SCIKIT-LEARN DESIGN DOC

CSCD01 Winter 2021

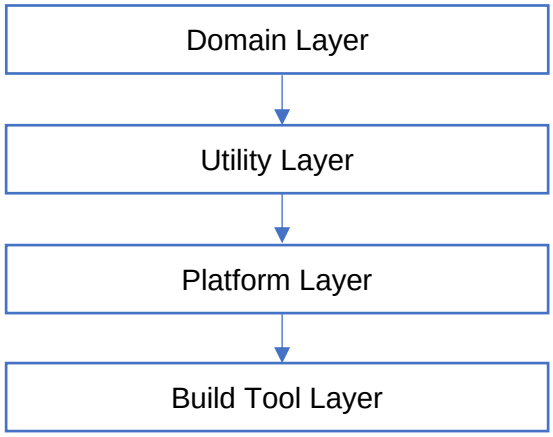
Spaghetti Code

A design overview of the Scikit-Learn open source project

Saad Ali, Jesse Francispillai, Adam Ah-Chong, Andrew Gao, Julian Barker, Laphonso Dominic Reyes

System Architecture

Scikit-learn is a machine-learning library constructed in Python and provides learning algorithms through various analytical and statistical implemented methods such as Regression, Classification, Clustering, and others. The system itself is composed of several modules organized in a fashion that lists the core purposes in a 4-layer architecture:

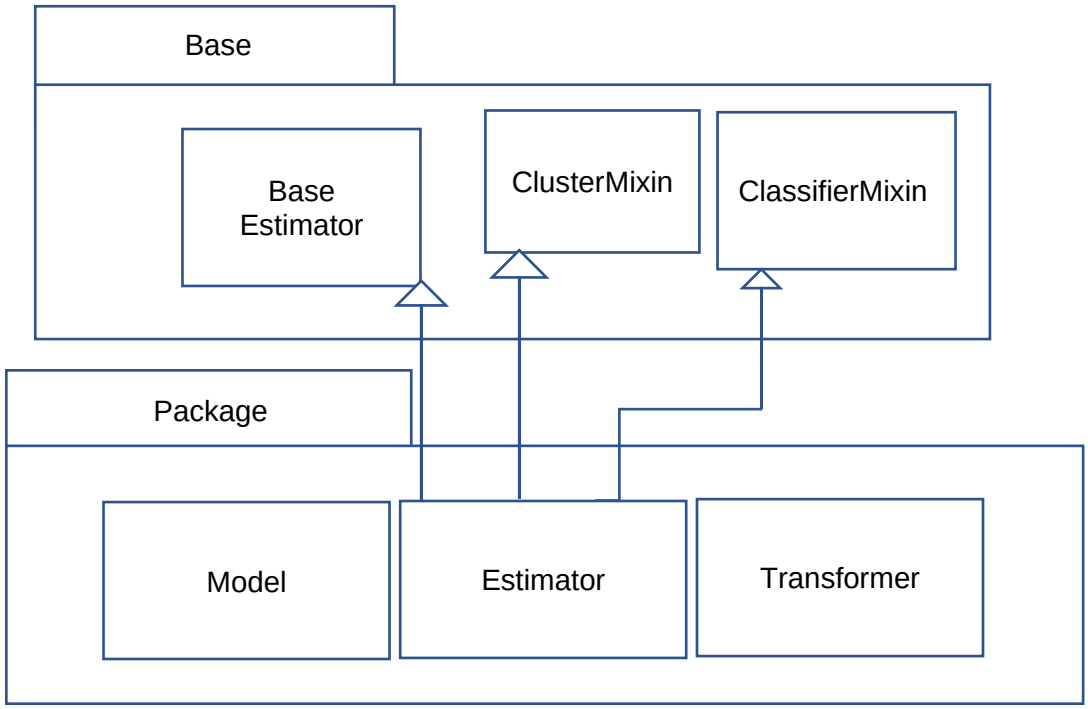


Resources used:
https://delftswa.gitbooks.io/desosa-2017/content/scikit-learn/chapter.html

The 4-Layer Architecture

The Domain Layer

The **Domain Layer** categorises the inner workings of scikit-learn with respect to their computations, data analysis, and predictions. Each package within sklearn module represents a unique fitness model. If a class within a package is an Estimator or a Predictor, it inherits from the BaseEstimator class in sklearn's *base.py*. Classes often also inherit combinations of Mixin classes that are specialized for the specific model they are being used for.



