasting layer we have the BaseEstimator class (since Predictors are really just Estimators) as well as Mixin classes. For the forecasting layer, we are concerned with the ClusterMixin and the OutlierMixin, since they both implement the fit_predict method, which makes them Predictors. Further, they can also be inherited by other classes that are Predictors. For example, ClusterMixin is inherited by the MeanShift and AffinityPropagation classes which also implement the predict method, so they're Predictors. Similarly, EllipticEnvelope and IsolationForest classes inherit from OutlierMixin and implement "predict" methods too. For the purposes of the diagram above, we have kept it simple to fit its purpose.

The links to the files discussed above:
BaseEstimator: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L141
ClusterMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L560
OutlierMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L727
IsolationForest: https://github.com/scikit-learn/scikit-learn/.../sklearn/ensemble/_iforest.py#L27
MeanShift: https://github.com/scikit-learn/scikit-learn/.../sklearn/cluster/_mean_shift.py#L243

Modifying Layer - Transformers



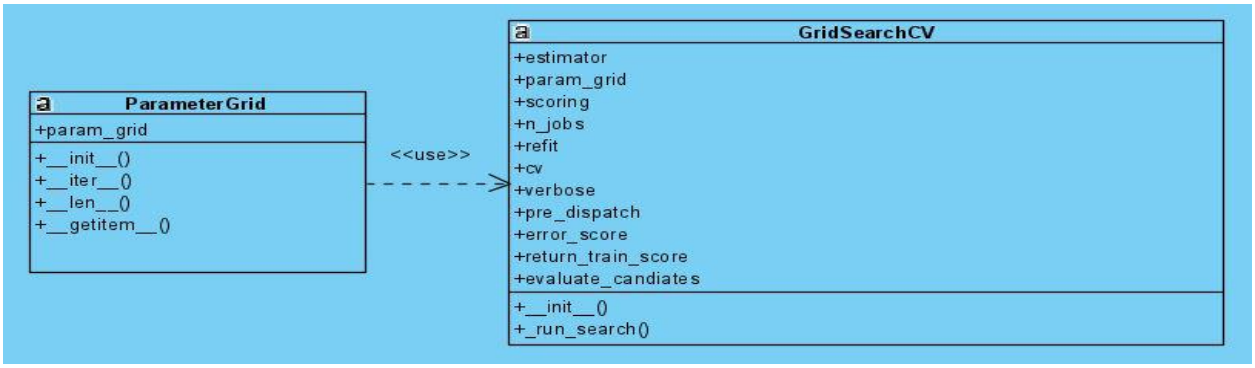
The modifying layer of scikit-learn contains the interfaces and classes responsible for transferring data. The main interface of this layer is the TransformerMixin class, which is the blueprint for all estimators implementing transformers. There are many classes that inherit the TransformerMixin class and implement the transform method. However, we decided to showcase two classes as all the other classes in this layer will have the same structure. The transform methods take in input data and target values, which are used to modify the input accordingly.

We see that the Modifying Layer inherits the Modelling Layer. This means that all the classes in this layer inherit the BaseEstimator class and the TransformerMixin class. Thus the classes implement both the fit and the transform method. In addition, some of the classes in this layer contain the predict method, creating a dependency on the Forecasting Layer. This is showcased in the Birch class in the diagram.

The links to the files discussed above:
BaseEstimator: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L141
ClusterMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L560
Birch: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/cluster/_birch.py#L335
SplineTransformer: https://github.com/scikit-learn/scikit-learn/blob/.../_polynomial.py#L23
TransformerMixin: https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L668

