CSCD01: Deliverable 2

Task 1: Auto-Generated UML

Our pyreverse-generated UML diagram can be found on our repository under this file.

Task 2: High-Level System Design

Scikit-learn is a tool that allows developers to easily accomplish numerous ML and ML-adjacent tasks. These include algorithms and implementations for clustering, regression, transformation, etc.

At its core, scikit-learn works around the concept of an estimator, implemented in **BaseEstimator**. On top of this, it tackles a number of high-level concepts such as transformation, regression, etc. For each of these high-level concepts, scikit-learn has a Mixin; a class whose relationship with concrete classes allows for said classes to inherit some shared method definition without having to be direct children.

Users can interact with scikit-learn by using **predictors** (which are estimators); predictors are used for supervised learning approaches like classification and regression, as well as unsupervised learning like clustering.
Predictors implement *predict* Which is used for creating an output given the input and a *score* method which evaluates the quality of a model's prediction.

Other important estimators are transformers which are applied to the input data to pre-process, modify and filter before being fed to the predictors.

Multiple estimators can be composed together as is often needed in ML.
The two most important modules with this functionality are ***pipelines*** and ***model selectors***.
Pipelines and Model_selectors all inherit from *Meta_estimator* class which is an estimator that allows for other estimators as parameters;

Pipelines allow the input to go through multiple transformations before being applied to a final estimator allowing the multiple fit, transform and predict calls all being done in the same method.

Model selector allows for training the estimator multiple times with different hyperparameters (which are parameters passed to the estimator before training starts), cross validation and for creating validation and learning curves.

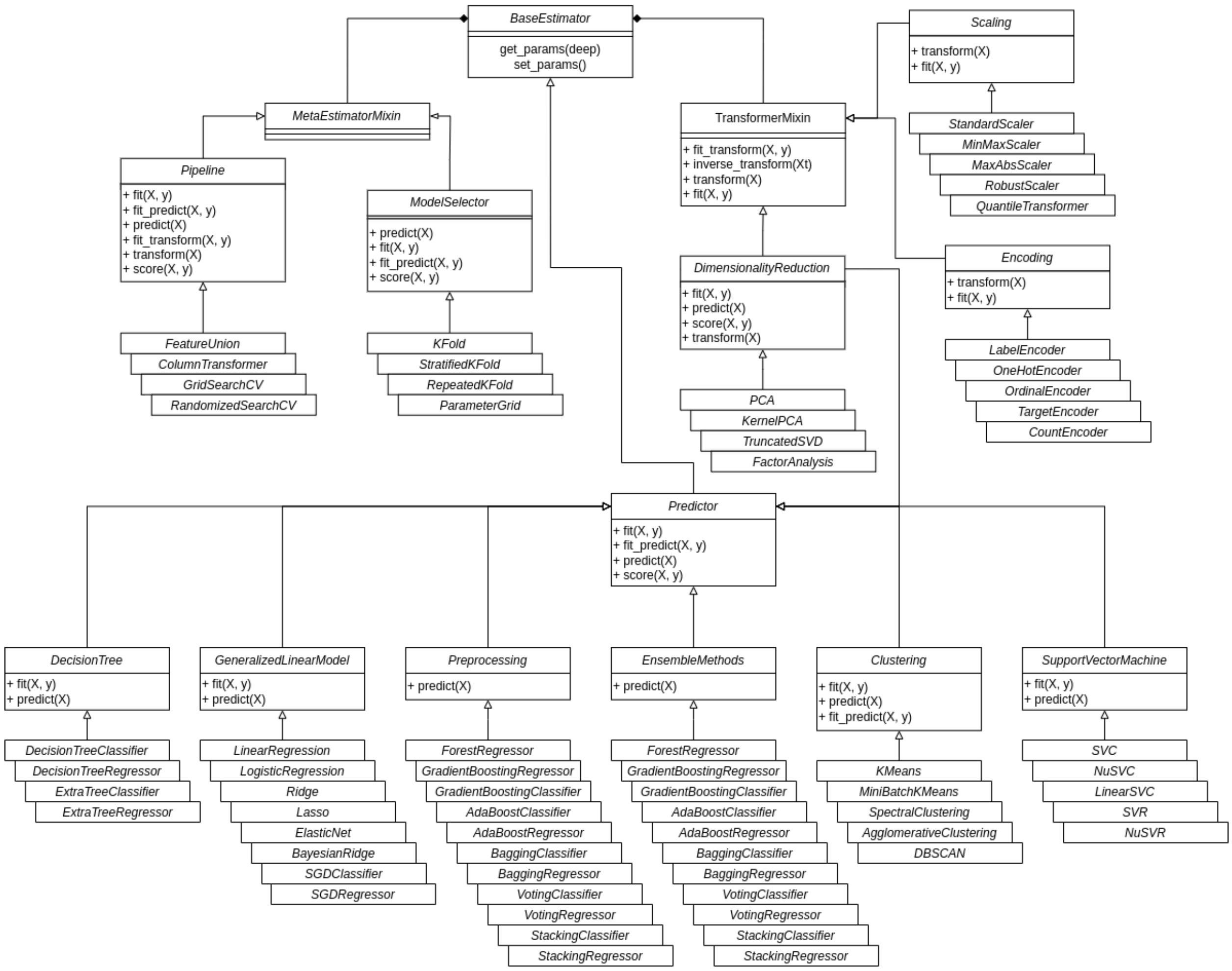
Outside of direct ML tasks such as classification and regression, scikit-learn offers utilities for tasks that are often encountered when working with ML workloads. Some examples of these utilities are as follow:

