Developing scikit-learn estimators

Whether you are proposing an estimator for inclusion in scikit-learn, developing a separate package compatible with scikit-learn, or implementing custom components for your own projects, this chapter details how to develop objects that safely interact with scikit-learn Pipelines and model selection tools.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 13)

Unknown directive type "currentmodule".

```
.. currentmodule:: sklearn
```

APIs of scikit-learn objects

To have a uniform API, we try to have a common basic API for all the objects. In addition, to avoid the proliferation of framework code, we try to adopt simple conventions and limit to a minimum the number of methods an object must implement.

Elements of the scikit-learn API are described more definitively in the :ref:`glossary`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 25); *backlink*

Unknown interpreted text role "ref".

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

Transformer: For filtering or modifying the data, in a supervised or unsupervised way, implements:

```
new_data = transformer.transform(data)
```

When fitting and transforming can be performed much more efficiently together than separately, implements:

```
new_data = transformer.fit_transform(data)
```

Model: A model that can give a goodness of fit measure or a likelihood of unseen data, implements (higher is better):

```
score = model.score(data)
```

Estimators

The API has one predominant object: the estimator. An estimator is an object that fits a model based on some training data and is capable of inferring some properties on new data. It can be, for instance, a classifier or a regressor. All estimators implement the fit method:

```
estimator.fit(X, y)
```

All built-in estimators also have a `set_params` method, which sets data-independent parameters (overriding previous parameter values passed to `__init__`).

All estimators in the main scikit-learn codebase should inherit from `sklearn.base.BaseEstimator`.

Instantiation

This concerns the creation of an object. The object's `__init__` method might accept constants as arguments that determine the estimator's behavior (like the C constant in SVMs). It should not, however, take the actual training data as an argument, as this is left to the `fit()` method:

```
clf2 = SVC(C=2.3)
clf3 = SVC([[1, 2], [2, 3]], [-1, 1]) # WRONG!
```

The arguments accepted by `__init__` should all be keyword arguments with a default value. In other words, a user should be able to instantiate an estimator without passing any arguments to it. The arguments should all correspond to hyperparameters describing the model or the optimisation problem the estimator tries to solve. These initial arguments (or parameters) are always remembered by the estimator. Also note that they should not be documented under the "Attributes" section, but rather under the "Parameters" section for that estimator.

In addition, **every keyword argument accepted by `__init__` should correspond to an attribute on the instance**. Scikit-learn relies on this to find the relevant attributes to set on an estimator when doing model selection.

To summarize, an `__init__` should look like:

```
def __init__(self, param1=1, param2=2):
    self.param1 = param1
    self.param2 = param2
```

There should be no logic, not even input validation, and the parameters should not be changed. The corresponding logic should be put where the parameters are used, typically in `fit`. The following is wrong:

```
def __init__(self, param1=1, param2=2, param3=3):
    # WRONG: parameters should not be modified
    if param1 > 1:
        param2 += 1
    self.param1 = param1
    # WRONG: the object's attributes should have exactly the name of
    # the argument in the constructor
    self.param3 = param2
```

The reason for postponing the validation is that the same validation would have to be performed in `set_params`, which is used in algorithms like `GridSearchCV`.

Fitting

The next thing you will probably want to do is to estimate some parameters in the model. This is implemented in the `fit()` method.

The `fit()` method takes the training data as arguments, which can be one array in the case of unsupervised learning, or two arrays in the case of supervised learning.

Note that the model is fitted using `X` and `y`, but the object holds no reference to `X` and `y`. There are, however, some exceptions to this, as in the case of precomputed kernels where this data must be stored for use by the predict method.

Parameters	
X	array-like of shape (n_samples, n_features)
y	array-like of shape (n_samples,)
kwargs	optional data-dependent parameters

`X.shape[0]` should be the same as `y.shape[0]`. If this requisite is not met, an exception of type `ValueError` should be raised.

`y` might be ignored in the case of unsupervised learning. However, to make it possible to use the estimator as part of a pipeline that can mix both supervised and unsupervised transformers, even unsupervised estimators need to accept a `y=None` keyword argument in the second position that is just ignored by the estimator. For the same reason, `fit_predict`, `fit_transform`, `score` and `partial_fit` methods need to accept a `y` argument in the second place if they are implemented.

