The General Pipeline Interface

The Pipeline class is not restricted to preprocessing and classification, but can in fact join any number of estimators together. For example, you could build a pipeline containing feature extraction, feature selection, scaling, and classification, for a total of four steps. Similarly, the last step could be regression or clustering instead of classification.

The only requirement for estimators in a pipeline is that all but the last step need to have a transform method, so they can produce a new representation of the data that can be used in the next step.

Internally, during the call to Pipeline.fit, the pipeline calls fit and then transform on each step in turn,[2] with the input given by the output of the transform method of the previous step. For the last step in the pipeline, just fit is called.

Brushing over some finer details, this is implemented as follows. Remember that pipeline.steps is a list of tuples, so pipeline.steps[0][1] is the first estimator, pipeline.steps[1][1] is the second estimator, and so on:

In[15]:

```
def fit(self, X, y):
    X_transformed = X
    for name, estimator in self.steps[:-1]:
        # iterate over all but the final step
        # fit and transform the data
        X_transformed = estimator.fit_transform(X_transformed, y)
    # fit the last step
    self.steps[-1][1].fit(X_transformed, y)
    return self
```

When predicting using Pipeline, we similarly transform the data using all but the last step, and then call predict on the last step:

In[16]:

```
def predict(self, X):
    X_transformed = X
    for step in self.steps[:-1]:
        # iterate over all but the final step
        # transform the data
        X_transformed = step[1].transform(X_transformed)
    # fit the last step
    return self.steps[-1][1].predict(X_transformed)
```

[2] Or just fit_transform.

The process is illustrated in Figure 6-3 for two transformers, T1 and T2, and a classifier (called Classifier).

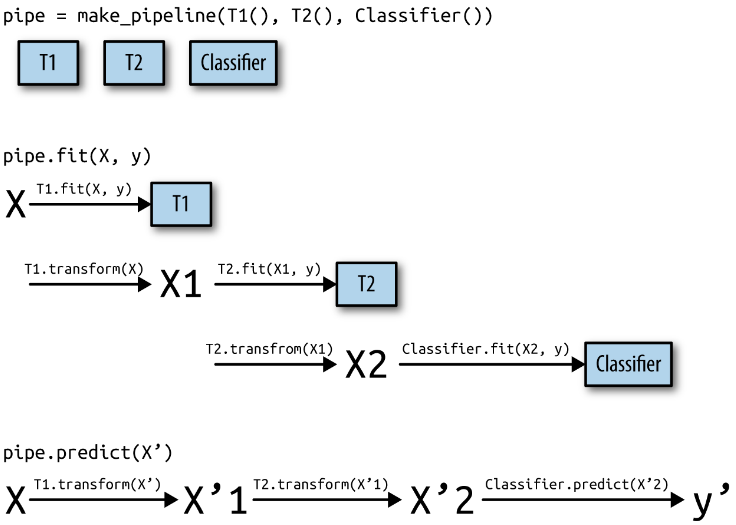

Figure 6-3. Overview of the pipeline training and prediction process

The pipeline is actually even more general than this. There is no requirement for the last step in a pipeline to have a predict function, and we could create a pipeline just containing, for example, a scaler and PCA. Then, because the last step (PCA) has a transform method, we could call transform on the pipeline to get the output of PCA.transform applied to the data that was processed by the previous step. The last step of a pipeline is only required to have a fit method.

Convenient Pipeline Creation with make_pipeline

Creating a pipeline using the syntax described earlier is sometimes a bit cumbersome, and we often don't need user-specified names for each step. There is a convenience function, make_pipeline, that will create a pipeline for us and automatically name each step based on its class. The syntax for make_pipeline is as follows:

In[17]:

```
from sklearn.pipeline import make_pipeline
# standard syntax
pipe_long = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC(C=100))])
# abbreviated syntax
pipe_short = make_pipeline(MinMaxScaler(), SVC(C=100))
```