Scikit-learn offers a **datasets** module with tools for loading, fetching, and verifying popular ground-truth datasets. This simplifies the process of securing sample data with which models can be trained, tested and evaluated. This module also offers utilities to generate data according to some desired distribution. While there are two main subsets of this module focused on the loading and generating sides, the module is fine as it is; still being largely cohesive in that it focuses on setting up the data for later use.

Further utilities offered by scikit-learn include those for **preprocessing**; allowing for scaling, normalizing, and otherwise manipulating datasets prior to use for training or testing. This allows for standardization of a dataset prior to use. The preprocessing module is highly cohesive and serves a clear purpose. There's little that we would change in this module.

Finally, scikit-learn offers tools for tuning hyperparameters in a model through the **model_selection** module. This includes tools for different kinds of hyperparameter search, such as grid search, random search, etc. This module could afford to be more cohesive, as it currently covers multiple problems such as model validation, splitter classes and hyperparameter optimization. For example, rather than `sklearn.model_selection.kfold`, it may be better to compartmentalize this functionality into multiple intermediate classes such as: `sklearn.model_selection.splitter_classes.kfold`.

https://drive.google.com/file/d/1MctyQB1D1SvV9xWHAdf9z9DhbHw6Rxq-/view?usp=sharing



Task 3: Two Design Patterns

https://desosa.nl/projects/scikit-learn/2020/03/06/scikit-learn-what-does-it-want-to-be.html

Design Pattern 1 🖥️

Factory Design Pattern 🏭

Ⅰ. *Description of the Design Pattern* 📜*:*

The Factory Design Pattern is a creational design pattern that offers a solution for object creation by providing an interface for the user to instantiate and return different classes based on parameters passed to the factory function. The pattern is designed around a factory function that is responsible for creating and returning objects based on the parameters it receives. This function is generally implemented as a standalone function that is not bound to any particular class, although it can be included in a specific class for organizational purposes.

One of the primary benefits of the Factory Design Pattern is that it decouples the object creation process from the rest of the code. This means that new objects related to classes that may be added in the future can be created without compromising the codebase. The pattern provides a flexible and scalable approach to object creation, allowing for changes in the system to be made with minimal impact to existing code.

In summary, the Factory Design Pattern is a powerful solution for object creation that provides an interface for users to create and return different classes based on parameters passed to a factory function. By decoupling the object creation process from the rest of the code, this pattern promotes scalability and flexibility in software design.

The Factory Design Pattern is an essential tool employed within the codebase of the Scikit-learn project. Our exploration of this codebase will allow us to delve deeper into the practical application of the pattern, and elucidate its inner workings using UML diagrams and detailed descriptions of relevant code files. Drawing upon the definition of the Factory Design Pattern presented earlier, we will provide a clear and comprehensive understanding of its utilization within the Scikit-learn project.

