선형 모델(Linear Models)

- 선형 모델은 과거 부터 지금 까지 널리 사용되고 연구 되고 있는 기계학습 방법
- 선형 모델은 입력 데이터에 대한 선형 함수를 만들어 예측 수행
- 회귀 분석을 위한 선형 모델은 다음과 같이 정의

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

- x: 입력 데이터
- w: 모델이 학습할 파라미터
- w_0: 편향
- w_1~w_p: 가중치

선형 회귀(Linear Regression)

- **선형 회귀(Linear Regression)**또는 **최소제곱법(Ordinary Least Squares)**은 가장 간단한 회귀 분석을 위한 선형 모델
- 선형 회귀는 모델의 예측과 정답 사이의 **평균제곱오차(Mean Squared Error)**를 최소화 하는 학습 파라미터 w를 찾음
- 평균제곱오차는 아래와 같이 정의

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

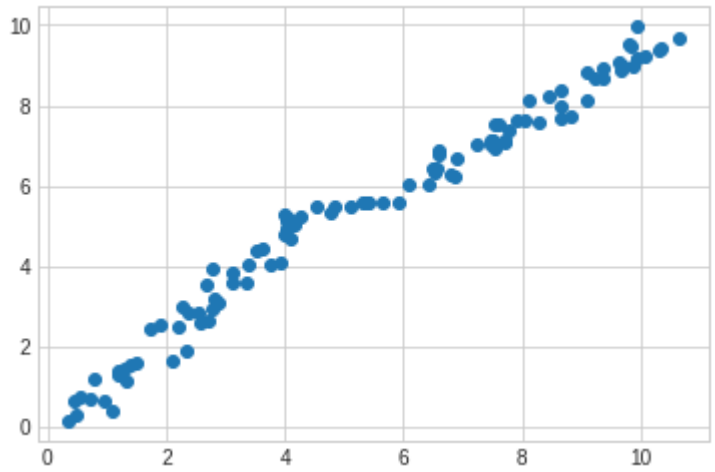
- y: 정답
- $\hat{y}$: 예측 값을 의미
- 선형 회귀 모델에서 사용하는 다양한 오류 측정 방법
  - MAE(Mean Absolute Error)
  - MAPE(Mean Absolute Percentage Error)
  - MSE(Mean Squared Error)
  - MPE(Mean Percentage Error)

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use(['seaborn-whitegrid'])
```

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

```
noise = np.random.rand(100, 1)
X = sorted(10 * np.random.rand(100, 1)) + noise
y = sorted(10 * np.random.rand(100))
```

```
plt.scatter(X, y);
```




```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model = LinearRegression()
model.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

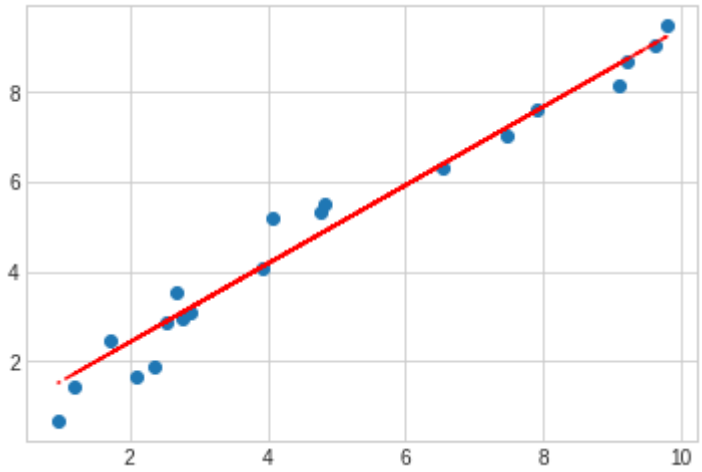
```
print("선형 회귀 가중치: {}".format(model.coef_))
print("선형 회귀 편향: {}".format(model.intercept_))
```

```
선형 회귀 가중치: [0.87746766]
선형 회귀 편향: 0.6601177129542171
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.9730525150106997
평가 데이터 점수: 0.9694304600667036
```

```
predict = model.predict(X_test)

plt.scatter(X_test, y_test)
plt.plot(X_test, predict, '--r');
```



보스턴 주택 가격 데이터

- 주택 가격 데이터는 도시에 대한 분석과 부동산, 경제적인 정보 분석 등 많은 활용 가능한 측면들이 존재
- 보스턴 주택 가격 데이터는 카네기 멜론 대학교에서 관리하는 StatLib 라이브러리에서 가져온 것
- 헤리슨(Harrison, D.)과 루빈펠트(Rubinfeld, D. L.)의 논문 "Hedonic prices and the demand for clean air', J. Environ. Economics & Management"에서 보스턴 데이터가 사용
- 1970년도 인구 조사에서 보스턴의 506개 조사 구역과 주택 가격에 영향을 주는 속성 21개로 구성

속성	설명
CRIM	자치시(town)별 1인당 범죄율
ZN	25,000 평방 피트가 넘는 거주지역 토지 비율
INDUS	자치시(town)별 비소매 상업지역 토지 비율
CHAS	찰스 강(Charles River)에 대한 변수 (강의 경계에 위치하면 1, 그렇지 않으면 0)
NOX	10,000,000당 일산화질소 농도
RM	주택 1가구당 평균 방의 수
AGE	1940년 이전에 건축된 소유주택 비율
DIS	5개의 보스턴 고용 센터까지의 가중 거리
RAD	방사형 고속도로 접근성 지수
TAX	10,000 달러당 재산 세율
PTRATIO	자치시(town)별 학생/교사 비율
B	1000(Bk-0.63)^2, Bk: 자치시별 흑인 비율
LSTAT	모집단의 하위계층 비율(%)
MEDV	소유자가 거주하는 주택가격(중앙값) (단위: 1,000 달러)

```
from sklearn.datasets import load_boston

boston = load_boston()
print(boston.keys())
print(boston.DESCR)
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
```

```
            - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
            - NOX      nitric oxides concentration (parts per 10 million)
            - RM       average number of rooms per dwelling
            - AGE      proportion of owner-occupied units built prior to 1940
            - DIS      weighted distances to five Boston employment centres
            - RAD      index of accessibility to radial highways
            - TAX      full-value property-tax rate per $10,000
            - PTRATIO  pupil-teacher ratio by town
            - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
            - LSTAT    % lower status of the population
            - MEDV     Median value of owner-occupied homes in $1000's

        :Missing Attribute Values: None

        :Creator: Harrison, D. and Rubinfeld, D.L.

    This is a copy of UCI ML housing dataset.
    https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


    This dataset was taken from the StatLib library which is maintained at Carnegie Mellon Univer

    The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
    prices and the demand for clean air', J. Environ. Economics & Management,
    vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
    ...', Wiley, 1980.   N.B. Various transformations are used in the table on
    pages 244-261 of the latter.

    The Boston house-price data has been used in many machine learning papers that address regres
    problems.

    .. topic:: References

       - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources
       - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on
```

```
import pandas as pd

boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
boston_df['MEDV'] = boston.target
boston_df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

