Introduction to Machine Learning with Python

6. Algorithm chains and Pipelines

Honedae Machine Learning Study Epoch #2

Contacts

Haesun Park

Email : haesunrpark@gmail.com

Meetup: https://www.meetup.com/Hongdae-Machine-Learning-Study/

Facebook : https://facebook.com/haesunrpark

Blog : https://tensorflow.blog

Book

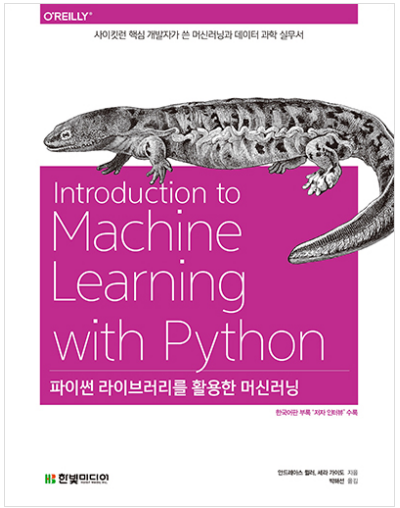
파이썬 라이브러리를 활용한 머신러닝, 박해선.

(Introduction to Machine Learning with Python, Andreas Muller & Sarah Guido의 번역서입니다.)

번역서의 1장과 2장은 블로그에서 무료로 읽을 수 있습니다.

원서에 대한 프리뷰를 온라인에서 볼 수 있습니다.

Github:
https://github.com/rickiepark/introduction_to_ml_with_python/

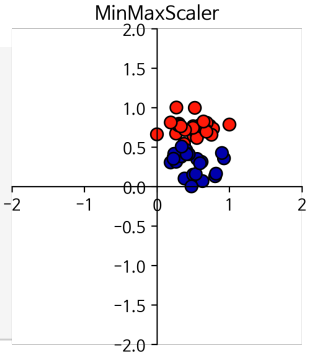


Algorithm Chains

cancer + MinMaxScaler + SVC

```
# 데이터 적재와 분할
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

# 훈련 데이터의 최솟값, 최댓값을 계산합니다
scaler = MinMaxScaler().fit(X_train)
```




```
# 훈련 데이터의 스케일을 조정합니다
X_train_scaled = scaler.transform(X_train)

svm = SVC()
# 스케일 조정된 훈련데이터에 SVM을 학습시킵니다
svm.fit(X_train_scaled, y_train)
# 테스트 데이터의 스케일을 조정하고 점수를 계산합니다
X_test_scaled = scaler.transform(X_test)
print("테스트 점수: {:.2f}".format(svm.score(X_test_scaled, y_test)))
```

테스트 점수: 0.95

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

모든 특성이
0과 1사이에 위치

알고리즘 체인의 필요성

입력 데이터의 표현 형태에 민감한 알고리즘이 많습니다.