The method should return the object (`self`). This pattern is useful to be able to implement quick one liners in an IPython session such as:

```
y_predicted = SVC(C=100).fit(X_train, y_train).predict(X_test)
```

Depending on the nature of the algorithm, `fit` can sometimes also accept additional keywords arguments. However, any parameter that can have a value assigned prior to having access to the data should be an `__init__` keyword argument. **fit parameters should be restricted to directly data dependent variables**. For instance a Gram matrix or an affinity matrix which are precomputed from the data matrix `X` are data dependent. A tolerance stopping criterion `tol` is not directly data dependent (although the optimal value according to some scoring function probably is).

When `fit` is called, any previous call to `fit` should be ignored. In general, calling `estimator.fit(X1)` and then `estimator.fit(X2)` should be the same as only calling `estimator.fit(X2)`. However, this may not be true in practice when `fit` depends on some random process, see :term:`random_state`. Another exception to this rule is when the hyper-parameter

`warm_start` is set to `True` for estimators that support it. `warm_start=True` means that the previous state of the trainable parameters of the estimator are reused instead of using the default initialization strategy.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 193); *backlink***
>
> Unknown interpreted text role "term".

Estimated Attributes

Attributes that have been estimated from the data must always have a name ending with trailing underscore, for example the coefficients of some regression estimator would be stored in a `coef_` attribute after `fit` has been called.

The estimated attributes are expected to be overridden when you call `fit` a second time.

Optional Arguments

In iterative algorithms, the number of iterations should be specified by an integer called `n_iter`.

Universal attributes

Estimators that expect tabular input should set a *n_features_in_* attribute at *fit* time to indicate the number of features that the estimator expects for subsequent calls to *predict* or *transform*. See SLEP010 for details.

Rolling your own estimator

If you want to implement a new estimator that is scikit-learn-compatible, whether it is just for you or for contributing it to scikit-learn, there are several internals of scikit-learn that you should be aware of in addition to the scikit-learn API outlined above. You can check whether your estimator adheres to the scikit-learn interface and standards by running :func:`~sklearn.utils.estimator_checks.check_estimator` on an instance. The :func:`~sklearn.utils.estimator_checks.parametrize_with_checks` pytest decorator can also be used (see its docstring for details and possible interactions with *pytest*):

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 235); *backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 235); *backlink***
>
> Unknown interpreted text role "func".

```
>>> from sklearn.utils.estimator_checks import check_estimator
>>> from sklearn.svm import LinearSVC
>>> check_estimator(LinearSVC())  # passes
```

The main motivation to make a class compatible to the scikit-learn estimator interface might be that you want to use it together with model evaluation and selection tools such as :class:`model_selection.GridSearchCV` and :class:`pipeline.Pipeline`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 249); *backlink***
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 249); *backlink***
>
> Unknown interpreted text role "class".

Before detailing the required interface below, we describe two ways to achieve the correct interface more easily.

Project template:

We provide a project template which helps in the creation of Python packages containing scikit-learn compatible estimators. It provides:

- an initial git repository with Python package directory structure
- a template of a scikit-learn estimator
- an initial test suite including use of `check_estimator`
- directory structures and scripts to compile documentation and example galleries
- scripts to manage continuous integration (testing on Linux and Windows)
- instructions from getting started to publishing on PyPi

`BaseEstimator` and mixins:

We tend to use "duck typing", so building an estimator which follows the API suffices for compatibility, without needing to inherit from or even import any scikit-learn classes.

However, if a dependency on scikit-learn is acceptable in your code, you can prevent a lot of boilerplate code by deriving a class from `BaseEstimator` and optionally the mixin classes in `sklearn.base`. For example, below is a custom classifier, with more examples included in the scikit-learn-contrib project template.

```
>>> import numpy as np
>>> from sklearn.base import BaseEstimator, ClassifierMixin
>>> from sklearn.utils.validation import check_X_y, check_array, check_is_fitted
>>> from sklearn.utils.multiclass import unique_labels
>>> from sklearn.metrics import euclidean_distances
>>> class TemplateClassifier(BaseEstimator, ClassifierMixin):
...
...     def __init__(self, demo_param='demo'):
...         self.demo_param = demo_param
...
...     def fit(self, X, y):
...
...         # Check that X and y have correct shape
...         X, y = check_X_y(X, y)
...         # Store the classes seen during fit
...         self.classes_ = unique_labels(y)
...
...         self.X_ = X
...         self.y_ = y
...         # Return the classifier
...         return self
...
...     def predict(self, X):
...
...         # Check if fit has been called
...         check_is_fitted(self)
...
...         # Input validation
...         X = check_array(X)
...
...         closest = np.argmin(euclidean_distances(X, self.X_), axis=1)
...         return self.y_[closest]
```

get_params and set_params

All scikit-learn estimators have `get_params` and `set_params` functions. The `get_params` function takes no arguments and returns a dict of the `__init__` parameters of the estimator, together with their values.

