CSCD01: Assignment 1

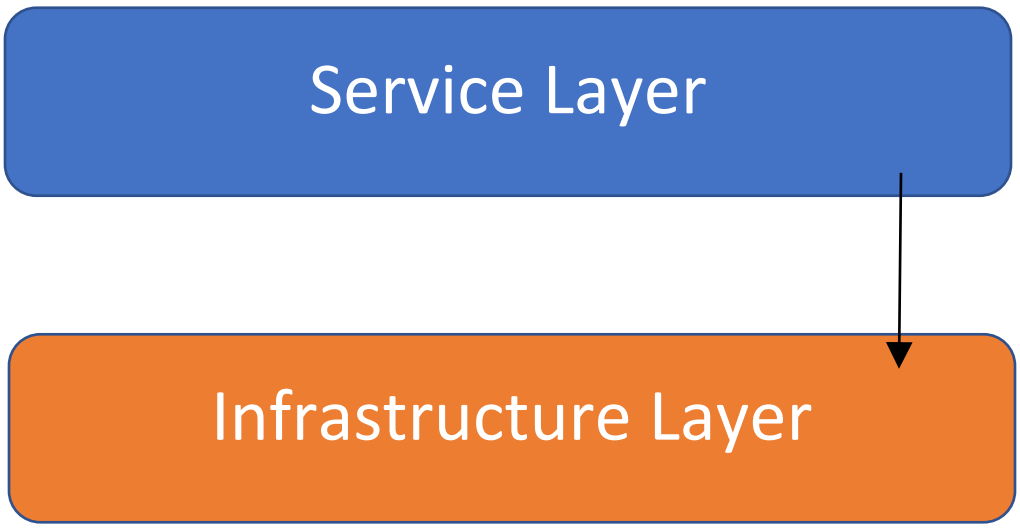
2021-02-11

Table of Contents:

General Architecture

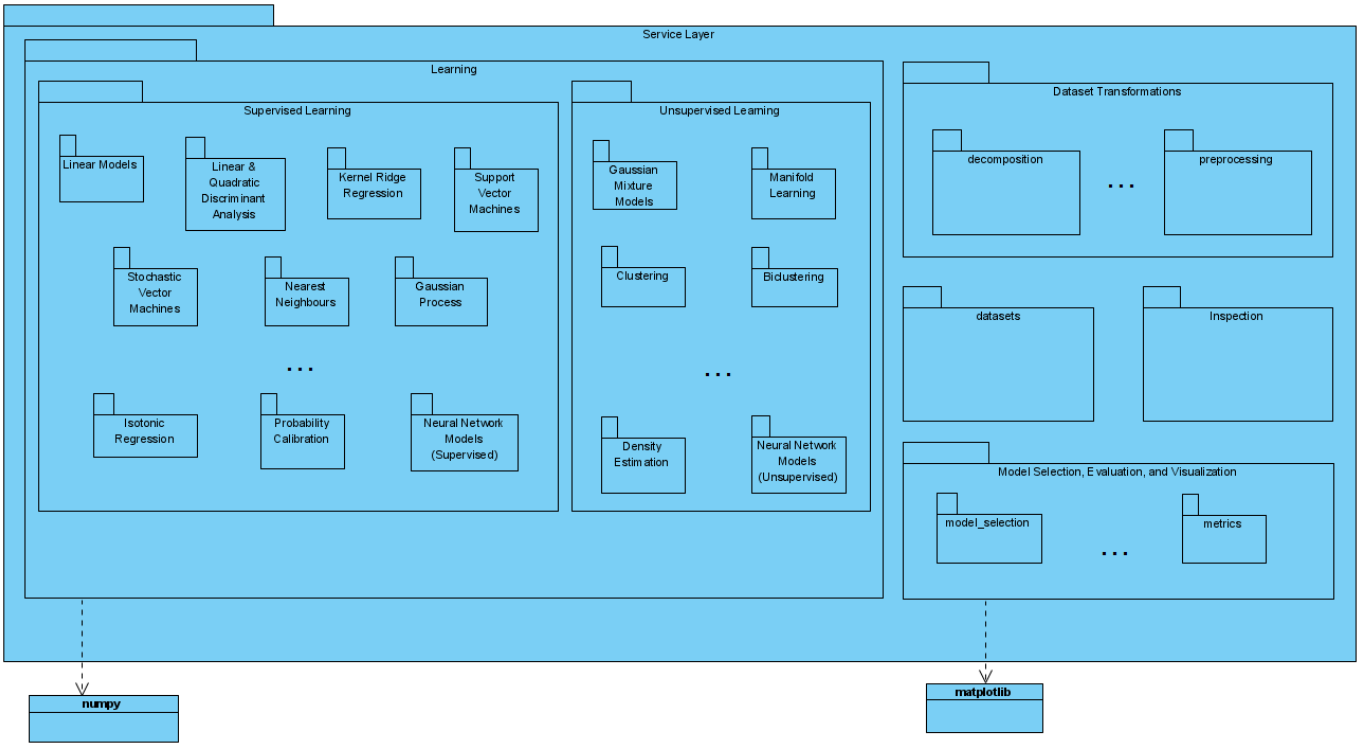
Scikit-learn is a Python library that offers machine learning practices. In general, the library expects the user to have knowledge on the appropriate practice for their given problem set (e.g. model fitting, predicting, etc.). Each practice has their own module, which the user can access through the sklearn library after importing it.