As there are many packages inheriting the abstraction provided by classes in the base module, the various types of Model Selection, Estimators, and Transformers specify a specialization in the data they are computing. Each package represents its own unique functionality, often Fitness Models, each with its own methods for training containing and special utilities. Their structures are always derived from the universal API described in the sklearn *base* package. Different types of fitness models are useful for different types of data and situations. For example, the PCA Learning fitness model is generally more accurate and precise than the sparser Dictionary Learning model, which focuses more on efficiency and readability of data.

The Utility Layer

The **Utility layer** provides a sense of supports to the domain layer by grouping the code that deals with testing, validation, and configurations of such computations. These implementations are classified as utilities, where their role lies in providing developers tools to validate input, efficient mathematical operations, random sampling, testing functions, and more. These described tools enable the domain layer to repeatedly use general computations throughout the project.

The Platform Layer

The **Platform layer** specifies the dependent projects that enables the inner workings to function, i.e., the packages that are external to the core computations of scikit learn, but making them possible, like Python, Numpy, Scipy, and Matplotlib.

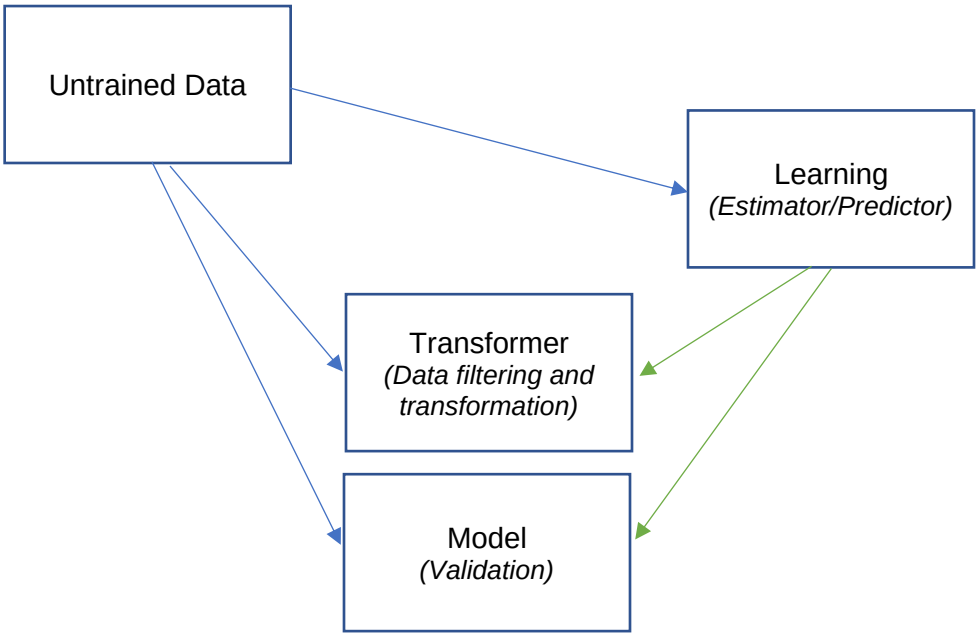
The Build Tool Layer

The **Build Tool Layer** consists of modules that enable successful and efficient building of this system to ensure the environment it is on may function as expected. Before operating scikit-learn for the first time, users must configure their environment and an appropriate build of scikit-learn.

Degree of Coupling

As previously stated, the base module provides an abstraction that enables the specification of specialized implementations of machine learning estimators, predictors, transformers, and modeling. As a result, this abstraction demonstrates a stand-alone behaviour with respect to its specialized implementations implying its efferent coupling e_c is extremely minimal. Conversely, every module that implements specialized functionalities demonstrates a high degree of afferent coupling a_c. Hence, the system's general architecture 's instability index $\frac{e_c}{e_c+a_c} \cong 0$. These findings suggests that the current methods and practices implemented in the design of scikit-learn indicate minimal dependencies between the specialized functionalities and their abstracted libraries.