It must take one keyword argument, `deep`, which receives a boolean value that determines whether the method should return the parameters of sub-estimators (for most estimators, this can be ignored). The default value for `deep` should be *True*. For instance considering the following estimator:

```
>>> from sklearn.base import BaseEstimator
>>> from sklearn.linear_model import LogisticRegression
>>> class MyEstimator(BaseEstimator):
...     def __init__(self, subestimator=None, my_extra_param="random"):
...         self.subestimator = subestimator
...         self.my_extra_param = my_extra_param
```

The parameter *deep* will control whether or not the parameters of the *subsestimator* should be reported. Thus when *deep=True*, the output will be:

```
>>> my_estimator = MyEstimator(subestimator=LogisticRegression())
>>> for param, value in my_estimator.get_params(deep=True).items():
...     print(f"{param} -> {value}")
my_extra_param -> random
subestimator__C -> 1.0
subestimator__class_weight -> None
subestimator__dual -> False
subestimator__fit_intercept -> True
```

```
subestimator__intercept_scaling -> 1
subestimator__l1_ratio -> None
subestimator__max_iter -> 100
subestimator__multi_class -> auto
subestimator__n_jobs -> None
subestimator__penalty -> l2
subestimator__random_state -> None
subestimator__solver -> lbfgs
subestimator__tol -> 0.0001
subestimator__verbose -> 0
subestimator__warm_start -> False
subestimator -> LogisticRegression()
```

Often, the *subestimator* has a name (as e.g. named steps in a :class:`~sklearn.pipeline.Pipeline` object), in which case the key should become <*name*>__*C*, <*name*>__*class_weight*, etc.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 362); *backlink***
>
> Unknown interpreted text role "class".

While when *deep=False*, the output will be:

```
>>> for param, value in my_estimator.get_params(deep=False).items():
...     print(f"{param} -> {value}")
my_extra_param -> random
subestimator -> LogisticRegression()
```

The `set_params` on the other hand takes as input a dict of the form `'parameter': value` and sets the parameter of the estimator using this dict. Return value must be estimator itself.

While the `get_params` mechanism is not essential (see :ref:`cloning` below), the `set_params` function is necessary as it is used to set parameters during grid searches.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 377); *backlink***
>
> Unknown interpreted text role "ref".

The easiest way to implement these functions, and to get a sensible `__repr__` method, is to inherit from `sklearn.base.BaseEstimator`. If you do not want to make your code dependent on scikit-learn, the easiest way to implement the interface is:

```
def get_params(self, deep=True):
    # suppose this estimator has parameters "alpha" and "recursive"
    return {"alpha": self.alpha, "recursive": self.recursive}

def set_params(self, **parameters):
    for parameter, value in parameters.items():
        setattr(self, parameter, value)
    return self
```

Parameters and init

As :class:`model_selection.GridSearchCV` uses `set_params` to apply parameter setting to estimators, it is essential that calling `set_params` has the same effect as setting parameters using the `__init__` method. The easiest and recommended way to accomplish this is to **not do any parameter validation in** `__init__`. All logic behind estimator parameters, like translating string arguments into functions, should be done in `fit`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 398); *backlink***
>
> Unknown interpreted text role "class".

Also it is expected that parameters with trailing _ are **not to be set inside the** `__init__` **method**. All and only the public attributes set by fit have a trailing _. As a result the existence of parameters with trailing _ is used to check if the estimator has been fitted.

Cloning

For use with the :mod:`model_selection` module, an estimator must support the `base.clone` function to replicate an estimator. This

can be done by providing a `get_params` method. If `get_params` is present, then `clone(estimator)` will be an instance of `type(estimator)` on which `set_params` has been called with clones of the result of `estimator.get_params()`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 416);** ***backlink***
>
> Unknown interpreted text role "mod".

Objects that do not provide this method will be deep-copied (using the Python standard function `copy.deepcopy`) if `safe=False` is passed to `clone`.

Pipeline compatibility

For an estimator to be usable together with `pipeline.Pipeline` in any but the last step, it needs to provide a `fit` or `fit_transform` function. To be able to evaluate the pipeline on any data but the training set, it also needs to provide a `transform` function. There are no special requirements for the last step in a pipeline, except that it has a `fit` function. All `fit` and `fit_transform` functions must take arguments `X, y`, even if y is not used. Similarly, for `score` to be usable, the last step of the pipeline needs to have a `score` function that accepts an optional `y`.