Design Patterns

Iterator Pattern



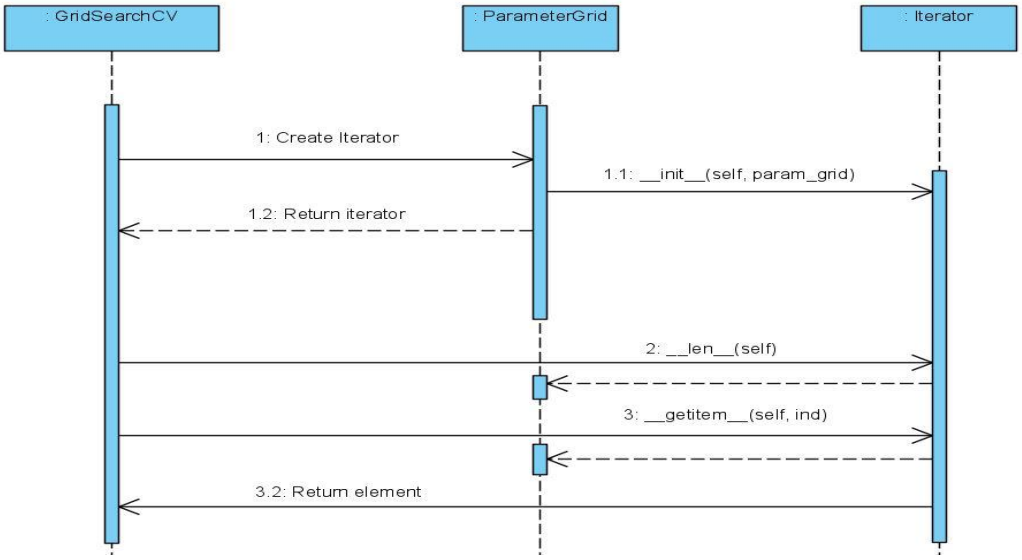
Class Diagram demonstrating Iterator Pattern

The iterator pattern is a behavioural design that allows for traversing the elements of a collection (a group of objects) without having to expose the implementation details.

Example Usage In Codebase:

File Path: scikit-learn/sklearn/model_selection/_search.py at line 118 in def __iter__self()

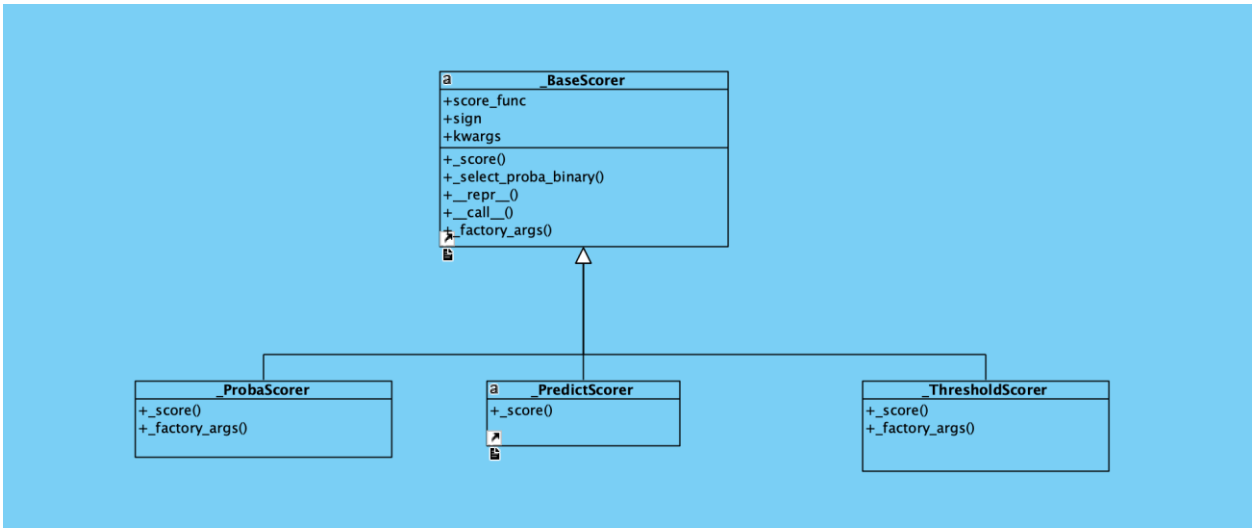
The __iter__ method in the ParameterGrid class iterates over the points in the grid. More specifically, we are iterating through the elements of a collection (grid in this case). The order of the generated parameter combinations is deterministic.



Sequence Diagram demonstrating Iterator Pattern

GridSearchCV uses ParameterGrid, the iterator to iterate over the points in the grid. This is implemented in the __iter__ method. The method __len__(self) returns the number of points on the grid. The method __getitem__(self, ind) returns a list of parameters that would be indth iteration.