Inspecting the source code reveals that, in the broad sense, the architecture can be divided into two layers, each layer with their own responsibilities.



The sklearn library has no UI layer. Essentially, this means that the user only inputs data to the appropriate module, and the module will use such data to determine the data it outputs. As such, we can think of the modules provided as a Service Layer; the user interacts with the library through the services (machine learning modules) it offers.

Service Layer



The above diagram shows the architecture of the Service Layer. Users can find the appropriate module by understanding the problem set they face: are they looking for Supervised Learning or Unsupervised Learning methods? Under Unsupervised Learning, the library offers a variety of modules for different problem sets (Gaussian Mixture Models, Manifold Learning, etc.). Supervised Learning is similar, and though the diagram does not show it, we can also separate the modules under two different categories (packages), Classification and Regression. For the sake of simplicity, we show that they are all a part of Supervised Learning. The modules within the Service Layer depend on Numpy, which is shown as a dependency relationship.

The Service Layer also holds modules that work side-by-side with Learning; there are Data Transformation modules, which allow the user to preprocess data, reduce dimensionality of data, and more. The data could be provided by the user, or it could be from a dataset provided by the library itself. Finally, there are tools that help the user in performance evaluation, providing metrics, scoring, and curves, and tools that help the

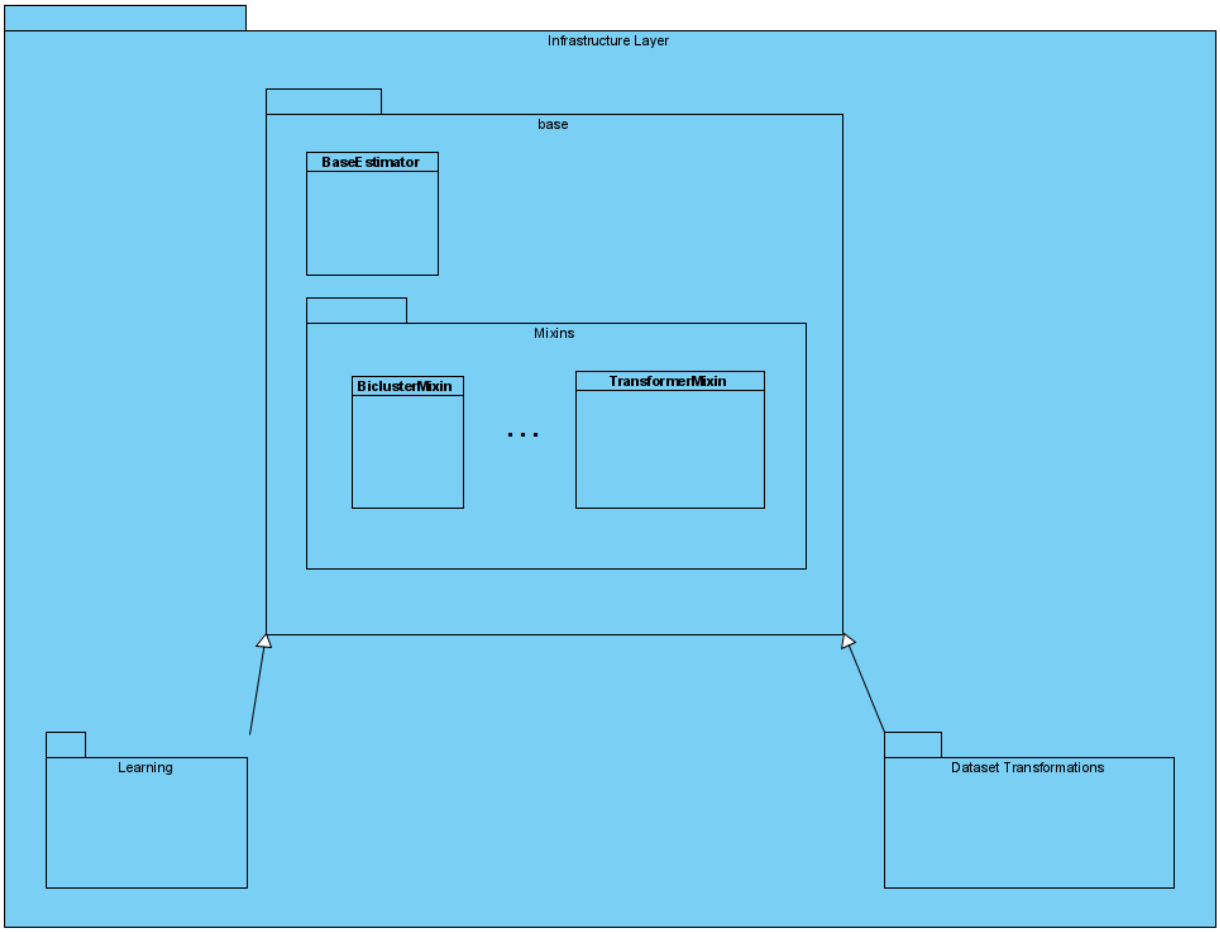
user understand predictions from a model and factors that affect them. Visualizations are offered through the matplotlib library.

A list of the modules, including the ones shown above, can be found at:

- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning (Supervised)
- https://scikit-learn.org/stable/unsupervised_learning.html (Unsupervised)
- https://scikit-learn.org/stable/model_selection.html (Model Selection, Evaluation)
- https://scikit-learn.org/stable/inspection.html (Inspection)
- https://scikit-learn.org/stable/visualizations.html (Visualization)
- https://scikit-learn.org/stable/data_transforms.html (Dataset Transformations)
- https://scikit-learn.org/stable/datasets.html (Datasets)

The source code can also be found by navigating through each individual module from the links above.