Estimator types

Some common functionality depends on the kind of estimator passed. For example, cross-validation in :class:`model_selection.GridSearchCV` and :func:`model_selection.cross_val_score` defaults to being stratified when used on a classifier, but not otherwise. Similarly, scorers for average precision that take a continuous prediction need to call `decision_function` for classifiers, but `predict` for regressors. This distinction between classifiers and regressors is implemented using the `_estimator_type` attribute, which takes a string value. It should be `"classifier"` for classifiers and `"regressor"` for regressors and `"clusterer"` for clustering methods, to work as expected. Inheriting from `ClassifierMixin`, `RegressorMixin` or `ClusterMixin` will set the attribute automatically. When a meta-estimator needs to distinguish among estimator types, instead of checking `_estimator_type` directly, helpers like :func:`base.is_classifier` should be used.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 441);** ***backlink***
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 441);** ***backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 441);** ***backlink***
>
> Unknown interpreted text role "func".

Specific models

Classifiers should accept `y` (target) arguments to `fit` that are sequences (lists, arrays) of either strings or integers. They should not assume that the class labels are a contiguous range of integers; instead, they should store a list of classes in a `classes_` attribute or property. The order of class labels in this attribute should match the order in which `predict_proba`, `predict_log_proba` and `decision_function` return their values. The easiest way to achieve this is to put:

```
self.classes_, y = np.unique(y, return_inverse=True)
```

in `fit`. This returns a new `y` that contains class indexes, rather than labels, in the range [0, `n_classes`).

A classifier's `predict` method should return arrays containing class labels from `classes_`. In a classifier that implements `decision_function`, this can be achieved with:

```
def predict(self, X):
    D = self.decision_function(X)
    return self.classes_[np.argmax(D, axis=1)]
```

In linear models, coefficients are stored in an array called `coef_`, and the independent term is stored in `intercept_`. `sklearn.linear_model._base` contains a few base classes and mixins that implement common linear model patterns.

The :mod:`sklearn.utils.multiclass` module contains useful functions for working with multiclass and multilabel problems.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 485);** ***backlink***
>
> Unknown interpreted text role "mod".

Estimator Tags

> **Warning**
>
> The estimator tags are experimental and the API is subject to change.

Scikit-learn introduced estimator tags in version 0.21. These are annotations of estimators that allow programmatic inspection of their capabilities, such as sparse matrix support, supported output types and supported methods. The estimator tags are a dictionary returned by the method `_get_tags()`. These tags are used in the common checks run by the :func:`~sklearn.utils.estimator_checks.check_estimator` function and the :func:`~sklearn.utils.estimator_checks.parametrize_with_checks` decorator. Tags determine which checks to run and what input data is appropriate. Tags can depend on estimator parameters or even system architecture and can in general only be determined at runtime.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 496);** ***backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 496);** ***backlink***
>
> Unknown interpreted text role "func".

The current set of estimator tags are:

allow_nan (default=False)

: whether the estimator supports data with missing values encoded as np.NaN

binary_only (default=False)

: whether estimator supports binary classification but lacks multi-class classification support.

multilabel (default=False)

: whether the estimator supports multilabel output

multioutput (default=False)

: whether a regressor supports multi-target outputs or a classifier supports multi-class multi-output.

multioutput_only (default=False)

: whether estimator supports only multi-output classification or regression.

no_validation (default=False)

: whether the estimator skips input-validation. This is only meant for stateless and dummy transformers!

non_deterministic (default=False)

: whether the estimator is not deterministic given a fixed `random_state`

pairwise (default=False)

: This boolean attribute indicates whether the data (*X*) :term:`fit` and similar methods consists of pairwise measures over samples rather than a feature representation for each sample. It is usually *True* where an estimator has a *metric* or *affinity* or *kernel* parameter with value 'precomputed'. Its primary purpose is to support a :term:`meta-estimator` or a cross validation procedure that extracts a sub-sample of data intended for a pairwise estimator, where the data needs to be indexed on both axes. Specifically, this tag is used by :func:`~sklearn.utils.metaestimators._safe_split` to slice rows and columns.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-**

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 534); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 534); backlink

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 534); backlink

Unknown interpreted text role "func".