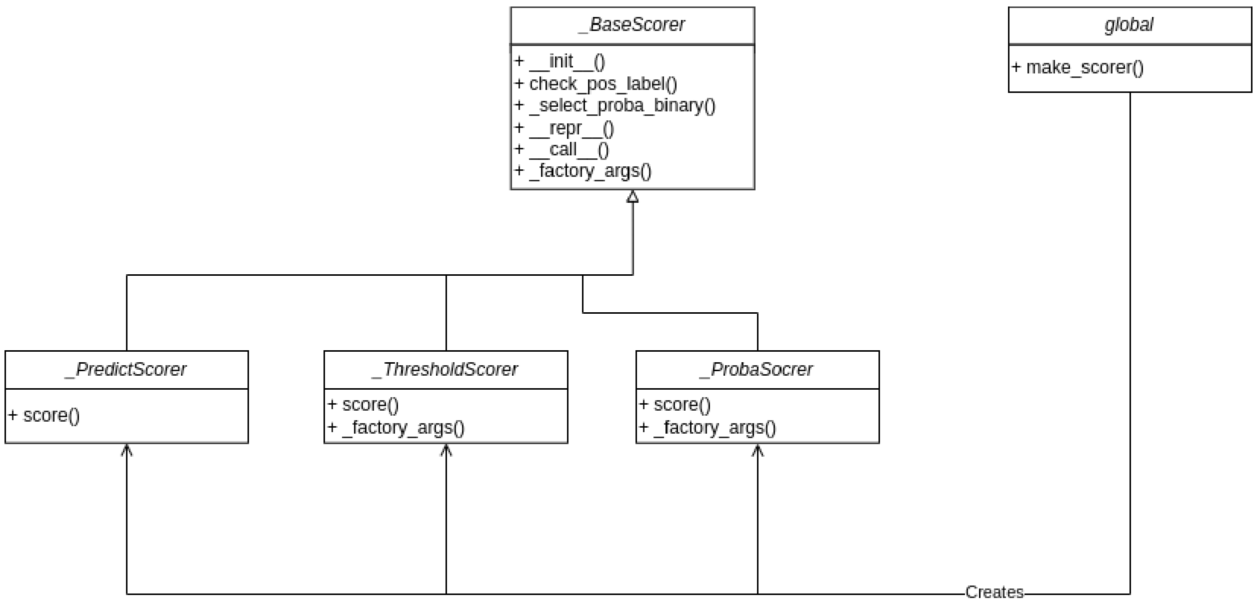
Ⅱ. *How the Design Pattern is Used ⁉️:*

The Scikit-Learn project implements the Factory Design Pattern in the ***make_scorer()*** method. This method serves as a factory function and is responsible for creating and returning objects of three distinct classes: ***_ProbaScorer***, ***_ThresholdScorer***, and ***_PredictScorer***.

The ***make_scorer()*** method conforms to the essential features of the factory design pattern by providing users with a simplified interface to instantiate and return objects of different classes based on the parameters passed to the factory function. The creation of objects is performed internally and is concealed from the user, thus maintaining a clear separation between the object creation process and the overall application logic.

We can confirm that the ***make_scorer()*** method is an excellent example of the factory design pattern in practice. It is worth noting that subclasses of the method can override the implementation to create and return objects that are relevant to their specific use cases. This approach promotes flexibility and modularity within the codebase, making it more adaptable to future changes or updates.

Ⅲ. *The UML Diagram and Its Explanation 📚:*



The UML diagram for the factory design pattern in the Scikit-Learn project contains four key components, along with the factory function. The parent class in this design pattern is called _BaseScorer, which has three subclasses that inherit from it: ***_ProbaScorer***, ***_ThresholdScorer***, and ***_PredictScorer***.

The ***make_scorer*** function serves as a global component in the UML diagram, responsible for creating instances of the three subclasses based on the parameters that are passed to it. This aligns with the behavior of the factory method pattern, which provides an interface for creating objects of different classes based on specific parameters, while hiding the creation process from the user.

One of the key benefits of using the factory method pattern in this context is that it helps to decouple the object creation process from the rest of the code, which means that new classes or components can be added in the future without significant changes to the existing codebase. This is illustrated in the UML diagram as the factory method make_scorer is presented as a global and independent component.

The decoupling factor of the factory design pattern is demonstrated in this UML diagram, as it allows for the addition of new components and classes in the future with minimal changes to the existing codebase.

Overall, the UML diagram for the factory design pattern in the Scikit-Learn project illustrates how the make_scorer function acts as a global component to create and return instances of the ***_ProbaScorer***, ***_ThresholdScorer***, and ***_PredictScorer*** subclasses based on specific parameters, while also helping to maintain a flexible and modular codebase.

Ⅳ. *Where the Design Pattern is Implemented 🗺️:*

scikit-learn/sklearn/metrics/_scorer.py :: Lines 604-700

Ⅴ. *Example Usage in the Project 💡:*

scikit-learn/sklearn/linear_model/tests/test_ridge.py :: Lines 955