```
boston_df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AG
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.5749
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.1488
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.9000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.0250
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.5000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.0750
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.0000

```
for i, col in enumerate(boston_df.columns):
  plt.figure(figsize=(8, 4))
  plt.plot(boston_df[col])
  plt.title(col)
  plt.xlabel('Town')
  plt.tight_layout()
```

```
for i, col in enumerate(boston_df.columns):
  plt.figure(figsize=(8, 4))
  plt.scatter(boston_df[col], boston_df['MEDV'])
  plt.ylabel('MEDV', size=12)
  plt.xlabel(col, size=12)
  plt.tight_layout()
```

```
import seaborn as sns

sns.pairplot(boston_df);
```

보스턴 주택 가격에 대한 선형 회귀

```
from sklearn.linear_model import LinearRegression

model = LinearRegression(normalize=True)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target, test_size=0.2)

model.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
print('평가 데이터 점수: {}'.format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7482623087655588
평가 데이터 점수: 0.6829792195509012
```

- 데이터를 두개로 분리하고 모델을 생성 및 검증하였지만, 데이터를 분리하였기 때문에 훈련에 사용할 수 있는 양도 작아지고, 분리가 잘 안된 경우에는 잘못된 검증이 될 수 있음
- 이럴 경우에는 테스트셋을 여러개로 구성하여 교차 검증을 진행
- `cross_val_score()` 함수는 교차 검증을 수행하여 모델을 검증
- 다음 예제에서는 모델 오류를 측정하는 점수로 NMSE(Negative Mean Squared Error)를 사용

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, boston.data, boston.target, cv=10, scoring='neg_mean_squared_error'
print("NMSE scores: {}".format(scores))
print("NMSE scores mean: {}".format(scores.mean()))
print("NMSE scores std: {}".format(scores.std()))
```

```
NMSE scores: [  -9.28694671  -14.15128316  -14.07360615  -35.20692433  -31.88511666
  -19.83587796   -9.94726918 -168.37537954  -33.32974507  -10.96041068]
NMSE scores mean: -34.70525594452485
NMSE scores std: 45.57399920030876
```

- 회귀모델의 검증을 위한 또 다른 측정 지표 중 하나로 결정 계수(coefficient of determination, R^2) 사용

```
r2_scores = cross_val_score(model, boston.data, boston.target, cv=10, scoring='r2')

print("R2 scores: {}".format(r2_scores))
print("R2 scores mean: {}".format(r2_scores.mean()))
print("R2 scores std: {}".format(r2_scores.std()))
```

```
R2 scores: [ 0.73376082  0.4730725  -1.00631454  0.64113984  0.54766046  0.73640292
  0.37828386 -0.12922703 -0.76843243  0.4189435 ]
R2 scores mean: 0.20252899006056363
R2 scores std: 0.5952960169512287
```

생성된 회귀 모델에 대해서 평가를 위해 LinearRegression 객체에 포함된 두 개의 속성 값을 통해 수식을 표현

- intercept_: 추정된 상수항
- coef_: 추정된 가중치 벡터

```
print('y = ' + str(model.intercept_) + ' ')
for i, c in enumerate(model.coef_):
  print(str(c) + ' * x' + str(i))
```

```
y = 40.151950673959114
-0.12569218138697888 * x0
0.05278091921625718 * x1
```

```
0.05519169712768707 * x2
3.259942844239003 * x3
-19.072854988873114 * x4
3.45561577414609 * x5
0.004643320238898417 * x6
-1.5404438013982902 * x7
0.35250048052693567 * x8
-0.012218498965201705 * x9
-0.9720525710088264 * x10
0.008879214127290019 * x11
-0.6170630928077566 * x12
```

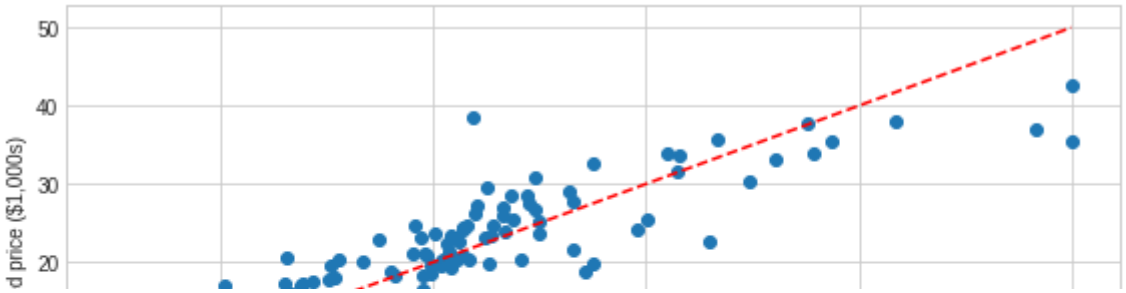
```
from sklearn.metrics import mean_squared_error, r2_score

y_train_predict = model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = r2_score(y_train, y_train_predict)

