OOP with Scikit-Learn

Introduction

As you learn more about machine learning algorithms, there are typically two components. First, the conceptual underlying logic of the algorithm -- how it works to process inputs and generate outputs. Second, the scikit-learn implementation of the algorithm -- how to use it in practice.

Before diving into specific examples of various scikit-learn models, it is helpful to understand the general structure they follow. Specifically, we'll go over some key classes, methods, and attributes common to scikit-learn.

Objectives

In this lesson you will:

- Recall the distinction between mutable and immutable types
- Define the four main inherited object types in scikit-learn
- Instantiate scikit-learn transformers and models
- Invoke scikit-learn methods
- Access scikit-learn attributes

Mutable and Immutable Data Types

In base Python, the built-in types are either mutable or immutable.

Mutable data types are data types that can be modified after they are initialized. For example, a list is a mutable data type in Python.

In [1]:
```
my_list = [1, 2, 3]
my_list
```

Out[1]:
```
[1, 2, 3]
```

One way you can mutate a Python `list` is using the `append` method:

In [2]:
```
my_list.append(4)
my_list
```

Out[2]:
```
[1, 2, 3, 4]
```

This is in contrast to ***immutable*** data types, which can't be modified after they are initialized. For example, a string is an immutable data type in Python.

In [3]:
```
my_str = "Hello!"
my_str
```

Out[3]:
```
'Hello!'
```

We can call methods on strings, but it doesn't modify their value:

In [4]:
```
my_str.upper()
my_str
```

Out[4]:
```
'Hello!'
```

This same principle applies to custom classes beyond base Python, including the classes used by scikit-learn.

Most scikit-learn classes are *mutable*, which means that calling methods on them changes their internal data.

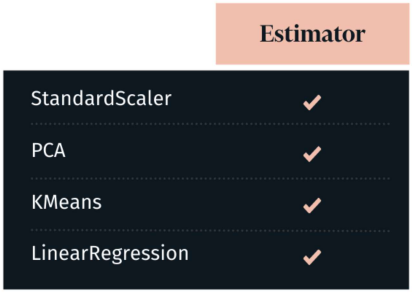
Scikit-Learn Classes

Scikit-learn has four main classes to be aware of:

- Estimator
- Transformer
- Predictor
- Model

They are defined based on which methods they possess. The classes are **not mutually exclusive**.

Estimator



Almost all scikit-learn classes you will use will be some kind of estimator. It is the "base object" in scikit-learn.

An estimator is defined by having a `fit` method. There are two typical forms for this method:

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

and

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

The first one is typically used in the context of a transformer or unsupervised learning predictor, while the second is used in the context of a supervised learning predictor.

Transformer

A transformer is an estimator that has a `transform` method:

```
transformer.transform(data)
```

The `transform` method is called after the `fit` method and returns a modified form of the input data.

An example of a transformer (that is not also a predictor or model) is:

`StandardScaler`

`StandardScaler` is used to standardize features by removing the mean and scaling to unit variance (documentation here (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html))

In [5]:
```
# Import class from scikit-learn
from sklearn.preprocessing import StandardScaler

# Instantiate the scaler (same step for all estimators, though specific args differ)
scaler = StandardScaler()
```

When the estimator is first instantiated, these are all of its attributes:

In [6]:
```
scaler.__dict__
```

Out[6]:
```
{'with_mean': True, 'with_std': True, 'copy': True}
```

(Note: the `__dict__` attribute starts with an underscore so it is not intended for "public" use and may not work consistently in the future. Look at the documentation page to see the list of public attributes.)

The next step, like with any scikit-learn estimator, is to fit the scaler on the data:

In [7]:
```
# Data representing a single feature
data = [[10], [20], [30], [40], [50]]

# Fit the scaler (same step for all estimators, though specific args differ)
scaler.fit(data)
```

Out[7]:
```
StandardScaler()
```

Now that `fit` has been called, because transformers are *mutable*, there are additional attributes:

In [8]:
```
scaler.__dict__
```

Out[8]:
```
{'with_mean': True,
 'with_std': True,
 'copy': True,
 'n_features_in_': 1,
 'n_samples_seen_': 5,
 'mean_': array([30.]),
 'var_': array([200.]),
 'scale_': array([14.14213562])}
```

The underscore (`_`) at the end of these new variables (e.g. `mean_`) is a scikit-learn convention, which means that these attributes are not available until the estimator has been fit.