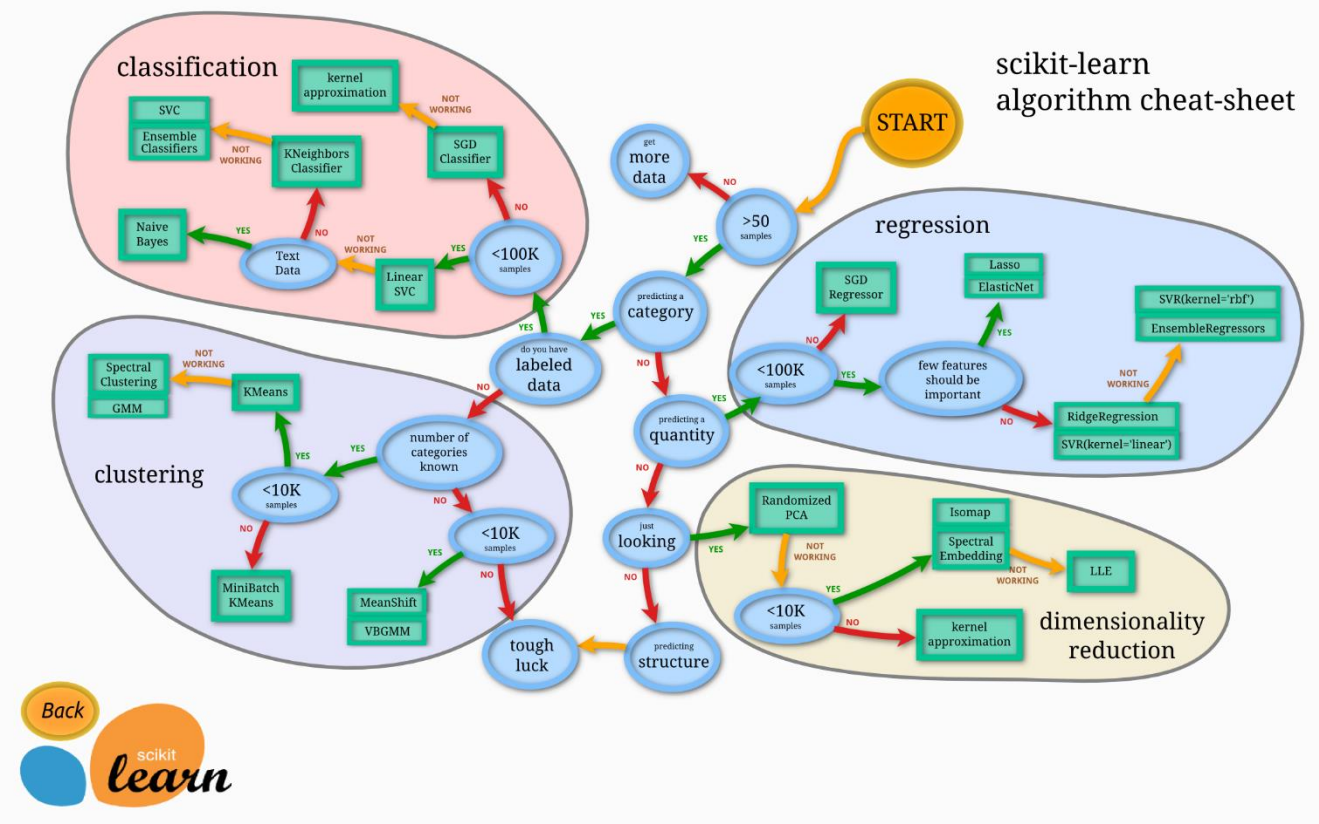
Infrastructure Layer



The above diagram depicts the Infrastructure Layer, which is meant to show the architecture with respect to the level of abstraction of various modules. While the users will interact with the Service Layer to use modules within the Learning package (high-level), the Learning package itself inherits from base, and is an implementation of an estimator based off the class BaseEstimator and various other Mixin classes (low-level). While users will typically use the higher level modules, it is possible to make use of lower level modules, perhaps to create custom estimators that differ from the ones offered by the library.

Put together, the two high-level UML diagrams give a basic idea on how the scikit-learn library operates. As mentioned in the beginning, the library expects the user to have knowledge on the appropriate practice for their given problem set when interacting with