print('RMSE: {}'.format(rmse))
print('R2 Score: {}'.format(r2))
```

```
RMSE: 4.680276272641772
R2 Score: 0.7482623087655588
```

```
y_test_predict = model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
r2 = r2_score(y_test, y_test_predict)

print('RMSE: {}'.format(rmse))
print('R2 Score: {}'.format(r2))
```

```
RMSE: 4.825657263170694
R2 Score: 0.6829792195509012
```

```
def plot_boston_prices(expected, predicted):
  plt.figure(figsize=(8, 4))
  plt.scatter(expected, predicted)
  plt.plot([5, 50], [5, 50], '--r')
  plt.xlabel('True price ($1,000s)')
  plt.ylabel('Predicted price ($1,000s)')
  plt.tight_layout()

predicted = model.predict(X_test)
expected = y_test

plot_boston_prices(expected, predicted)
```



캘리포니아 주택 가격 데이터

속성	설명
MedInc	블록의 중간 소득
HouseAge	블록의 중간 주택 연도
AveRooms	평균 방 수
AveBedrms	평균 침실 수
Population	블록 내 거주중인 인구수
AveOccup	평균 주택점유율
Latitude	주택 블록 위도
Longitude	주택 블록 경도

```
from sklearn.datasets import fetch_california_housing

california = fetch_california_housing()
print(california.keys())
print(california.DESCR)
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR'])
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block
        - HouseAge      median house age in block
        - AveRooms      average number of rooms
        - AveBedrms     average number of bedrooms
        - Population    block population
        - AveOccup      average house occupancy
        - Latitude      house block latitude
        - Longitude     house block longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
http://lib.stat.cmu.edu/datasets/

The target variable is the median house value for California districts.
```

```
This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. topic:: References

    - Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
      Statistics and Probability Letters, 33 (1997) 291-297
```

```
import pandas as pd

california_df = pd.DataFrame(california.data, columns=california.feature_names)
california_df['Target'] = california.target
california_df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

```
california_df.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOc
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333

```
import matplotlib.pyplot as plt

for i, col in enumerate(california_df.columns):
  plt.figure(figsize=(8, 5))
```

```
  plt.plot(california_df[col])
  plt.title(col)
  plt.tight_layout()
```

```
for i, col in enumerate(california_df.columns):
  plt.figure(figsize=(8, 5))
  plt.scatter(california_df[col], california_df['Target'])
  plt.ylabel('Target', size=12)
  plt.xlabel(col, size=12)
  plt.tight_layout()
```

```
import seaborn as sns

sns.pairplot(california_df.sample(1000));
```

```
california_df.plot(kind='scatter', x='Longitude', y='Latitude', alpha=0.2, figsize=(12, 10));
```

```
california_df.plot(kind='scatter', x='Longitude', y='Latitude', alpha=0.2,
                   s=california_df['Population']/100, label='Population', figsize=(15, 10),
                   c='Target', cmap=plt.get_cmap('viridis'), colorbar=True);
```

캘리포니아 주택 가격에 대한 선형 회귀

```
model = LinearRegression(normalize=True)

X_train, X_test, y_train, y_test = train_test_split(california.data, california.target, test_size=0

model.fit(X_train, y_train)
print('학습 데이터 점수: {}'.format(model.score(X_train, y_train)))
print('평가 데이터 점수: {}'.format(model.score(X_test, y_test)))

scores = cross_val_score(model, california.data, california.target, cv=10, scoring='neg_mean_square
print('NMSE mean: {}'.format(scores.mean()))
print('NMSE std: {}'.format(scores.std()))

r2_scores = cross_val_score(model, california.data, california.target, cv=10, scoring='r2')
print('R2 Score mean: {}'.format(r2_scores.mean()))
```

```
학습 데이터 점수: 0.6030502949820096
평가 데이터 점수: 0.6186826267129517
NMSE mean: -0.5509524296956643
NMSE std: 0.19288582953865113
R2 Score mean: 0.5110068610523768
```

```
print('y = ' + str(model.intercept_) + ' ')
for i, c in enumerate(model.coef_):
  print(str(c) + ' * x' + str(i))
```

```
y = -37.15410780660094
0.43398521597345735 * x0
```

```
0.009704997150488259 * x1
-0.10170438395907236 * x2
0.6142372921671886 * x3
-3.5718620464048345e-06 * x4
-0.003485326205097975 * x5
-0.42486952546728746 * x6
-0.43743594148657466 * x7
```

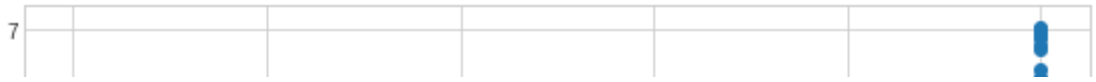
```
y_train_predict = model.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = r2_score(y_train, y_train_predict)

print('RMSE: {}'.format(rmse))
print('R2 Score: {}'.format(r2))
```

```
RMSE: 0.7311922016729673
R2 Score: 0.6030502949820096
```

```
y_test_predict = model.predict(X_test)
rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
r2 = r2_score(y_test, y_test_predict)

print('RMSE: {}'.format(rmse))
print('R2 Score: {}'.format(r2))
```

```
RMSE: 0.6953404831675986
R2 Score: 0.6186826267129517
```

```
def plot_california_prices(expected, predicted):
  plt.figure(figsize=(8, 4))
  plt.scatter(expected, predicted)
  plt.plot([0, 5], [0, 5], '--r')
  plt.xlabel('True price ($100,000s)')
  plt.ylabel('Predicted price ($100,000s)')
  plt.tight_layout()

predicted = model.predict(X_test)
expected = y_test

plot_california_prices(expected, predicted)
```



릿지 회귀(Ridge Regression)

- 릿지 회귀는 선형 회귀를 개선한 선형 모델
- 릿지 회귀는 선형 회귀와 비슷하지만, 가중치의 절대값을 최대한 작게 만든다는 것이 다름
- 이러한 방법은 각각의 특성(feature)이 출력 값에 주는 영향을 최소한으로 만들도록 규제(regularization)를 거는 것
- 규제를 사용하면 다중공선성(multicollinearity) 문제를 방지하기 때문에 모델의 과대적합을 막을 수 있게 됨
- 다중공선성 문제는 두 특성이 일치에 가까울 정도로 관련성(상관관계)이 높을 경우 발생
- 릿지 회귀는 다음과 같은 함수를 최소화 하는 파라미터 w를 찾음

$$RidgeMSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \alpha\sum_{i=1}^{p}w_i^2$$

- α: 사용자가 지정하는 매개변수
- α가 크면 규제의 효과가 커지고, α가 작으면 규제의 효과가 작아짐

보스턴 주택 가격에 대한 릿지 회귀