We can access these fitted attributes using the standard dot notation:

In [9]:
```
scaler.mean_
```

Out[9]:
```
array([30.])
```

Now that the scaler is fit, we can use it to transform the data:

In [10]:
```
scaler.transform(data)
```

Out[10]:
```
array([[-1.41421356],
       [-0.70710678],
       [ 0.        ],
       [ 0.70710678],
       [ 1.41421356]])
```

Note that even though we passed in a base Python list, the scaler returned a NumPy `ndarray` . Transformers always return this type of array regardless of whether you pass in a base Python data structure, a NumPy data structure, or a pandas data structure.

Some additional examples of transformers (that aren't also predictors) are:

- `OneHotEncoder` (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html): used to convert categorical features into one-hot encoded features
- `CountVectorizer` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html): used to convert text data into a matrix of token counts

Predictor

As you might have...*predicted*...a predictor is an estimator that has a `predict` method:

```
predictor.predict(X)
```

The `predict` method is called after the `fit` method and can be part of a supervised or unsupervised learning model. It returns a list of predictions `y` associated with the input data `X`.

An example of a predictor is:

`LinearRegression`

`LinearRegression` is a class that represents an ordinary least squares linear regression model (documentation here (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html))

In [11]:
```
# Import class from scikit-learn
from sklearn.linear_model import LinearRegression

# Instantiate the model (same step for all estimators, though specific args differ)
lr = LinearRegression()
```

When the estimator is first instantiated, these are all of its attributes:

In [12]:
```
lr.__dict__
```

Out[12]:
```
{'fit_intercept': True, 'normalize': False, 'copy_X': True, 'n_jobs': None}
```

The next step, like with any scikit-learn estimator, is to fit the linear regression on the data:

In [13]:
```
# Data representing X (features) and y (target), where y = 10x + 5
X = [[1], [2], [3], [4], [5]]
y = [15, 25, 35, 45, 55]

# Fit the linear regression (same step for all estimators, though specific args differ)
lr.fit(X, y)
```

Out[13]:
```
LinearRegression()
```

Note that this differs from the `fit` method in the `StandardScaler` (and most transformers) because it requires both `X` and `y`.

Once again, there are additional attributes now that `fit` has been called, since `LinearRegression` is mutable:

In [14]:
```
lr.__dict__
```

Out[14]:
```
{'fit_intercept': True,
 'normalize': False,
 'copy_X': True,
 'n_jobs': None,
 'n_features_in_': 1,
 'coef_': array([10.]),
 '_residues': 1.8932661725304283e-29,
 'rank_': 1,
 'singular_': array([3.16227766]),
 'intercept_': 5.000000000000007}
```

We can access the fitted attributes using dot notation. For example, below we access the intercept and coefficient of the regression:

In [15]:
```
print(lr.intercept_)
print(lr.coef_[0])
```

```
5.000000000000007
9.999999999999998
```

Because this is a predictor and not a transformer, the next step is to use the `predict` method rather than the `transform` method:

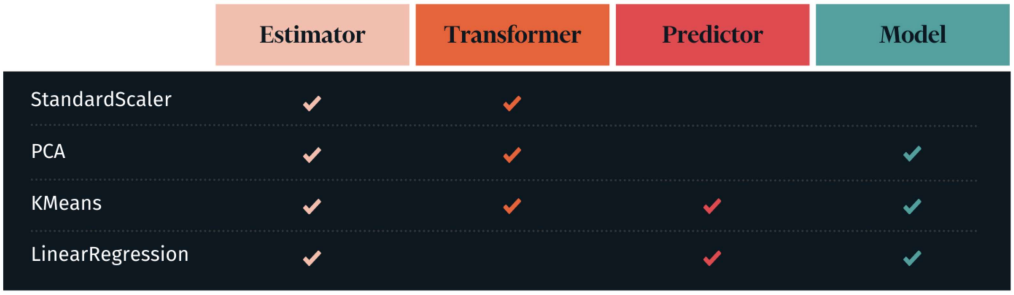
In [16]:
```
lr.predict(X)
```

Out[16]:
```
array([15., 25., 35., 45., 55.])
```

Some additional examples of predictors (that aren't also transformers) are:

- `LogisticRegression` (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html): a classifier that uses the logistic regression algorithm
- `KNeighborsRegressor` (https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html): a regressor that uses the k-nearest neighbors algorithm
- `DecisionTreeClassifier` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html): a classifier that uses the decision tree algorithm

Model



A model is an estimator that has a `score` method. There are two typical forms for this method:

```
model.score(X, y)
```

and

```
model.score(X)
```

For example, using the linear regression model from above, we can score the model using r-squared:

In [17]:
```
lr.score(X, y)
```

Out[17]:
```
1.0
```

An example of a model that produces a score with just `X` would be `PCA` (documentation here (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)):

In [18]:
```
# Import class from scikit-learn
from sklearn.decomposition import PCA

# Instantiate the model (same step for all estimators, though specific args differ)
pca = PCA(n_components=1)
```

In [19]:
```
pca.__dict__
```

Out[19]:
```
{'n_components': 1,
 'copy': True,
 'whiten': False,
 'svd_solver': 'auto',
 'tol': 0.0,
 'iterated_power': 'auto',
 'random_state': None}
```

In [20]:
```
# Data representing two features
X = [[1, 11], [2, 12], [3, 14], [4, 16], [5, 18]]

# Fit the PCA (same step for all estimators, though specific args differ)
pca.fit(X)
```

Out[20]:
```
PCA(n_components=1)
```

In [21]:
```
pca.__dict__
```

Out[21]:
```
{'n_components': 1,
 'copy': True,
 'whiten': False,
 'svd_solver': 'auto',
 'tol': 0.0,
 'iterated_power': 'auto',
 'random_state': None,
 'n_features_in_': 2,
 '_fit_svd_solver': 'full',
 'mean_': array([ 3. , 14.2]),
 'noise_variance_': 0.023415728630588915,
 'n_samples_': 5,
 'n_features_': 2,
 'components_': array([[0.48215553, 0.87608564]]),
 'n_components_': 1,
 'explained_variance_': array([10.67658427]),
 'explained_variance_ratio_': array([0.99781161]),
 'singular_values_': array([6.53500858])}
```

In [22]:
```
pca.score(X)
```

Out[22]:
```
-1.9447298858494009
```

To understand what a given score means, look at the documentation for the model (e.g. here (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=score#sklearn.linear_model.LinearRegression.score) for `LinearRegression` or here (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html?highlight=score#sklearn.decomposition.PCA.score) for `PCA`).

Overlapping Classes

As stated previously, these scikit-learn classes are not mutually exclusive.

`StandardScaler` is an **estimator** and a **transformer** but not a predictor or a model.

`LinearRegression` is an **estimator**, a **predictor**, and a **model** but not a transformer.

`KMeans` is an **estimator**, a **transformer**, a **predictor**, and a **model**.

`PCA` is an **estimator**, a **transformer**, and a **model** but not a predictor.

(Don't worry if you're not sure what all of these classes are used for. We'll get there eventually!)

Takeaways

You do not need to memorize these labels for every scikit-learn class you encounter. You can always figure out what a class can do by looking at its documentation:

- If it has a `fit` method, it's an estimator
- If it has a `transform` method, it's a transformer
- If it has a `predict` method, it's a predictor
- If it has a `score` method, it's a model

Recognizing these terms can help you navigate the official documentation as well as third-party resources, which might refer to these classes and their instances with various labels interchangeably, since multiple labels often apply.

Also, keep in mind that estimators are mutable and store important information during the `fit` step, which means that you always need to call the `fit` method before you can call the `transform`, `predict`, or `score` methods.

Summary

In this lesson, you learned about the four main classes in scikit-learn: estimators, transformers, predictors, and models. You saw how the attributes of the estimators changed when the `fit` method was called, as well as how to use other methods such as `transform`, `predict`, and `score`.