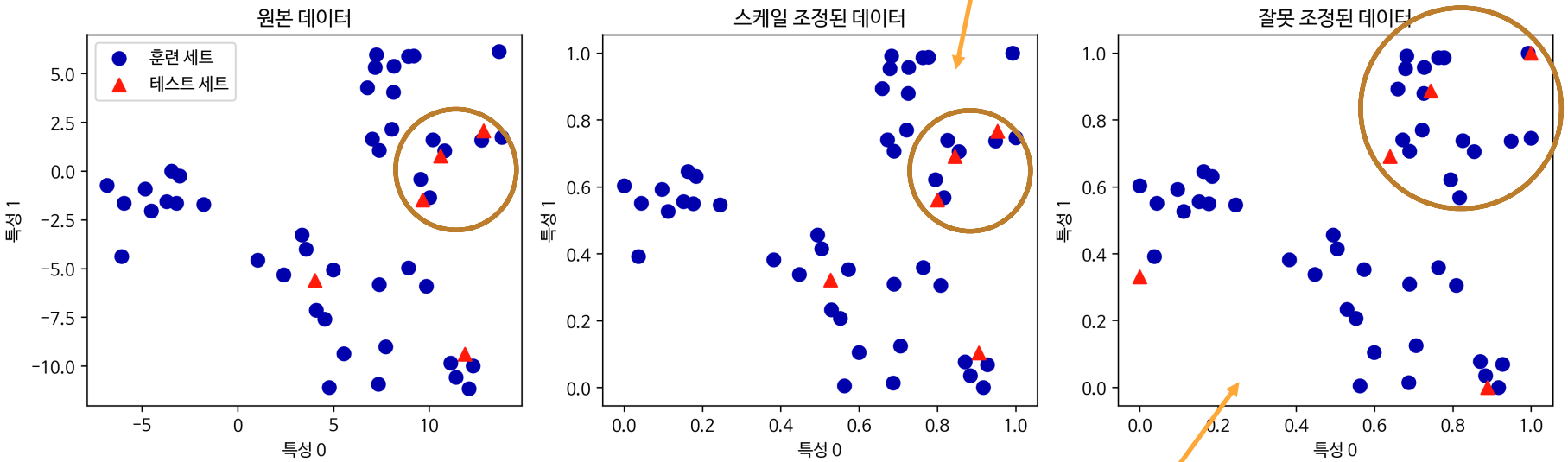
데이터 스테일 조정, 특성 연결(다항 특성), 비지도 학습으로 새로운 특성 생성(PCA) 등 대부분 머신러닝 애플리케이션은 하나의 알고리즘이 아니라 여러 단계의 처리과정과 모델이 연결되어 있습니다.

Scikit-Learn은 이런 전처리 단계와 모델을 연결하는 Pipeline 클래스를 제공합니다.

Pipeline과 GridSearchCV를 사용해 전처리 과정에 필요한 매개변수를 탐색할 수 있습니다.

train과 test의 스케일 조정

```
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```