```
from sklearn.linear_model import Ridge
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
```

```
X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

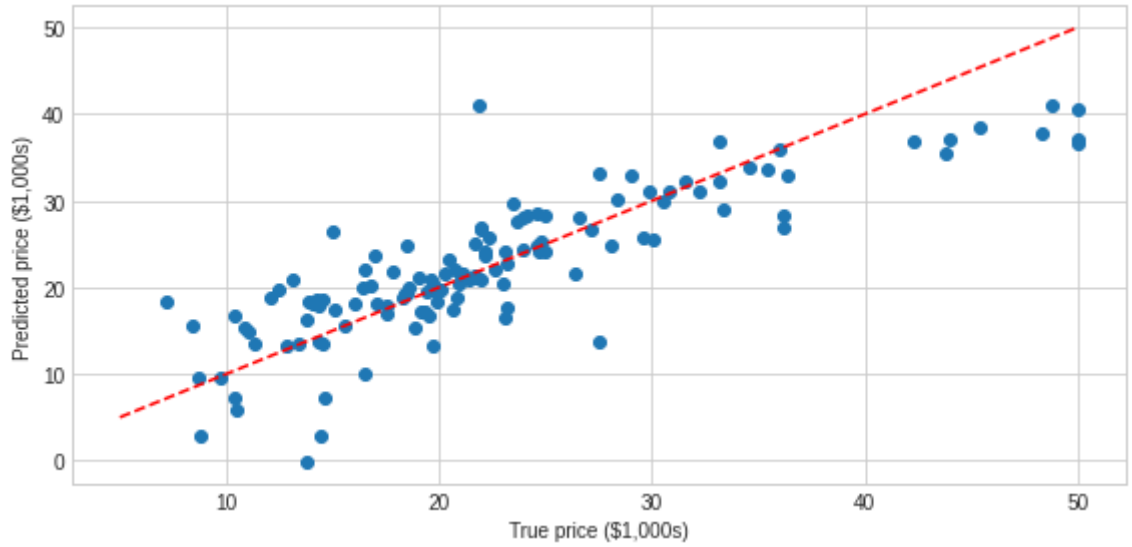
```
model = Ridge(alpha=0.2)
model.fit(X_train, y_train)
```

```
Ridge(alpha=0.2, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7465360751567834
평가 데이터 점수: 0.703181973999957
```

```
predicted = model.predict(X_test)
expected = y_test

plot_boston_prices(expected, predicted)
```



- 릿지 회귀는 가중치에 제약을 두기 때문에 선형 회귀 모델보다 훈련 데이터 점수가 낮을 수 있음
- 일반화 성능은 릿지 회귀가 더 높기 때문에 평가 데이터 점수는 릿지 회귀가 더 좋음
- 일반화 성능에 영향을 주는 매개 변수인 α 값을 조정해 보면서 릿지 회귀 분석의 성능이 어떻게 변하는지 확인 필요

캘리포니아 주택 가격에 대한 릿지 회귀

```
from sklearn.datasets import fetch_california_housing

california = fetch_california_housing()

X_train, X_test, y_train, y_test = train_test_split(california.data, california.target, test_size=0
```

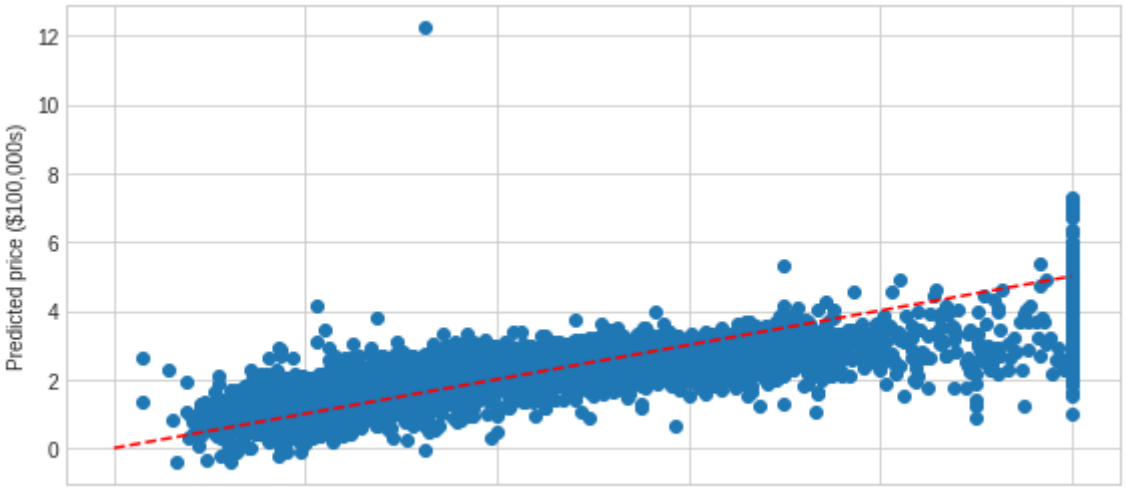
```
model = Ridge(alpha=0.1)
model.fit(X_train, y_train)
```

```
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.6132269432190609
평가 데이터 점수: 0.5701715569408305
```

```
predicted = model.predict(X_test)
expected = y_test

plot_california_prices(expected, predicted)
```



라쏘 회귀(Lasso Regression)