Scikit-learn Workflow



Sklearn is a machine learning open-source software, whose repositories hold an impressive number of different takes on machine learning algorithms and processes.

Sklearn takes untrained data and runs it through one of its many fitness models. These models are contained in and processed by classes in each package which each process data in their own unique way.

The overall structure of any Sklearn machine learning model consists of up to three steps that can be run in any order or combination: the algorithm learns from untrained data, transforms said data if necessary, and validates the accuracy of the results of the learned model.

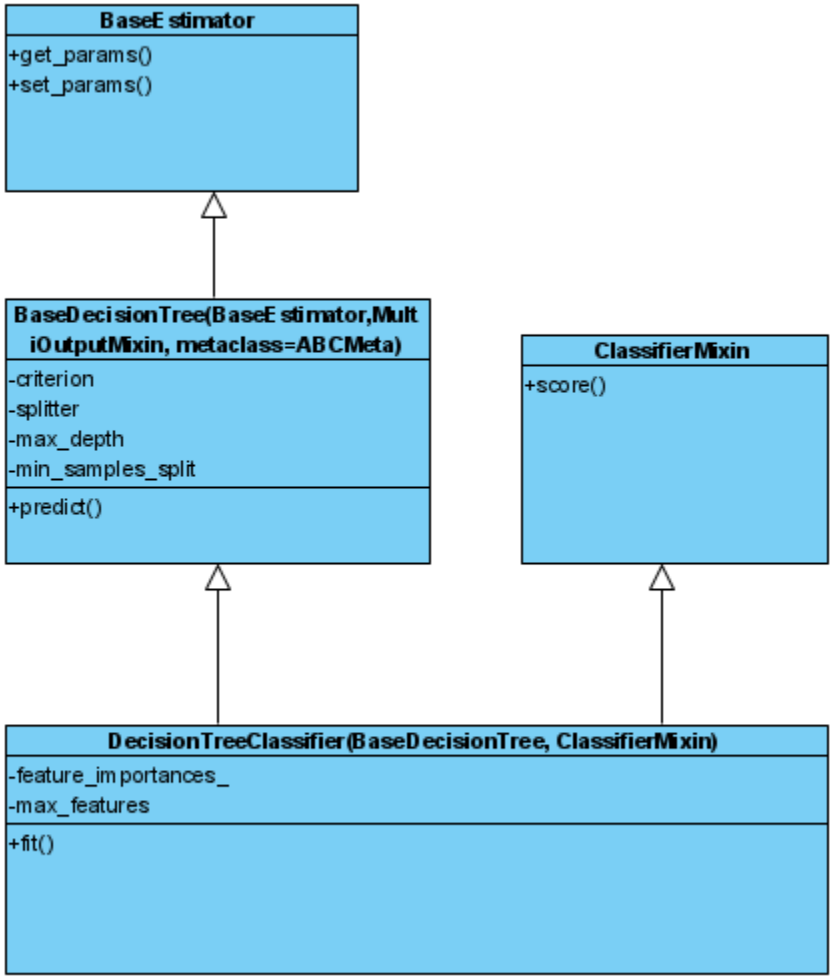
In order to do this, Sklearn fitness models are structured as in a way that they implement a combination of any number of the following interfaces:

- Estimator, a fitness model generally used for unsupervised learning
- Predictor, a fitness model generally used for supervised learning
- Transformer, for filtering or modifying data
- Model, used for finding the *goodness of fit* measure, or validation of a fitness model

Design Patterns

Decision Tree Design Pattern

One of the Design Patterns that Sklearn uses is the Decision Tree Design Pattern.