Factory Pattern



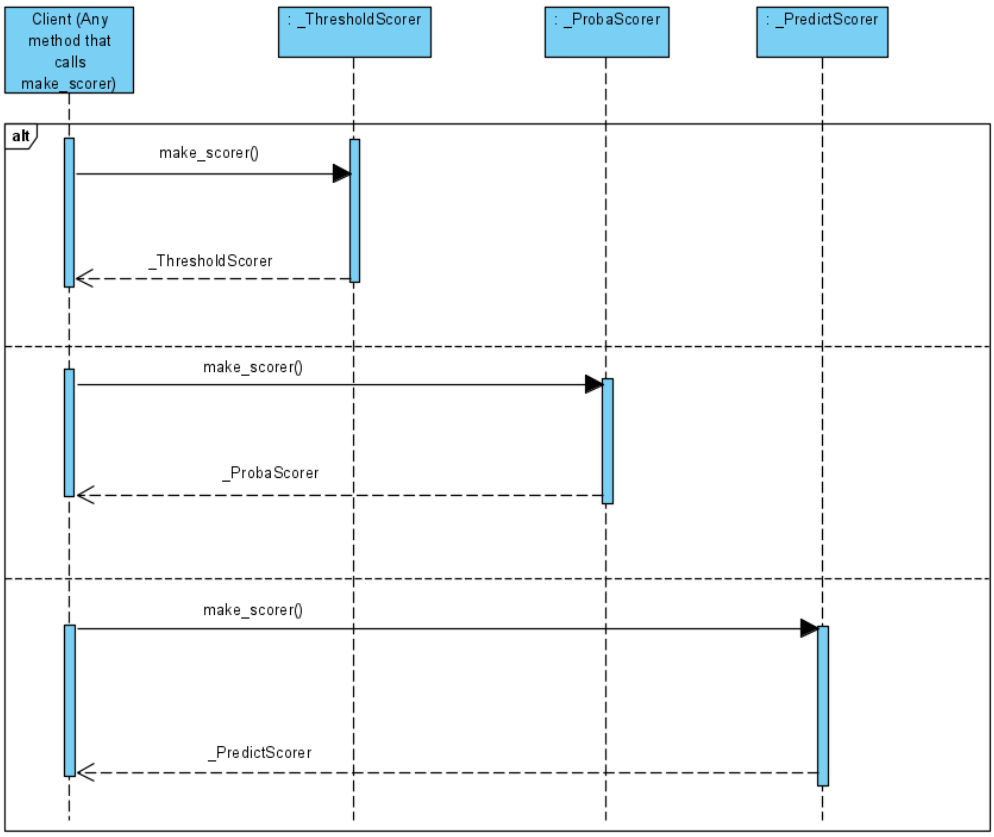
Class Diagram demonstrating Factory Pattern

The factory pattern is a creational design pattern that allows us to create objects without exposing the creation logic to the client.

Example Usage In Codebase:

File Path: scikit-learn\sklearn\metrics_scorer.py at line 538 in make_scorer method

An example of where the factory pattern is used is in the make_scorer method as it returns different subclasses (_ProbaScorer, _ThresholdScorer, _PredictScorer) of _BaseScorer depending on certain conditions. For example, if needs_proba is true, make_scorer returns the _ProbaScorer class. The make_scorer method allows us to create a subclass of _BaseScorer without exposing the creation logic to the client.

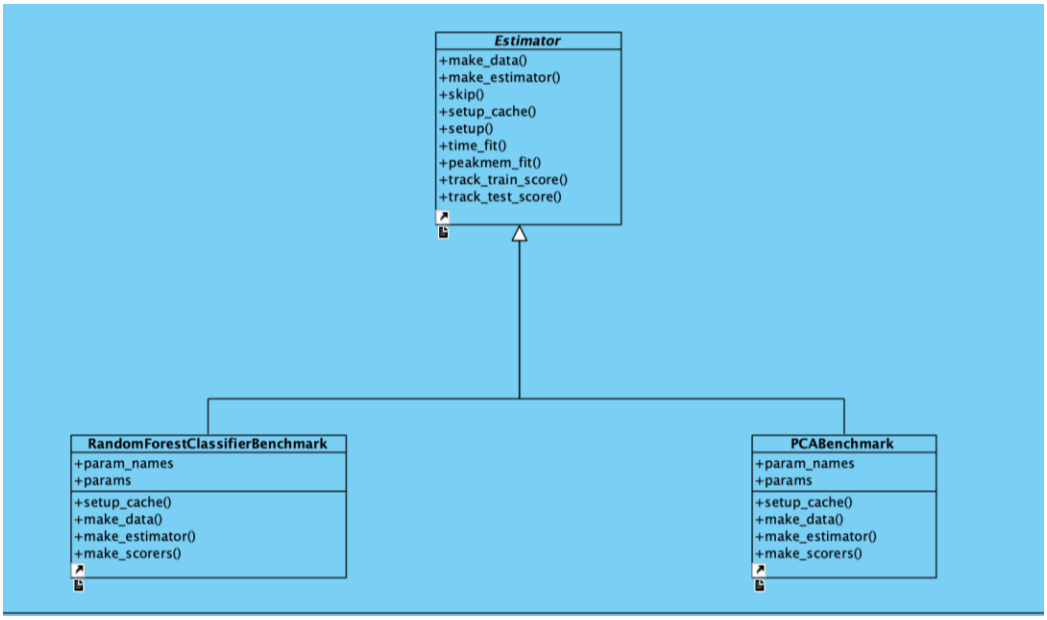


Sequence Diagram demonstrating Factory Pattern

Note: In the sequence diagram above, the Client refers to any other method that calls the make_scorer().