```
test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_test_scaled_badly = test_scaler.transform(X_test)
```

단순한 매개변수 탐색

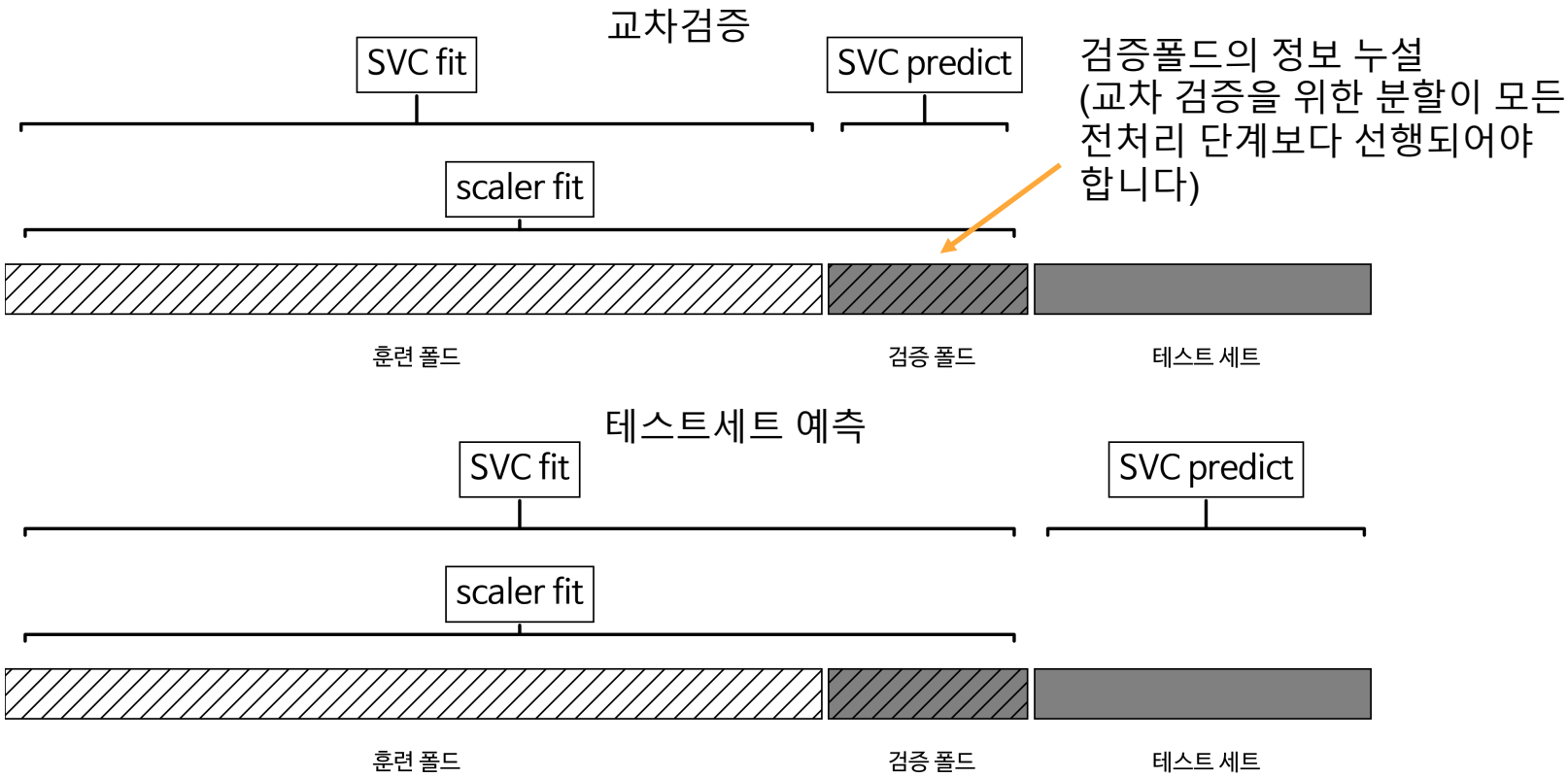
테스트 데이터는 훈련 데이터의 통계값으로 스케일을 조정합니다.

하지만 GridSearchCV안의 검증 데이터는 스케일 조정에 사용한 후입니다.

```
from sklearn.model_selection import GridSearchCV
# 이 코드는 예를 위한 것입니다. 실제로 사용하지 마세요.
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
grid = GridSearchCV(SVC(), param_grid=param_grid, cv=5)
grid.fit(X_train_scaled, y_train)
print("최상의 교차 검증 정확도: {:.2f}".format(grid.best_score_))
print("테스트 점수: {:.2f}".format(grid.score(X_test_scaled, y_test)))
print("최적의 매개변수: ", grid.best_params_)
```

```
최상의 교차 검증 정확도: 0.98
테스트 점수: 0.97
최적의 매개변수:  {'C': 1, 'gamma': 1}
```

Validation vs. Predict



정보 누설의 예

헤이스티, 팁시라니, 프리드먼의 ESL에 포함된 예제

100개의 샘플, 1만개의 랜덤 특성

```
rnd = np.random.RandomState(seed=0)
X = rnd.normal(size=(100, 10000))
y = rnd.normal(size=(100,))
```

5%=500개 특성만 선택