- 선형 회귀에 규제를 적용한 또 다른 모델로 라쏘 회귀가 있음
- 라쏘 회귀는 릿지 회귀와 비슷하게 가중치를 0에 가깝게 만들지만, 조금 다른 방식을 사용
- 라쏘 회귀에서는 다음과 같은 함수를 최소화 하는 파라미터 w를 찾음

$$LassoMSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \alpha\sum_{i=1}^{p}|w_i|$$

- 라쏘 회귀도 매개변수인 α 값을 통해 규제의 강도 조절 가능

보스턴 주택 가격에 대한 라쏘 회귀

```
from sklearn.linear_model import Lasso
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
```

```
X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
model = Lasso(alpha=0.001)
model.fit(X_train, y_train)
```

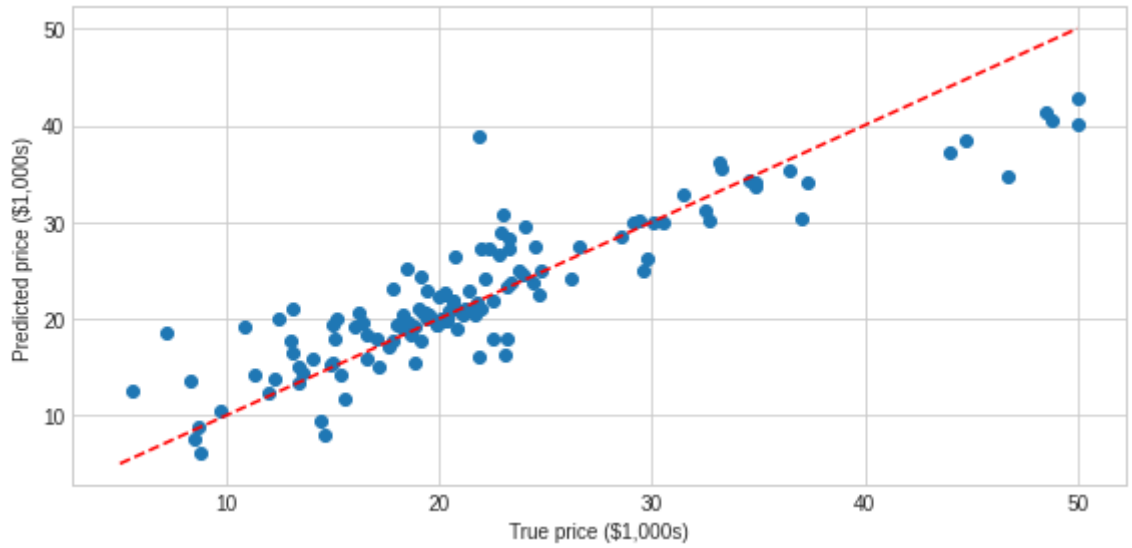
```
Lasso(alpha=0.001, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7217424795484668
평가 데이터 점수: 0.7887478546987542
```

```
predicted = model.predict(X_test)
```

```
expected = y_test

plot_boston_prices(expected, predicted)
```



캘리포니아 주택 가격에 대한 라쏘 회귀

```
from sklearn.datasets import fetch_california_housing

california = fetch_california_housing()

X_train, X_test, y_train, y_test = train_test_split(california.data, california.target, test_size=0
```

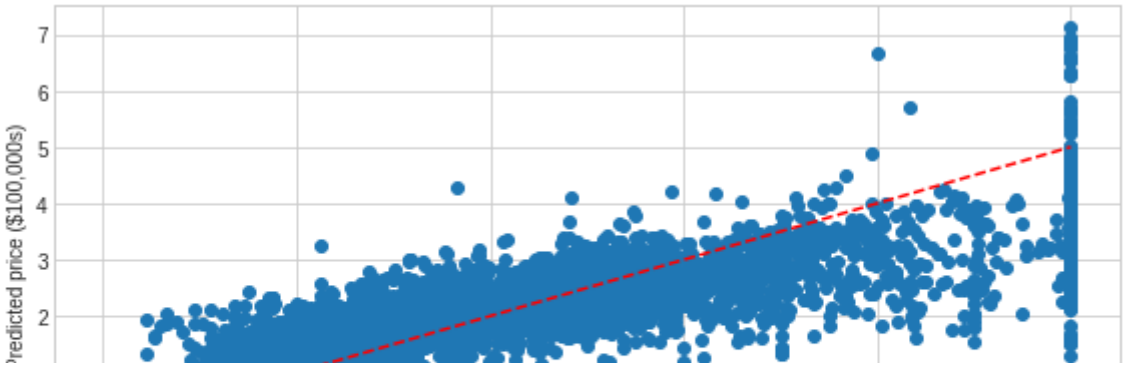
```
model = Lasso(alpha=0.001)
model.fit(X_train, y_train)
```

```
Lasso(alpha=0.001, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.6074193037326745
평가 데이터 점수: 0.6003182663534099
```

```
predicted = model.predict(X_test)
expected = y_test

plot_california_prices(expected, predicted)
```



신축망 (Elastic-Net)

- 신축망은 릿지 회귀와 라쏘 회귀, 두 모델의 모든 규제를 사용하는 선형 모델
- 두 모델의 장점을 모두 갖고 있기 때문에 좋은 성능을 보임
- 데이터 특성이 많거나 서로 상관 관계가 높은 특성이 존재 할 때 위의 두 모델보다 좋은 성능을 보여 줌
- 신축망은 다음과 같은 함수를 최소화 하는 파라미터 w를 찾음

$$ElasticMSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i) + \alpha\rho\sum_{i=1}^{p}|w_i| + \alpha(1-\rho)\sum_{i=1}^{p}w_i^2$$