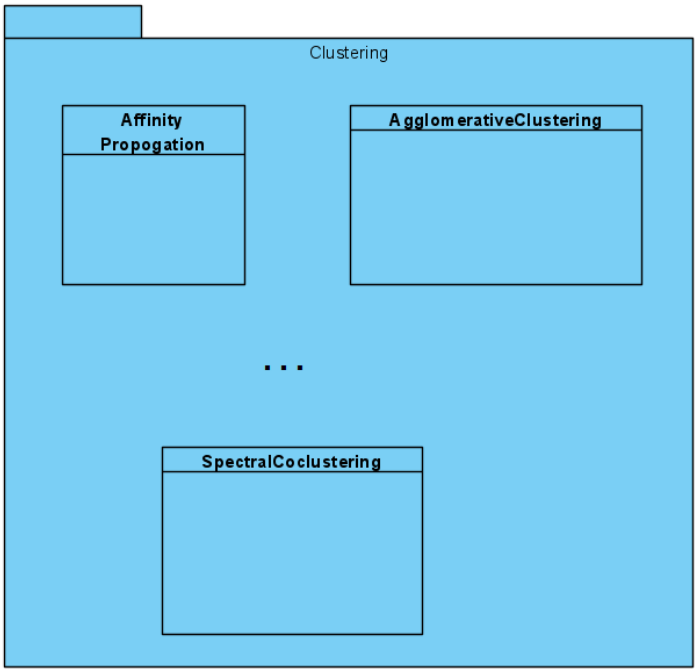
the Service Layer. However, the documents offer a flow-chart that works nicely with how the Service Layer is offered, as seen below:



SOURCE: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

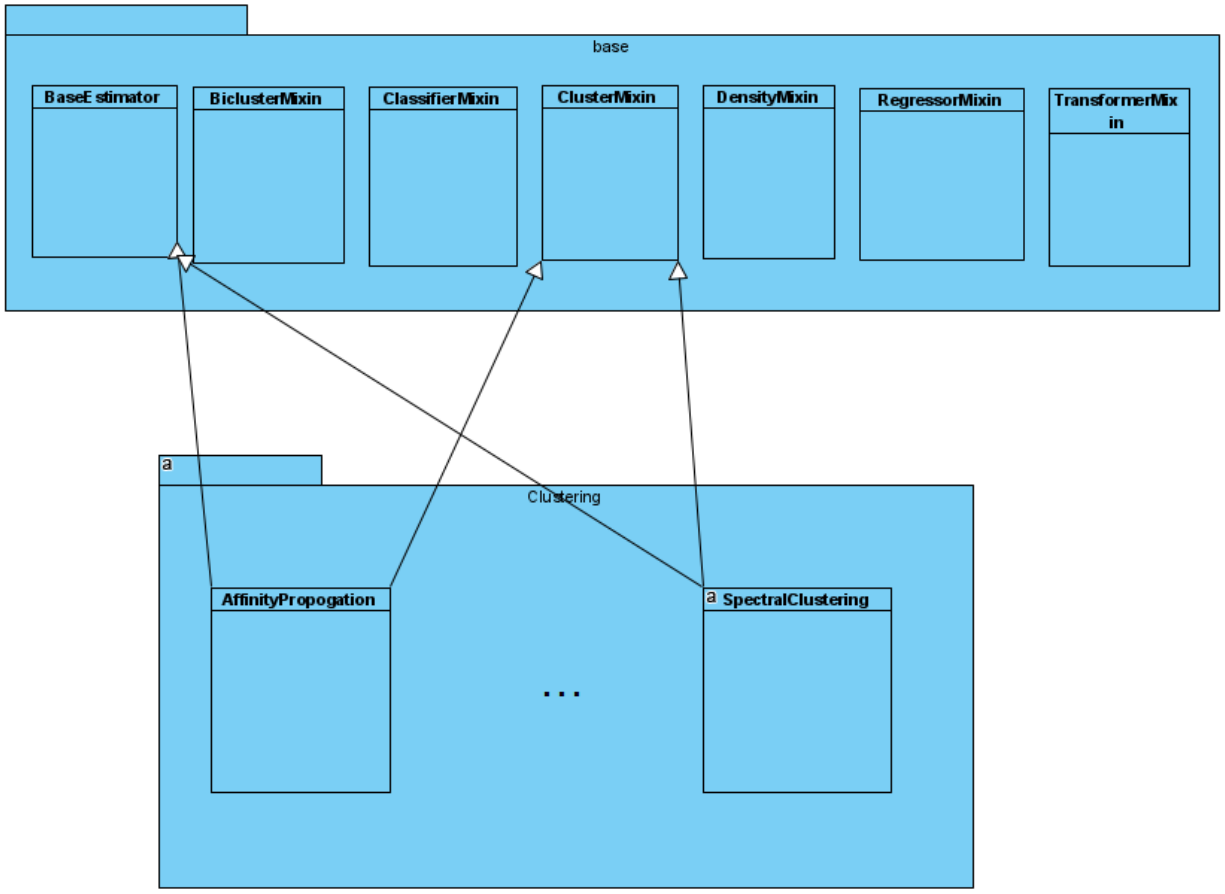
Design Patterns:

Module Design Pattern:



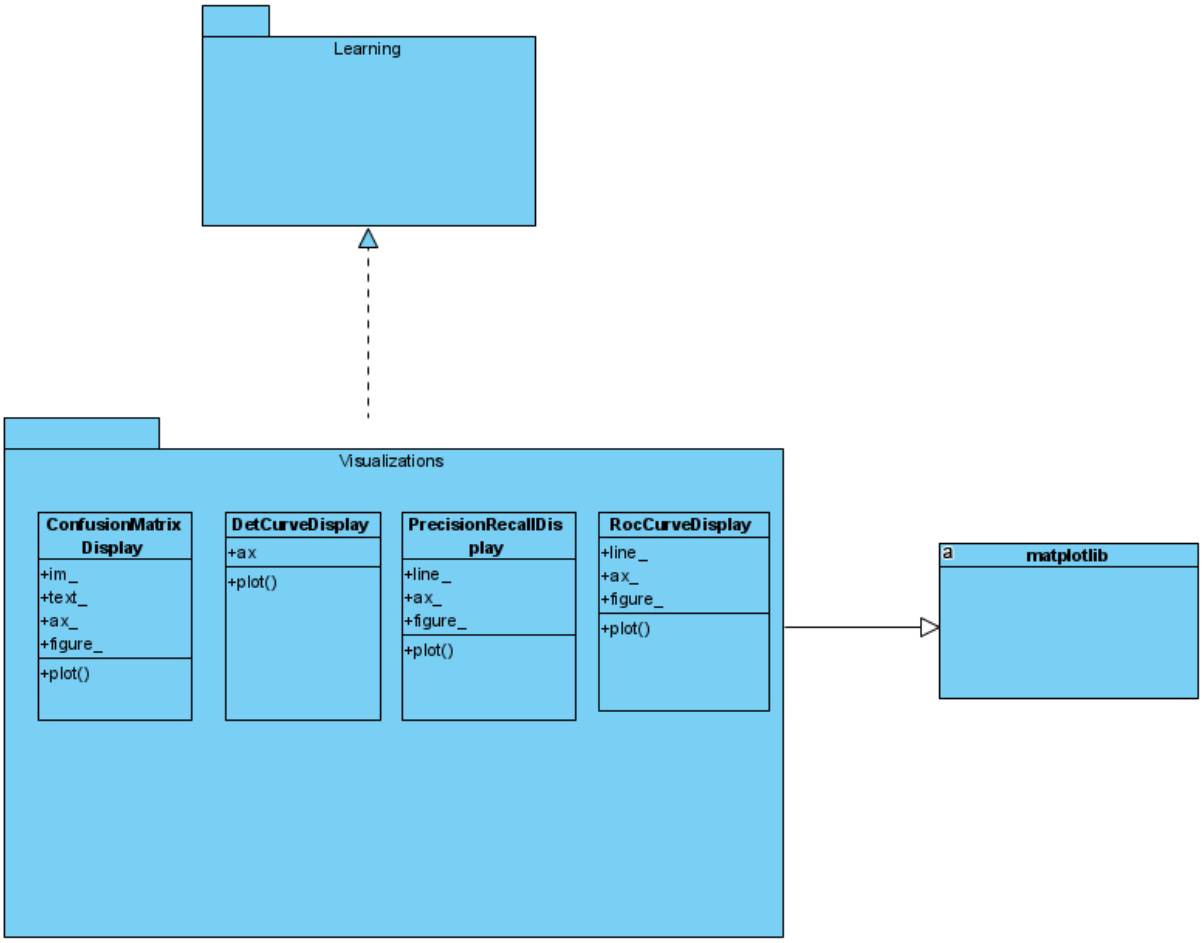
Picking up from where we left off when looking at the provided scikit-learn algorithm cheat-sheet, we notice that the Python modules are arranged in a conceptual manner. For example, while there may be many different types of clustering, all clustering algorithms are related conceptually, and thus fall under the clustering module. This is also the case for other types of learning algorithms, supervised or unsupervised. Such design fulfills the Module Design Pattern.