```
select = SelectPercentile(score_func=f_regression, percentile=5).fit(X, y)
X_selected = select.transform(X)
from sklearn.linear_model import Ridge
print("교차 검증 점수 (릿지): {:.2f}".format(
      np.mean(cross_val_score(Ridge(), X_selected, y, cv=5))))
```

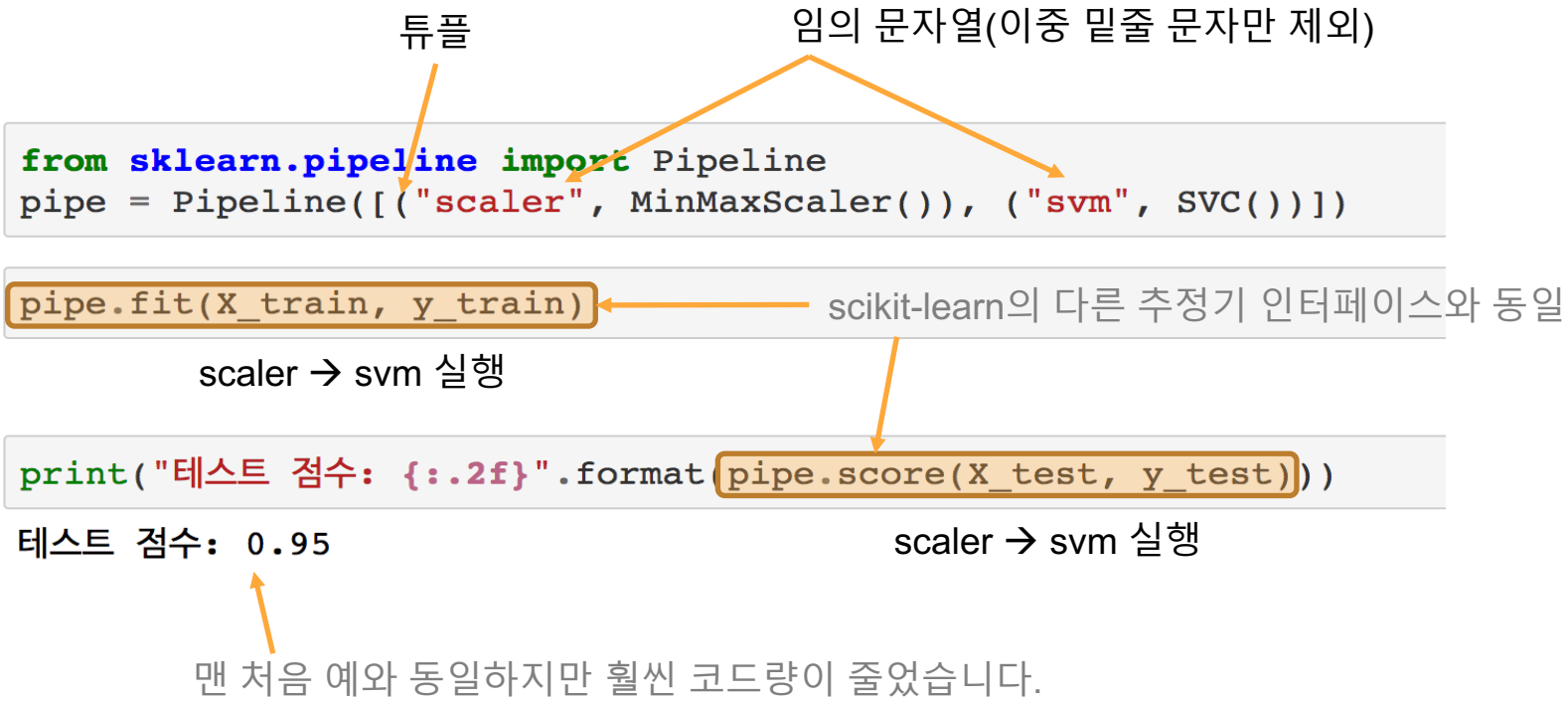
```
교차 검증 점수 (릿지): 0.91
```

검증 데이터를 포함해서 데이터를 변환했기 때문에 상호연관된 특성이 골라졌습니다

```
pipe = Pipeline([("select", SelectPercentile(score_func=f_regression,
                                             percentile=5)),
                 ("ridge", Ridge())])
print("교차 검증 점수 (파이프라인): {:.2f}".format(
      np.mean(cross_val_score(pipe, X, y, cv=5))))
```

```
교차 검증 점수 (파이프라인): -0.25
```

Simple Pipeline



Pipeline + GridSearchCV

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