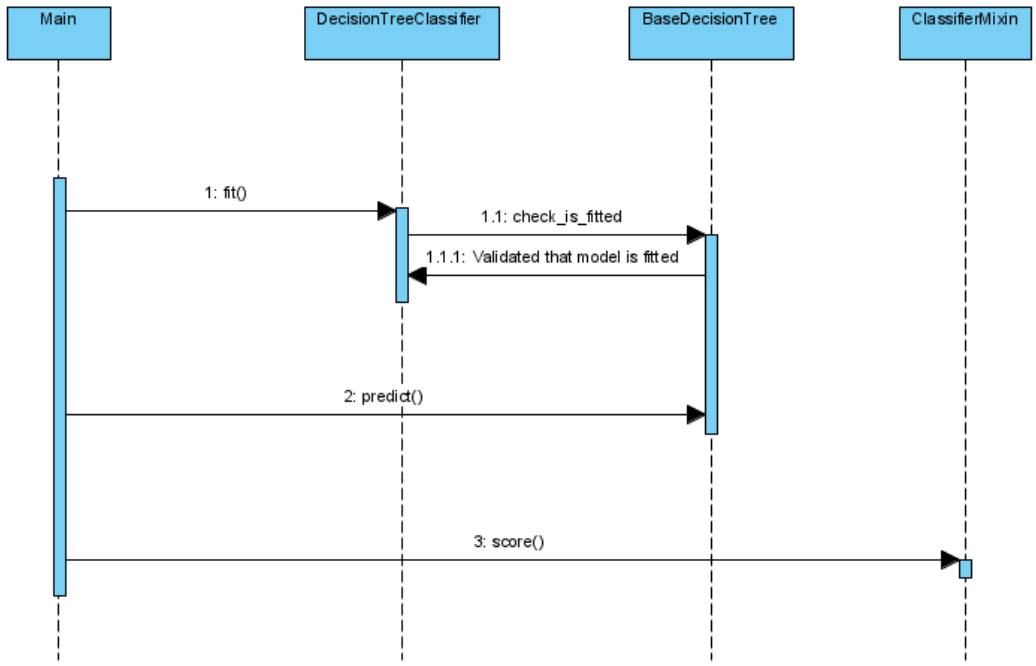


As you see from the UML the DecisionTreeClassifier builds upon the pre-defined BaseDecisionTreeEstimator and Mixins. The functionality is split between the various classes which all contain important functions for model predicting. At the top-level the DecisionTreeClassifier is a class that is used to perform a multi-class classification on a dataset. The fit method is contained which is the function is the main function where we pass in the training data for the algorithm to learn.

DecisionTreeClassifier itself takes in ClassifierMixin and a BaseDecisionTree. BaseDecisionTree contains the predict function where we take in the a set of X samples and returns a y value based on the fit() we passed in. ClassifierMixin is where the score function is contained which takes in the X samples and the y values and returns a mean value on the accuracy of our data. At the

lowest level BaseEstimator is the set and get functions for the parameters for the DecisionTree which are iterators to get and set the parameters required to do the model predicting.

The sequence diagram is as follows:
First we must create a DecisionTreeClassifier Object and call the fit function with our training data. After we can use the predict() to get our y values or pass in a y-value for the score() and get the mean score of the data passed in



Links to code above:

BaseEstimator:
https://github.com/scikit-learn/scikit-learn/blob/dac560551c5767d9a8608f86e3f253e706026189/sklearn/base.py#L141

BaseDecisionTree:
https://github.com/scikit-learn/scikit-learn/blob/dac560551c5767d9a8608f86e3f253e706026189/sklearn/tree/_classes.py#L80