Decorator Design Pattern:



The scikit-learn library defines a base module, which offers access to a basic estimator, BaseEstimator, as well as a variety of Mixin classes. All of the learning algorithms, such as clustering algorithms, are an extension of such estimators and mixins. However, at the same time, these algorithms are independent of each other. This satisfies the requirement of the Decorator Design Pattern, as we can add additional functionality/behaviour to estimators without affecting any other estimator.

Adapter Design Pattern:

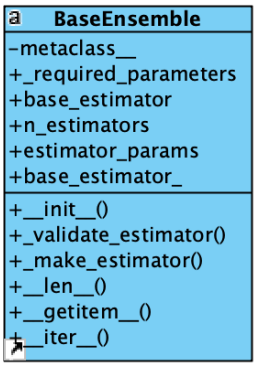


The scikit-learn library offers an API for creating graphs for learning algorithms. Under the hood, the plotting is done by matplotlib, and so it is natural that there must be classes that help the library communicate with matplotlib. This describes the Adapter Design Pattern, which can be seen when inspecting the classes inside the Visualizations module. Each of the classes take in as parameters, some form of attributes used to be used for matplotlib. This allows the classes of the learning algorithms to work with the matplotlib interface.

Mock Object Design Pattern

The scikit-learn library uses mock objects in their testing. This allows them to create mock instances of classes with controlled behavior. For example, in test_weight_boosting.py a mockEstimator is created. The mockEstimator has one function that returns a controlled probability, allowing for easy testing of _samma_prob.

Iterator Design Pattern



The scikit-learn library offers users the ability to traverse an object and access each of its elements. This is seen through the implementation of the python __iter__ method in the BaseEnsemble class. This allows users to iterate through the list of estimators in the object. This example is located in the _base.py file of the ensemble class.

Quality of Architecture

The design of the scikit-learn architecture is very intuitive, partly due to the Modular Design Pattern. While typically Python's '.py' files are modular by nature, the modular aspect we refer to is actually the way the library's API is unified (all relevant classes and functions are conceptually organized under the appropriate modules). Additionally, the library uses base classes as if they were some sort of interface, allowing the built-in algorithms to extend upon the base classes, adding their own implementations within a higher level class. Users are able to easily identify the type of object they have on hand (estimator, predictor, transformer, etc.), which adds to the library's ease of use.

Such use of base classes also increases the flexibility of the library. If a user wishes to use a custom learning algorithm, they can do it by following the structure of the base classes (all of which are open to the users). This flexibility is also seen from several modules offered, such as the multiclass module; the library allows users to "experiment with custom multiclass strategies" by creating their own meta-estimators.

A critique of the scikit-learn architecture would be the lack of consistency in regards to the coding guidelines stated under the Pull request checklist section of contributing. Scikit-learn is supposed to follow the PEP 8 coding format (with few additional rules). PEP 8 states that imports should always be at the top of the python file, however, many classes have import statements positioned within the code. Some examples of this include:

- impute/_iterative.py line 597 https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/impute/_iterative.py#L597
- cluster/_spectral.py line 76 and 77 https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/cluster/_spectral.py#L76
- datasets/_lfw.py line 109 and 120 https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/datasets/_lfw.py#L109

A comment in _lfw.py line 120 indicates that a reason for import statements to not be at the top of the file is that scikit only wants the library to be included when it is needed. A recommendation would be to change the import statements to follow the PEP 8 coding format or if import statements within the code are necessary, include this as part of their additional rules list.

SOURCE: https://scikit-learn.org/stable/developers/develop.html#coding-guidelines