- α: 규제의 강도를 조절하는 매개변수
- ρ: 라쏘 규제와 릿지 규제 사이의 가중치를 조절하는 매개변수

보스턴 주택 가격에 대한 신축망

```
from sklearn.linear_model import ElasticNet
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
```

```
X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

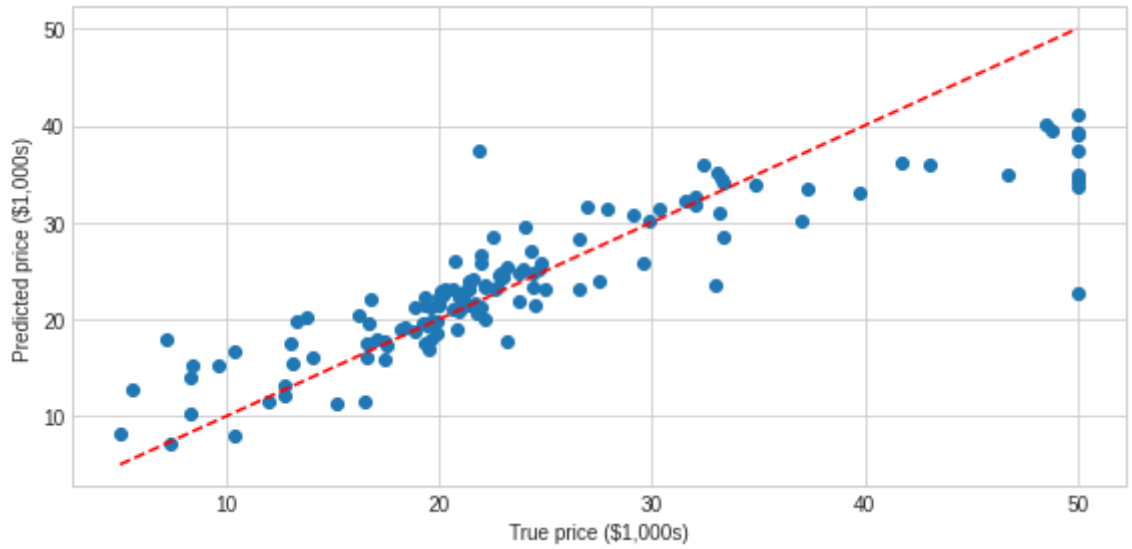
```
model = ElasticNet(alpha=0.01, l1_ratio=0.5)
model.fit(X_train, y_train)
```

```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7244369382924891
평가 데이터 점수: 0.7298219236333228
```

```
predicted = model.predict(X_test)
expected = y_test

plot_boston_prices(expected, predicted)
```



캘리포니아 주택 가격에 대한 신축망

```
from sklearn.datasets import fetch_california_housing

california = fetch_california_housing()

X_train, X_test, y_train, y_test = train_test_split(california.data, california.target, test_size=0
```

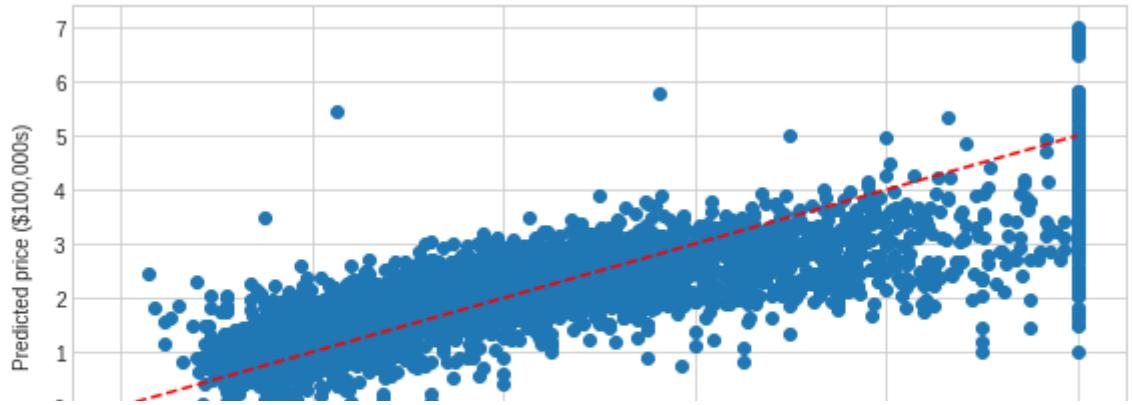
```
model = ElasticNet(alpha=0.01, l1_ratio=0.5)
model.fit(X_train, y_train)
```

```
ElasticNet(alpha=0.01, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.6035911312557962
평가 데이터 점수: 0.6078513369923881
```

```
predicted = model.predict(X_test)
expected = y_test

plot_california_prices(expected, predicted)
```



직교 정합 추구 (Orthogonal Matchin Pursuit)

- 직교 정합 추구 방법은 모델에 존재하는 가중치 벡터에 특별한 제약을 거는 방법
- 직교 정합 추구 방법은 다음을 만족하는 파라미터 w를 찾는것이 목표

$$\arg\min_{w} ||y - \hat{y}||_2^2 \; subject\ to \; ||w||_0 \leq k$$

- $||w||_0$: 가중치 벡터 w에서 0이 아닌 값의 개수
- 직교 정합 추구 방법은 가중치 벡터 w에서 0이 아닌 값이 k개 이하가 되도록 훈련됨
- 이러한 방법은 모델이 필요 없는 데이터 특성을 훈련 과정에서 자동으로 제거 하도록 만들 수 있음

보스턴 주택 가격에 대한 직교 정합 추구

```
from sklearn.linear_model import OrthogonalMatchingPursuit
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
```