As per the factory design pattern, the object creation details are hidden from the client. So in the diagram above, the client calls the make_scorer method, which returns a _ThresholdScorer, _ProbaScorer, or _PredictScorer class.

Decorator Pattern



Class Diagram demonstrating Decorator Pattern

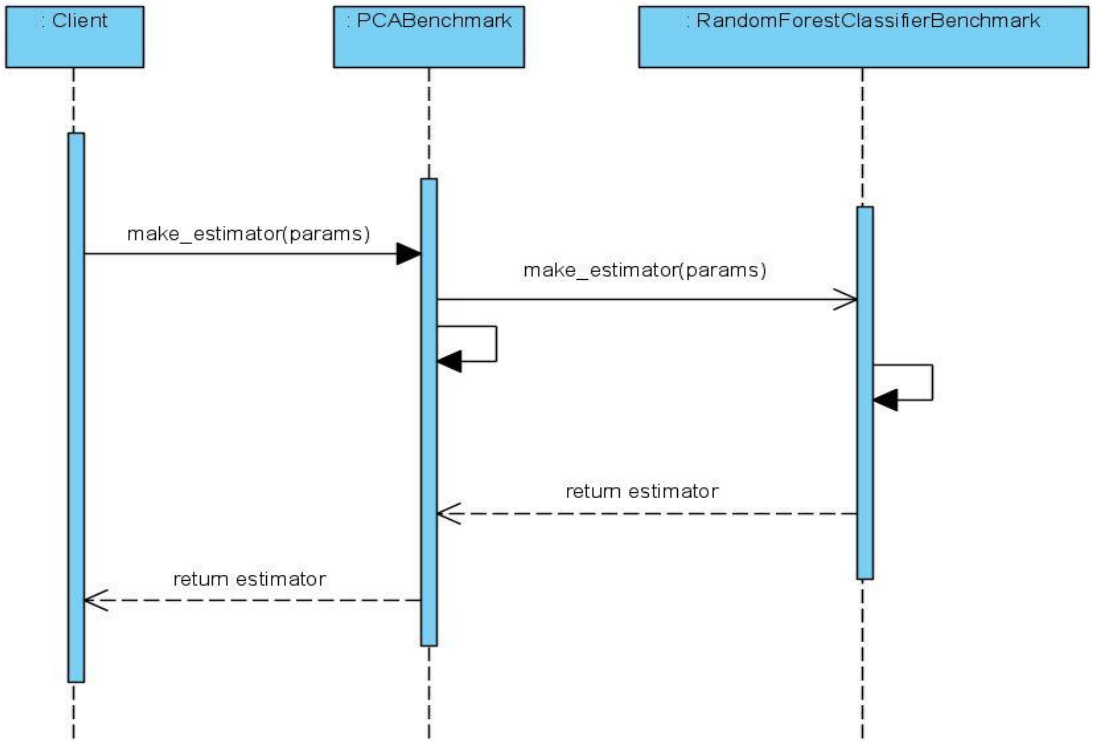
The decorator pattern is a structural pattern that allows for attaching additional functionalities to an object dynamically.

Example Usage In Codebase:

File Path:

- scikit-learn/asv_benchmarks/benchmarks/ensemble.py at line 34 in def make_estimator()
- scikit-learn/asv_benchmarks/benchmarks/decomposition.py at line 23 in def make_estimator()
- scikit-learn/asv_benchmarks/benchmarks/common.py at line 114 in def make_estimator()

The Estimator abstract class defines a method (make_estimator in this case) and each subclass of Estimator (RandomForestClassifierBenchmark and PCABenchmark) defines its own version of the make_estimator method.



Sequence Diagram demonstrating Decorator Pattern

The Client executes make_estimator() over PCABenchmark. The PCABenchmark executes the same operation over RandomForestClassifierBenchmark . The RandomForestClassifierBenchmark executes the make_estimator() operation. The PCABenchmark executes the make_estimator. The Client receives a decorated object from all the Decorators.