```
pipe.fit(X_train, y_train)
```

두 개의 밑줄 문자로 파이프라인 단계 구분

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

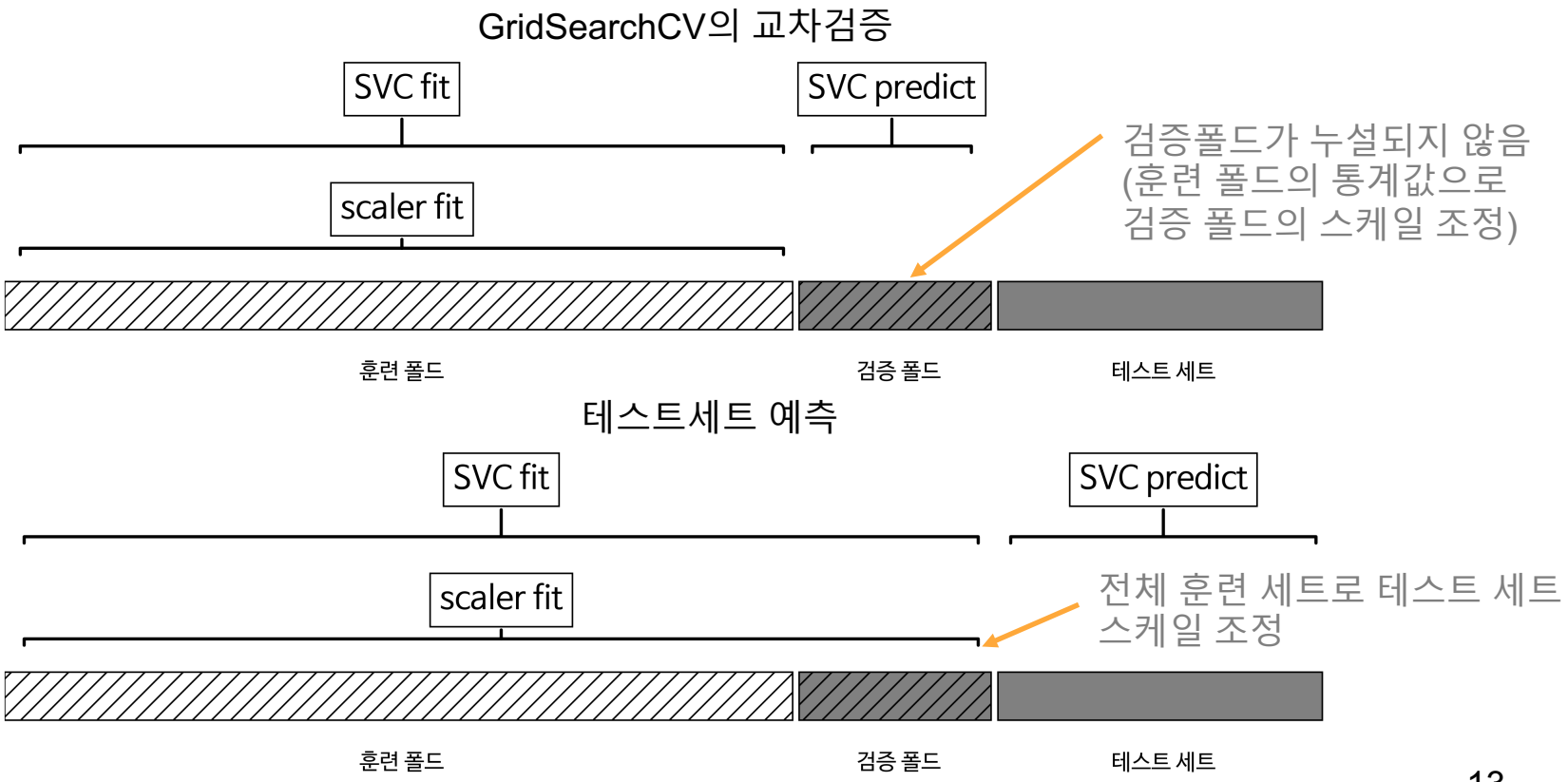
그리드서치만 사용했을 때 매개변수 그리드