```
X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
model = OrthogonalMatchingPursuit(n_nonzero_coefs=7)
model.fit(X_train, y_train)
```

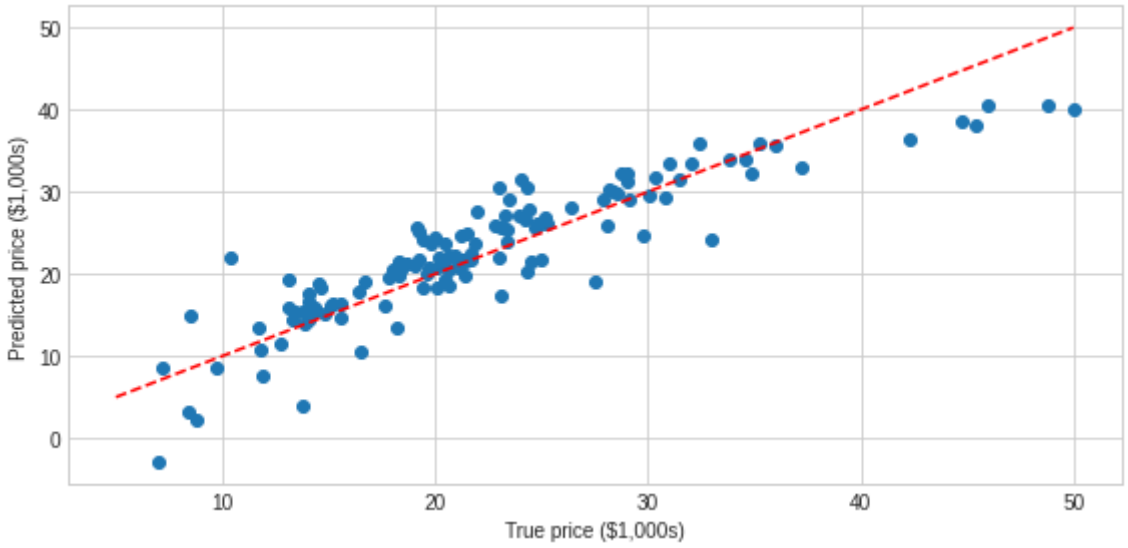
```
OrthogonalMatchingPursuit(fit_intercept=True, n_nonzero_coefs=7, normalize=True,
                          precompute='auto', tol=None)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7000129949756899
평가 데이터 점수: 0.7958342055548993
```

```
predicted = model.predict(X_test)
expected = y_test
```

```
plot_boston_prices(expected, predicted)
```



- 직교 정합 추구 방법은 위에서 설명한 제약 조건 대신에 다음 조건을 만족하도록 변경 가능

$$\arg\min_{w} ||w||_0 \ subject\ to\ ||y - \hat{y}||_2^2 \leq tol$$

- $||y - \hat{y}||_2^2$는 $\sum_{i=1}^{N}(y - \hat{y})^2$와 같은 의미
- 위의 식을 통해서 직교 정합 추구 방법을 y와 $\hat{y}$ 사이의 오차 제곱 합을 tol 이하로 하면서 $||w||_0$를 최소로 하는 모델로 대체 가능

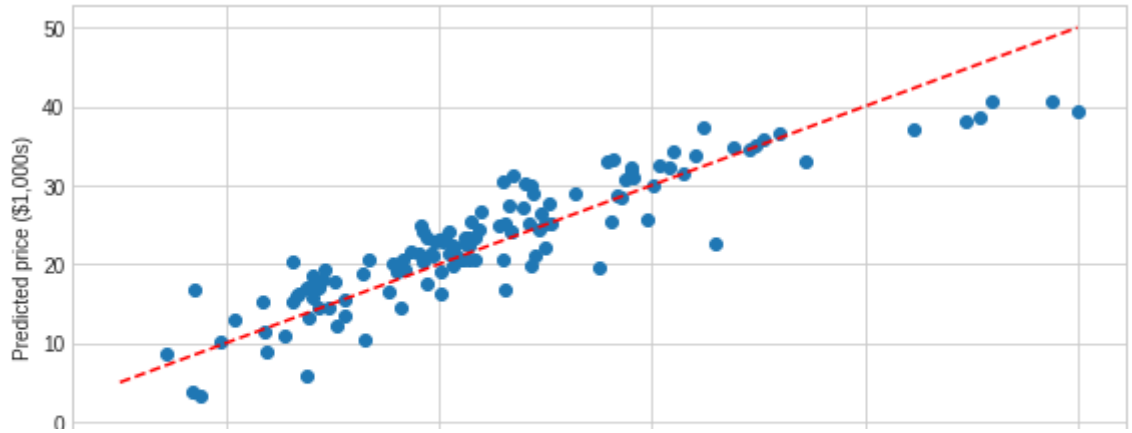
```
model = OrthogonalMatchingPursuit(tol=1.)
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_omp.py:673: RuntimeWarning:  Ort
dependence in the dictionary. The requested precision might not have been met.

  return_n_iter=True)
OrthogonalMatchingPursuit(fit_intercept=True, n_nonzero_coefs=None,
                          normalize=True, precompute='auto', tol=1.0)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.7230758734324543
평가 데이터 점수: 0.79345937200776
```

```
predicted = model.predict(X_test)
expected = y_test

plot_boston_prices(expected, predicted)
```



캘리포니아 주택 가격에 대한 직교 정합 추구

```
from sklearn.datasets import fetch_california_housing

california = fetch_california_housing()

X_train, X_test, y_train, y_test = train_test_split(california.data, california.target, test_size=0
```

```
model = OrthogonalMatchingPursuit(n_nonzero_coefs=5)
model.fit(X_train, y_train)
```

```
OrthogonalMatchingPursuit(fit_intercept=True, n_nonzero_coefs=5, normalize=True,
                          precompute='auto', tol=None)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.5952693437915713
평가 데이터 점수: 0.609268230841713
```

