📚 **Tutorial: baseline RS methods in `BaseEstimator` type**

Absolutely! Developing a **custom scikit-learn estimator** is a great way to extend the functionality of scikit-learn for your own models or preprocessing steps. Here's a **step-by-step tutorial** designed for students in *STAT3009*:

🛠️ How to Develop a Custom `sklearn` Estimator

Understand the Estimator API

All scikit-learn estimators (including classifiers, regressors, transformers) follow a **consistent API**:

- **Initialization**: All parameters are set in `__init__`.
- **Fitting**: The `fit(X, y)` method learns from data.
- **Prediction/Transformation**: The `predict(X)` or `transform(X)` method applies the learned model.

Import Required Base Classes

Depending on your estimator type, you'll inherit from:

- `BaseEstimator` (for all custom estimators)
- `TransformerMixin` (for transformers)
- `ClassifierMixin` or `RegressorMixin` (for classifiers / regressors)

```
from sklearn.base import BaseEstimator, TransformerMixin
```

1. BaseEstimator

- **Purpose:**
  Provides the basic structure for all scikit-learn estimators.
- **Key Features:**
  - Implements `get_params()` and `set_params()` automatically (crucial for grid search and pipelines).
  - Ensures all constructor arguments are stored as attributes.

- Enforces scikit-learn conventions for estimator design.

2. RegressorMixin

- **Purpose:**
  Adds a default implementation of the `score()` method (R^2 score), which is standard for regressors.
- **Key Features:**
  - Supplies the `score(X, y)` method so you don't have to implement it yourself.
  - Ensures your estimator can be used with scikit-learn's model selection and evaluation tools.

Step-by-Step Conversion

1. **Inherit from `BaseEstimator`**:
   - We will inherit from `BaseEstimator` to get basic functionality like hyperparameter setting and getting.
2. **Define the `fit` and `predict` methods**:
   - The `fit` method will compute the global mean of the training ratings.
   - The `predict` method will return the global mean for each test pair.

Here's the example code for global mean RS methods:

```
import numpy as np
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.utils.validation import check_is_fitted

class GlobalMeanRS(BaseEstimator, RegressorMixin):
    def __init__(self):
        # model parameters
        self.glb_mean_ = 0

    def fit(self, X, y):
        # fit parameter
        self.glb_mean_ = np.mean(y)

    def predict(self, X):
        # Ensure the estimator is fitted
        check_is_fitted(self, 'glb_mean_')
        # Return the global mean for each test pair
        y_pred = np.ones(len(X))
        return y_pred*self.glb_mean_

    def score(self, X, y):
        # Custom score: negative mean absolute error
        y_pred = self.predict(X)
        mae = np.mean(np.abs(y - y_pred))
```

```
        return -mae  # Negative because higher score is better in sklearn
```

Explanation

1. **Initialization (`__init__` method)**:
    - Initialize `glb_mean_` to 0. The trailing underscore indicates that this is an attribute set during the `fit` method.
2. **Fitting the Model (`fit` method)**:
    - Compute the global mean of the training ratings and store it in `glb_mean_`.
3. **Making Predictions (`predict` method)**:
    - Ensure the estimator has been fitted using `check_is_fitted`.
    - Return the global mean for each test pair.

Usage

The example usage demonstrates how to use this custom estimator with Scikit-learn's familiar `fit`, `predict` methods. This allows you to leverage Scikit-learn's powerful tools for model selection, evaluation, and preprocessing while using your tailored algorithms.

By following this approach, you can create custom machine learning estimators that integrate seamlessly with Scikit-learn's ecosystem.

⭐ *Checklist for Custom Estimators*

- √ Inherit from `BaseEstimator` and `RegressorMixin`.
- √ All parameters in `__init__`.
- √ No logic in `__init__` except assigning parameters.
- √ `fit` must return `self`.
- √ Use `set_params` and `get_params` for hyperparameter tuning (inherited from `BaseEstimator`).
- √ Test thoroughly!

Application: Use in Pipelines

Your custom estimator can be used just like any scikit-learn estimator:

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ('imputer', MeanImputer()),
    # Add other steps...
])
```