```
param_grid = {'C': [0.001, 0.01,
                    'gamma': [0.001, 0.
```

```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
print("최상의 교차 검증 정확도: {:.2f}".format(grid.best_score_))
print("테스트 세트 점수: {:.2f}".format(grid.score(X_test, y_test)))
print("최적의 매개변수: {}".format(grid.best_params_))
```

```
최상의 교차 검증 정확도: 0.98
테스트 세트 점수: 0.97
최적의 매개변수: {'svm__C': 1, 'svm__gamma': 1}
```

Validation vs. Predict 2

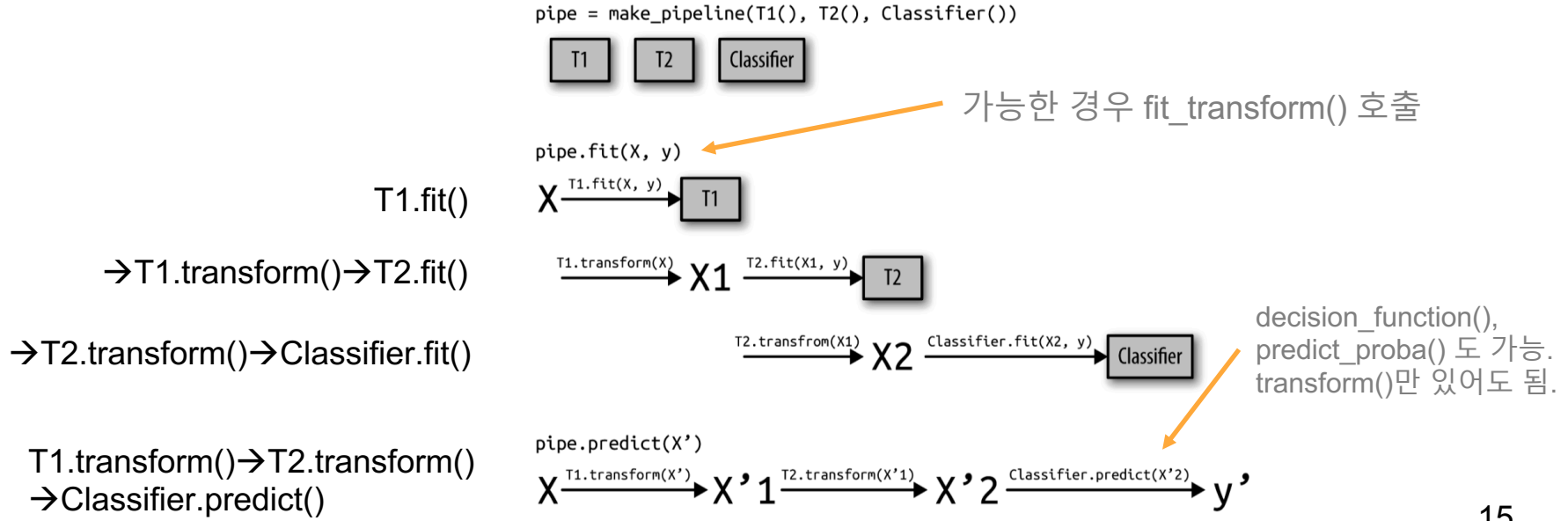


Pipeline

Pipeline 인터페이스

여러개의 어떤 Estimator 클래스와(특성 추출, 특성 선택, 스케일 변경, 분류/회귀)도 연결 가능합니다.

마지막 단계를 빼고는 모두 transform() 메서드가 있어야 합니다.



make_pipeline

파이프라인 단계에 자동으로 이름을 부여하여 파이프라인 객체를 만듭니다.

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

소문자 클래스 이름

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