preserves_dtype (default=``[np.float64]``)

applies only on transformers. It corresponds to the data types which will be preserved such that *X_trans.dtype* is the same as *X.dtype* after calling *transformer.transform(X)*. If this list is empty, then the transformer is not expected to preserve the data type. The first value in the list is considered as the default data type, corresponding to the data type of the output when the input data type is not going to be preserved.

poor_score (default=False)

whether the estimator fails to provide a "reasonable" test-set score, which currently for regression is an R2 of 0.5 on a subset of the boston housing dataset, and for classification an accuracy of 0.83 on `make_blobs(n_samples=300, random_state=0)`. These datasets and values are based on current estimators in sklearn and might be replaced by something more systematic.

requires_fit (default=True)

whether the estimator requires to be fitted before calling one of *transform*, *predict*, *predict_proba*, or *decision_function*.

requires_positive_X (default=False)

whether the estimator requires positive X.

requires_y (default=False)

whether the estimator requires y to be passed to *fit*, *fit_predict* or *fit_transform* methods. The tag is True for estimators inheriting from *~sklearn.base.RegressorMixin* and *~sklearn.base.ClassifierMixin*.

requires_positive_y (default=False)

whether the estimator requires a positive y (only applicable for regression).

_skip_test (default=False)

whether to skip common tests entirely. Don't use this unless you have a *very good* reason.

_xfail_checks (default=False)

dictionary `{check_name: reason}` of common checks that will be marked as *XFAIL* for pytest, when using :func:`~sklearn.utils.estimator_checks.parametrize_with_checks`. These checks will be simply ignored and not run by :func:`~sklearn.utils.estimator_checks.check_estimator`, but a *SkipTestWarning* will be raised. Don't use this unless there is a *very good* reason for your estimator not to pass the check. Also note that the usage of this tag is highly subject to change because we are trying to make it more flexible: be prepared for breaking changes in the future.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 581); backlink

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 581); backlink

Unknown interpreted text role "func".

stateless (default=False)

whether the estimator needs access to data for fitting. Even though an estimator is stateless, it might still need a call to `fit`

for initialization.

X_types (default=['2darray'])

Supported input types for X as list of strings. Tests are currently only run if '2darray' is contained in the list, signifying that the estimator takes continuous 2d numpy arrays as input. The default value is ['2darray']. Other possible types are `'string'`, `'sparse'`, `'categorical'`, `dict`, `'1dlabels'` and `'2dlabels'`. The goal is that in the future the supported input type will determine the data used during testing, in particular for `'string'`, `'sparse'` and `'categorical'` data. For now, the test for sparse data do not make use of the `'sparse'` tag.

It is unlikely that the default values for each tag will suit the needs of your specific estimator. Additional tags can be created or default tags can be overridden by defining a *_more_tags()* method which returns a dict with the desired overridden tags or new tags. For example:

```
class MyMultiOutputEstimator(BaseEstimator):

    def _more_tags(self):
        return {'multioutput_only': True,
                'non_deterministic': True}
```

Any tag that is not in *_more_tags()* will just fall-back to the default values documented above.

Even if it is not recommended, it is possible to override the method *_get_tags()*. Note however that **all tags must be present in the dict**. If any of the keys documented above is not present in the output of *_get_tags()*, an error will occur.

In addition to the tags, estimators also need to declare any non-optional parameters to `__init__` in the `_required_parameters` class attribute, which is a list or tuple. If `_required_parameters` is only `["estimator"]` or `["base_estimator"]`, then the estimator will be instantiated with an instance of `LogisticRegression` (or `RidgeRegression` if the estimator is a regressor) in the tests. The choice of these two models is somewhat idiosyncratic but both should provide robust closed-form solutions.

Coding guidelines

The following are some guidelines on how new code should be written for inclusion in scikit-learn, and which may be appropriate to adopt in external projects. Of course, there are special cases and there will be exceptions to these rules. However, following these rules when submitting new code makes the review easier so new code can be integrated in less time.

Uniformly formatted code makes it easier to share code ownership. The scikit-learn project tries to closely follow the official Python guidelines detailed in PEP8 that detail how code should be formatted and indented. Please read it and follow it.