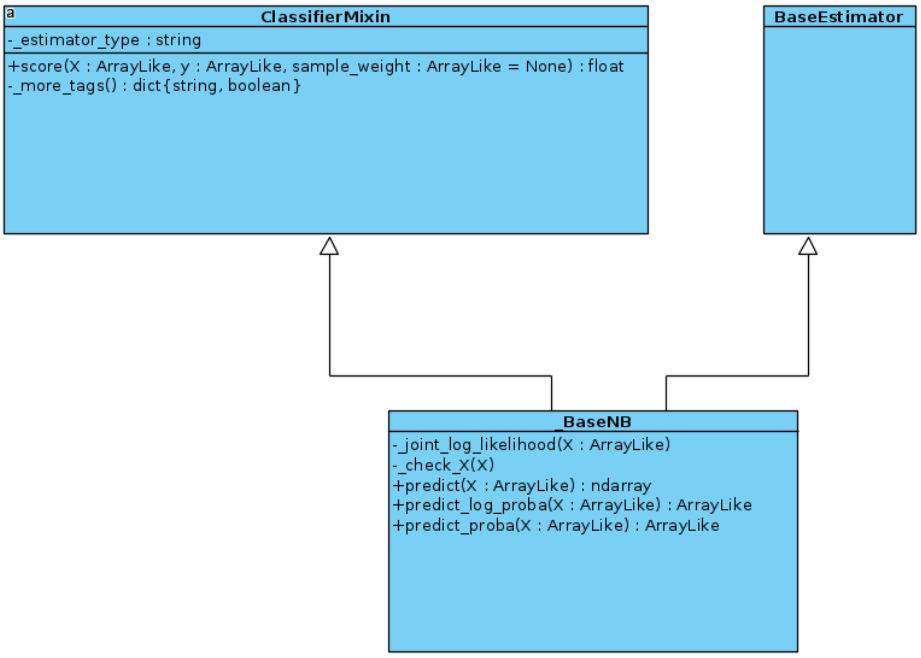
ClassifierMixin
https://github.com/scikit-learn/scikit-learn/blob/dac560551c5767d9a8608f86e3f253e706026189/sklearn/base.py#L470

DecisionTreeClassifier:
https://github.com/scikit-learn/scikit-learn/blob/dac560551c5767d9a8608f86e3f253e706026189/sklearn/tree/_classes.py#L607

Mixin Design Pattern

Scikit-learn uses a Mixin design pattern. A mixin is a class that implements one feature for classes to inherit. In Python, classes can inherit from multiple parent classes, so a mixin helps ensure there is no conflict in inheritance because when used, it must be the only parent class that implements its feature. Mixins are easily found because their class names must end with “Mixin”.

One example of a mixin used in Scikit-learn is the ClassifierMixin class in the base.py file. _BaseNB in the naive_bayes.py file inherits from ClassifierMixin and BaseEstimator. ClassifierMixin only implements a (default) method called “score” which returns a float that defines how accurate the fit/estimate is. BaseEstimator’s attributes/methods are left blank for brevity.