```
파이프라인 단계:
[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('svc', SVC(C=100, cache_siz
  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))]
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

pipe = make_pipeline(StandardScaler(), PCA(n_components=2), StandardScaler())
print("파이프라인 단계:\n{}".format(pipe.steps))
```

두 개 이상일 때

```
파이프라인 단계:
[('standardscaler-1', StandardScaler(copy=True, with_mean=True, with_std=True)), ('pca', PCA(c
  svd_solver='auto', tol=0.0, whiten=False)), ('standardscaler-2', StandardScaler(copy=True, w
```

Pipeline 단계에 접근하기

선형 모델의 계수나 PCA 주성분을 확인할 때 파이프라인 단계에 접근할 필요가 있습니다.

named_steps 속성을 사용하면 대해 단계 이름을 키로 하여 각 객체에 접근할 수 있습니다.

```
# cancer 데이터셋에 앞서 만든 파이프라인을 적용합니다
pipe.fit(cancer.data)
# "pca" 단계의 두 개 주성분을 추출합니다
components = pipe.named_steps["pca"].components_
print("components.shape: {}".format(components.shape))
```

named_steps 딕셔너리

```
components.shape: (2, 30)
```

그리드서치 Pipeline

```
pipe = make_pipeline(StandardScaler(), LogisticRegression())
```

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=4)
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)
```

GridSearchCV의 best_estimator_ 속성에 최적의 Pipeline 객체 저장

```
print("로지스틱 회귀 단계:\n{}".format(
      grid.best_estimator_.named_steps["logisticregression"]))
```

```
로지스틱 회귀 단계:
LogisticRegression(C=0.1, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
print("로지스틱 회귀 계수:\n{}".format(
      grid.best_estimator_.named_steps["logisticregression"].coef_))
```

```
로지스틱 회귀 계수:
[[-0.389 -0.375 -0.376 -0.396 -0.115  0.017 -0.355 -0.39  -0.058  0.209
  -0.495 -0.004 -0.371 -0.383 -0.045  0.198  0.004 -0.049  0.21   0.224
  -0.547 -0.525 -0.499 -0.515 -0.393 -0.123 -0.388 -0.417 -0.325 -0.139]]
```

전처리 매개변수를 위한 그리드서치

```
from sklearn.datasets import load_boston
boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target,
                                                    random_state=0)

from sklearn.preprocessing import PolynomialFeatures
pipe = make_pipeline(
    StandardScaler(),
    PolynomialFeatures(),
    Ridge())
```

make_pipeline에서 만들 이름
(클래스이름의 소문자 버전)

```
param_grid = {'polynomialfeatures__degree': [1, 2, 3],
              'ridge__alpha': [0.001, 0.01, 0.1, 1, 10, 100]}
```

최적의 다항식 차수를
검증 세트 점수로 찾음

```
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=-1)
grid.fit(X_train, y_train)
```

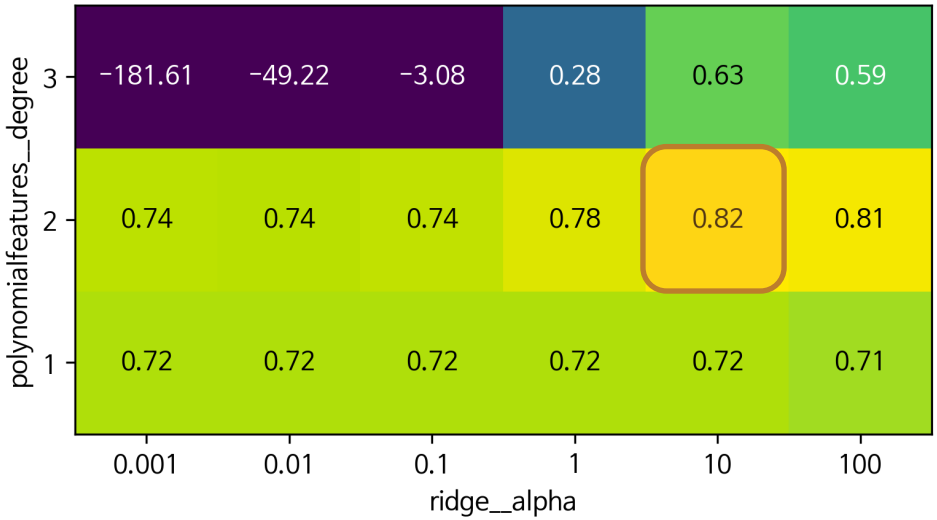
선택된 다항 특성 확인

```
print("최적의 매개변수: {}".format(grid.best_params_))
```

```
최적의 매개변수: {'ridge__alpha': 10, 'polynomialfeatures__degree': 2}
```

```
print("테스트 세트 점수: {:.2f}".format(grid.score(X_test, y_test)))
```

```
테스트 세트 점수: 0.77
```



