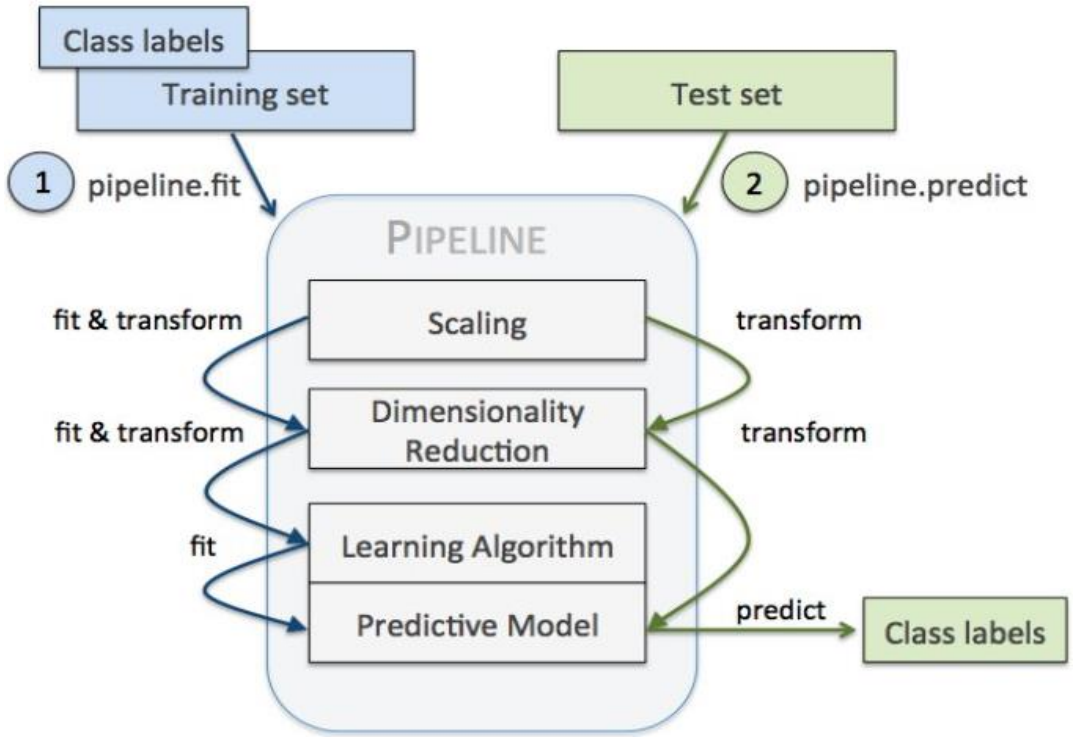
Pipeline:
Workflow Optimization

Pipeline: chaining estimators

- Pipeline can be used to chain multiple estimators into one.
- Pipeline serves two purposes:
  - Convenience and encapsulation
  - Joint parameter selection



Building Pipelines

```
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
# load and split the data
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)
```

```
from sklearn.pipeline import Pipeline
pipe = Pipeline([("scaler", MinMaxScaler()), ("svm", SVC())])
```

The **Pipeline** is built using a list of **(key, value)** pairs, where the **key** is a string containing the name you want to give this step and **value** is an estimator object:

```
pipe.fit(X_train, y_train).score(X_test, y_test)
```

```
0.951048951048951
```

You only have to call **fit** and **predict** once on your data to fit a whole sequence of estimators

Using Pipelines in Grid-searches

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'svm__C': [0.001, 0.01, 0.1, 1, 10, 100],
              'svm__gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

Parameters of the estimators in the pipeline shoud be defined using the **estimator__parameter** syntax

```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best cross-validation accuracy: {:.2f}".format(
    grid.best_score_))
print("Test set score: {:.2f}".format(grid.score(X_test, y_test)))
print("Best parameters: {}".format(grid.best_params_))
```

```
Best cross-validation accuracy: 0.98
Test set score: 0.97
Best parameters: {'svm__C': 1, 'svm__gamma': 1}
```

Convenient Pipeline creation with *make_pipeline*

```
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))
```

```
print("Pipeline steps:\n{}".format(pipe_short.steps))
```

```
Pipeline steps:
[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('svc',
SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))]
```

Make_pipeline does not require, and does not permit, naming the estimators. Instead, their names will be set to the **lowercase of their types** automatically.

Pipeline Interface

- All estimators in a pipeline, except the last one, must be transformers. The last estimator may be any type (transformer, classifier, etc.)
- Training and prediction procedure of the pipeline