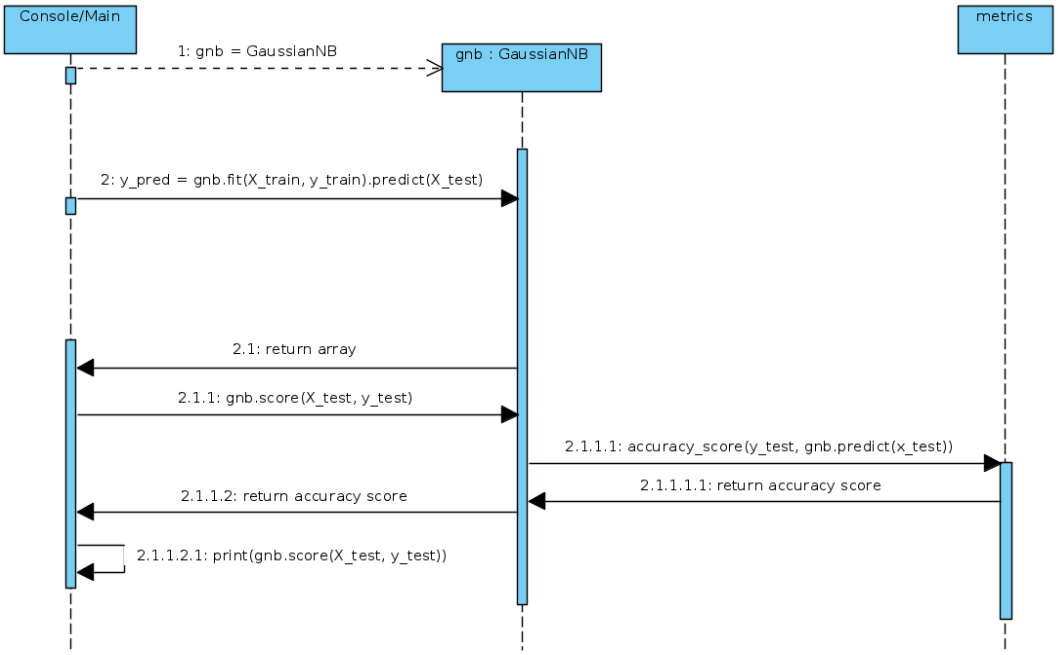


There are also other similar mixins in the base package that are used for different types of estimators (RegressorMixin for regression estimators, etc).

An example of the ClassifierMixin being used (1.9.1. Gaussian Naive Bayes from documentation, but changed to print score of fit using test data)

```
from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.naive_bayes import GaussianNB
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
random_state=0)
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(X_train, y_train).predict(X_test)
>>> print(gnb.score(X_test,y_test))
0.9466666666666667
```

The sequence diagram for this code (starting at gnb = GaussianNB()) is as follows:



In this case, GaussianNB inherits from _BaseNB which inherits from ClassifierMixin, and the "score" method implemented in ClassifierMixin is called.

Links to classes mentioned:

BaseEstimator:
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L141

ClassifierMixin:
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L470

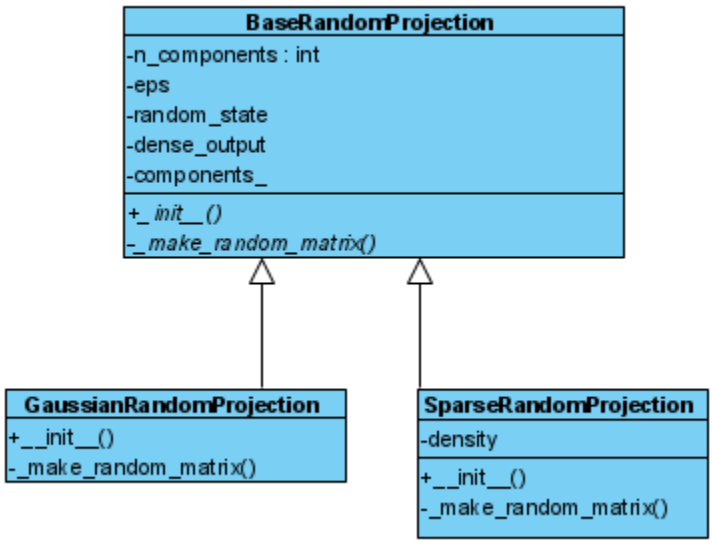
RegressorMixin:
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/base.py#L506

_BaseNB:
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/naive_bayes.py#L42

