Scikit-learn has a lot of nice helper functions for making estimators in a way that sklearn expects. The problem is that some of these helper functions are relatively new, and may not be included in your installation.

The good news is that these are "nice-to-have", and not "must-have"...

See also:

- https://scikit-learn.org/stable/glossary.html#class-apis-and-estimator-types
    - Which methods are required for, for example, a Regressor, and what should those methods do?
- https://scikit-learn.org/stable/developers/develop.html
    - For the nitty-gritty details

```
from sklearn.base import RegressorMixin, ClassifierMixin, 
BaseEstimator
# Here's a minimal DummyRegressor
# First the mixin, then the base estimator
class DummyRegressor(RegressorMixin, BaseEstimator):
    def fit(self, X, y):
        # Validate data is nice to have to validate the shapes of X 
and y (and also in predict, to make sure)
        # that the number of features doesn't change in the meantime.
        # The nicest thing it does however, is make sure X and y are 
(turned into) numpy arrays. For example,
        # it converts pandas dataframes or scipy sparse matrices into 
numpy arrays. If you don't have access
        # to validate_data, you need to make sure you convert your 
data to np arrays yourself; for example
        # after splitting.
        # See also sklearn.utils.check_X_y and check_array
        # X, y = validate_data(self, X, y)
        self.mean_ = np.mean(y)
        return self

    def predict(self, X):
        # X = validate_data(self, X, reset=False)  # See fit
        # check_is_fitted does what it says on the tin. It makes sure 
this estimator has been fit before
        # you try to use it, and raises an error otherwise.
        # check_is_fitted(self)
        return np.full(X.shape[0], self.mean_)

from sklearn.utils.multiclass import unique_labels
class DummyClassifier(ClassifierMixin, BaseEstimator):

    def fit(self, X, y):
        # X, y = validate_data(self, X, y)  # See regressor
        # Besides sanity checking, unique_labels returns a sorted 
array of unique classes.
```

```
        # You should be able to use the following as a drop-in
replacement:
        # self.classes_, y = np.unique(y, return_inverse=True)
        self.classes_ = unique_labels(y)
        onehot = y[:, np.newaxis] == self.classes_[np.newaxis]
        self.likelihoods_ = np.mean(onehot, axis=0)
        return self

    def predict_proba(self, X):
        # X = validate_data(self, X, reset=False)  # See regressor
        # check_is_fitted(self)  # See regressor
        return np.full((X.shape[0], self.classes_.shape[0]),
self.likelihoods_[np.newaxis, :])

    def predict(self, X):
        probas = self.predict_proba(X)
        idxs = np.argmax(probas, axis=1)
        return self.classes_[idxs]
```