모델 선택을 위한 그리드서치

파이프라인의 구성 단계를 탐색 대상으로 삼을 수 있습니다(탐색할 모델이 크게 증가됩니다).

매개변수 그리드의 리스트를 사용합니다(5장 비대칭 매개변수 그리드와 비슷).

```
pipe = Pipeline([('preprocessing', StandardScaler()), ('classifier', SVC())])
```

```
from sklearn.ensemble import RandomForestClassifier

param_grid = [
    {'classifier': [SVC()], 'preprocessing': [StandardScaler()],
     'classifier__gamma': [0.001, 0.01, 0.1, 1, 10, 100],
     'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100]},
    {'classifier': [RandomForestClassifier(n_estimators=100)],
     'preprocessing': [None], 'classifier__max_features': [1, 2, 3]}]
```

랜덤포레스트는 전처리가 필요 없습니다.

그리드서치로 모델 선택

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(X_train, y_train)

print("최적의 매개변수:\n{}\n".format(grid.best_params_))
print("최상의 교차 검증 점수: {:.2f}".format(grid.best_score_))
print("테스트 세트 점수: {:.2f}".format(grid.score(X_test, y_test)))
```

```
최적의 매개변수:
{'preprocessing': StandardScaler(copy=True, with_mean=True, with_std=True), 'classifier': SVC(
  decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False), 'classifier__gamma': 0.01, 'classifier__C': 10}

최상의 교차 검증 점수: 0.99
테스트 세트 점수: 0.98
```

Pipeline Caching

Scikit-Learn 0.19 추가 사항

```
pipe = make_pipeline(StandardScaler(), PolynomialFeatures(), Ridge())
grid = GridSearchCV(pipe, param_grid=param_grid, cv=5, n_jobs=-1)
```

```
%timeit grid.fit(X_train, y_train)
```

```
10.7 s ± 644 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
from tempfile import mkdtemp
from shutil import rmtree
```

파이프라인 단계의 transform() 결과를 저장

```
cache_dir = mkdtemp()
pipe2 = make_pipeline(StandardScaler(), PolynomialFeatures(), Ridge(), memory=cache_dir)
grid2 = GridSearchCV(pipe2, param_grid=param_grid, cv=5, n_jobs=-1)
```

```
%timeit grid2.fit(X_train, y_train)
```

```
6.57 s ± 315 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
rmtree(cache_dir)
```

요약 및 정리

Pipeline 클래스는 여러 처리 단계를 하나의 객체로 캡슐화 합니다.

전처리가 있는 매개변수 선택에서는 올바른 교차검증을 위해 필수적입니다.

코드를 간결하게 만들어 주고 실수를 방지할 수 있습니다.

완벽한 조합을 찾는 것은 예술에 가까운 일입니다.

매개변수와 모델 등 탐색 범위가 커지면 시간이 많이 걸릴 수 있습니다.

실험 단계에서는 처리 단계가 꼭 필요한지 검토하고 너무 복잡하게 만들지 않습니다.

6장까지 머신러닝에 필요한 범용적인 도구와 알고리즘을 모두 배웠습니다.

감사합니다.

-질문-