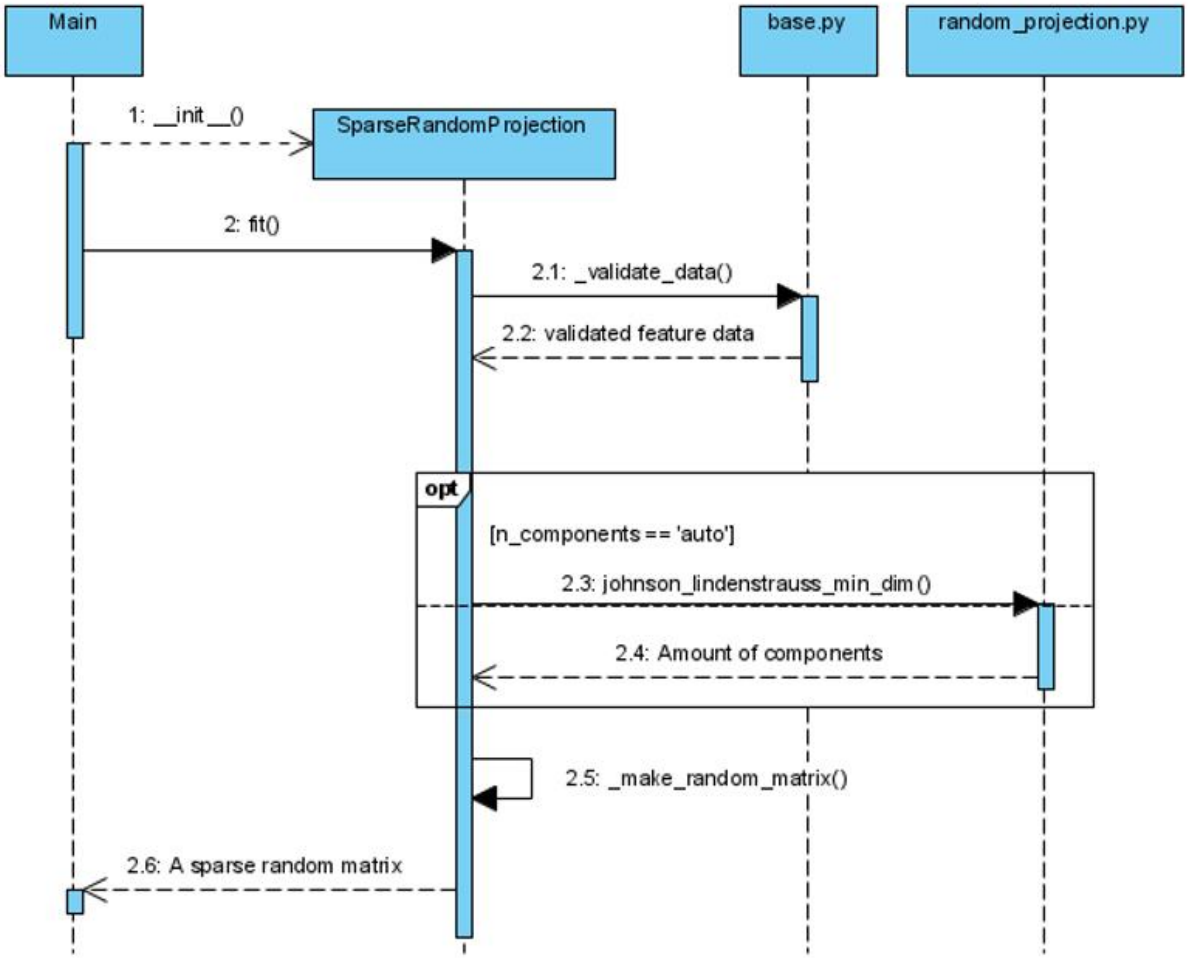
GaussianNB:
https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/naive_bayes.py#L118

Resources used:
https://www.residentmar.io/2019/07/07/python-mixins.html
https://scikit-learn.org/stable/modules/naive_bayes.html

Template Method Design Pattern



The BaseRandomProjection classes and its two subclasses use the Template Method design pattern. In BaseRandomProjection a fit() method is implemented which uses another method in BaseRandomProjection called _make_random_projection() which is an abstract method. The Template Method in BaseRandomProjection is fit(). In order to use the fit() method using Template Method design pattern, there must be a subclass of BaseRandomProjection that implements the _make_random_matrix() method but does not override the Template Method fit(). As we can see from the diagram, the two subclasses GaussianRandomProjection and SparseRandomProjection both have their own implementation of _make_random_matrix() but don't override fit(), therefore we can see that Template Method design pattern is used here.



Here is an example of the sequence of processes for a user instantiating a SparseRandomProjection and calling the template method fit() which is implemented in the parent class BaseRandomProjection. In fit() the method _make_random_matrix() is called, which is only defined in the subclasses of BaseRandomProjection

Links to classes mentioned:

BaseRandomProjection:
https://github.com/scikit-learn/scikit-learn/blob/dac560551c5767d9a8608f86e3f253e706026189/sklearn/random_projection.py#L293

GaussianRandomProjection:
https://github.com/scikit-learn/scikit-learn/blob/dac560551c5767d9a8608f86e3f253e706026189/sklearn/random_projection.py#L420

SparseRandomProjection:

https://github.com/scikit-learn/scikit-learn/blob/dac560551c5767d9a8608f86e3f253e706026189/sklearn/random_projection.py#L513