In addition, we add the following guidelines:

- Use underscores to separate words in non class names: `n_samples` rather than `nsamples`.
- Avoid multiple statements on one line. Prefer a line return after a control flow statement (`if`/`for`).
- Use relative imports for references inside scikit-learn.
- Unit tests are an exception to the previous rule; they should use absolute imports, exactly as client code would. A corollary is that, if `sklearn.foo` exports a class or function that is implemented in `sklearn.foo.bar.baz`, the test should import it from `sklearn.foo`.
- **Please don't use** `import *` **in any case**. It is considered harmful by the official Python recommendations. It makes the code harder to read as the origin of symbols is no longer explicitly referenced, but most important, it prevents using a static analysis tool like pyflakes to automatically find bugs in scikit-learn.
- Use the numpy docstring standard in all your docstrings.

A good example of code that we like can be found here.

Input validation

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 689)

Unknown directive type "currentmodule".

```
.. currentmodule:: sklearn.utils
```

The module :mod:`sklearn.utils` contains various functions for doing input validation and conversion. Sometimes, `np.asarray` suffices for validation; do *not* use `np.asanyarray` or `np.atleast_2d`, since those let NumPy's `np.matrix` through, which has a different API (e.g., `*` means dot product on `np.matrix`, but Hadamard product on `np.ndarray`).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 691); *backlink*

Unknown interpreted text role "mod".

In other cases, be sure to call :func:`check_array` on any array-like argument passed to a scikit-learn API function. The exact parameters to use depends mainly on whether and which `scipy.sparse` matrices must be accepted.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 698);** ***backlink***
>
> Unknown interpreted text role "func".

For more information, refer to the :ref:`developers-utils` page.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 702);** ***backlink***
>
> Unknown interpreted text role "ref".

Random Numbers

If your code depends on a random number generator, do not use `numpy.random.random()` or similar routines. To ensure repeatability in error checking, the routine should accept a keyword `random_state` and use this to construct a `numpy.random.RandomState` object. See :func:`sklearn.utils.check_random_state` in :ref:`developers-utils`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 707);** ***backlink***
>
> Unknown interpreted text role "func".

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 707);** ***backlink***
>
> Unknown interpreted text role "ref".

Here's a simple example of code using some of the above guidelines:

```
from sklearn.utils import check_array, check_random_state

def choose_random_sample(X, random_state=0):
    """Choose a random point from X.

    Parameters
    ----------
    X : array-like of shape (n_samples, n_features)
        An array representing the data.
    random_state : int or RandomState instance, default=0
        The seed of the pseudo random number generator that selects a
        random sample. Pass an int for reproducible output across multiple
        function calls.
        See :term:`Glossary <random_state>`.

    Returns
    -------
    x : ndarray of shape (n_features,)
        A random point selected from X.
    """
    X = check_array(X)
    random_state = check_random_state(random_state)
    i = random_state.randint(X.shape[0])
    return X[i]
```

If you use randomness in an estimator instead of a freestanding function, some additional guidelines apply.

First off, the estimator should take a `random_state` argument to its `__init__` with a default value of `None`. It should store that argument's value, **unmodified**, in an attribute `random_state`. `fit` can call `check_random_state` on that attribute to get an actual random number generator. If, for some reason, randomness is needed after `fit`, the RNG should be stored in an attribute `random_state_`. The following example should make this clear:

```
class GaussianNoise(BaseEstimator, TransformerMixin):
    """This estimator ignores its input and returns random Gaussian noise.

    It also does not adhere to all scikit-learn conventions,
```

```
    but showcases how to handle randomness.
    """

    def __init__(self, n_components=100, random_state=None):
        self.random_state = random_state
        self.n_components = n_components

    # the arguments are ignored anyway, so we make them optional
    def fit(self, X=None, y=None):
        self.random_state_ = check_random_state(self.random_state)

    def transform(self, X):
        n_samples = X.shape[0]
        return self.random_state_.randn(n_samples, self.n_components)
```

The reason for this setup is reproducibility: when an estimator is `fit` twice to the same data, it should produce an identical model both times, hence the validation in `fit`, not `__init__`.

Numerical assertions in tests

When asserting the quasi-equality of arrays of continuous values, do use :func:`sklearn.utils._testing.assert_allclose`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 781); *backlink***
>
> Unknown interpreted text role "func".

The relative tolerance is automatically inferred from the provided arrays dtypes (for float32 and float64 dtypes in particular) but you can override via `rtol`.

When comparing arrays of zero-elements, please do provide a non-zero value for the absolute tolerance via `atol`.

For more information, please refer to the docstring of :func:`sklearn.utils._testing.assert_allclose`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\scikit-learn-main\doc\developers\[scikit-learn-main][doc][developers]develop.rst, line 791); *backlink***
>
> Unknown interpreted text role "func".