```
predicted = model.predict(X_test)
expected = y_test

plot_california_prices(expected, predicted)
```

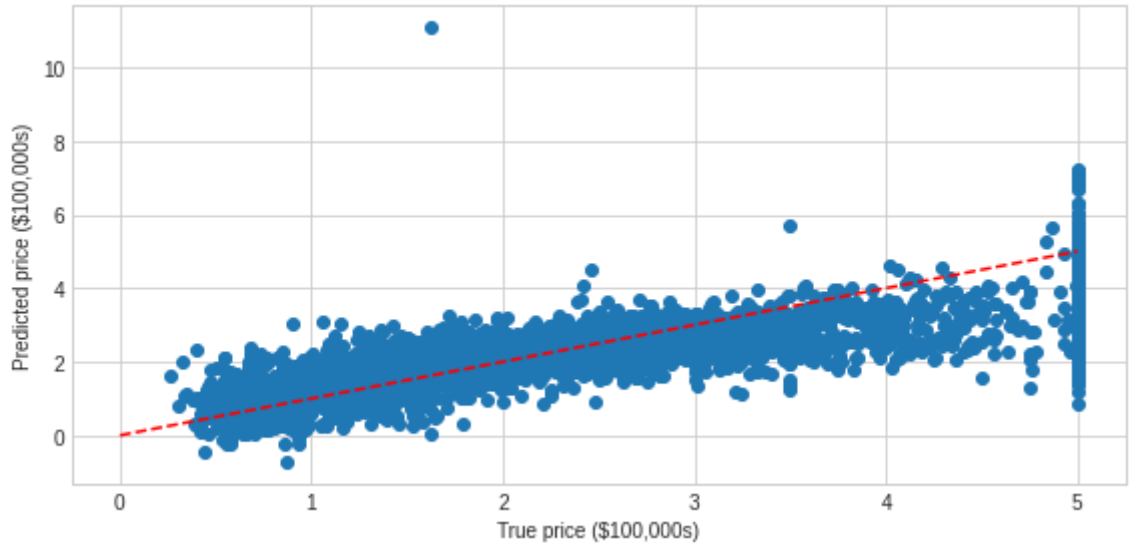
```
model = OrthogonalMatchingPursuit(tol=1.)
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_omp.py:673: RuntimeWarning:  Ort
dependence in the dictionary. The requested precision might not have been met.

  return_n_iter=True)
OrthogonalMatchingPursuit(fit_intercept=True, n_nonzero_coefs=None,
                          normalize=True, precompute='auto', tol=1.0)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.6039635201185489
평가 데이터 점수: 0.6117176753541198
```

```
predicted = model.predict(X_test)
expected = y_test

plot_california_prices(expected, predicted)
```

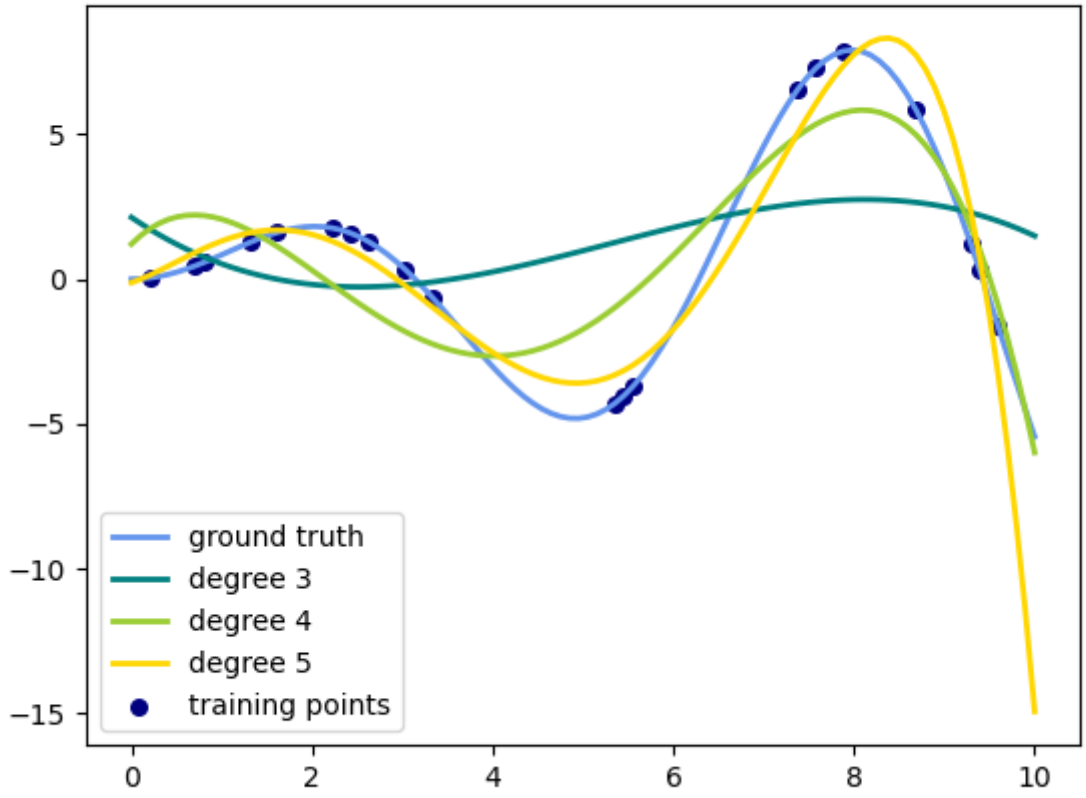


다항 회귀 (Polynomial Regression)

- 입력 데이터를 비선형 변환 후 사용하는 방법
- 모델 자체는 선형 모델

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_1^2 + w_5 x_2^2$$

- 차수가 높아질수록 더 복잡한 데이터 학습 가능



보스턴 주택 가격에 대한 다항 회귀

```
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_boston
```

```
X, y = load_boston(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=123)
```

```
model = make_pipeline(
    PolynomialFeatures(degree=2),
    StandardScaler(),
    LinearRegression()
)
model.fit(X_train, y_train)
```

```
Pipeline(memory=None,
         steps=[('polynomialfeatures',
                 PolynomialFeatures(degree=2, include_bias=True,
                                    interaction_only=False, order='C')),
                ('standardscaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
```

```
                ('linearregression',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.9346787783950696
평가 데이터 점수: 0.825786471800239
```

```
predicted = model.predict(X_test)
expected = y_test

plot_boston_prices(expected, predicted)
```

캘리포니아 주택 가격에 대한 다항 회귀

```
from sklearn.datasets import fetch_california_housing

california = fetch_california_housing()

X_train, X_test, y_train, y_test = train_test_split(california.data, california.target, test_size=0
```

```
model = make_pipeline(
    PolynomialFeatures(degree=2),
    StandardScaler(),
    LinearRegression()
)
model.fit(X_train, y_train)
```

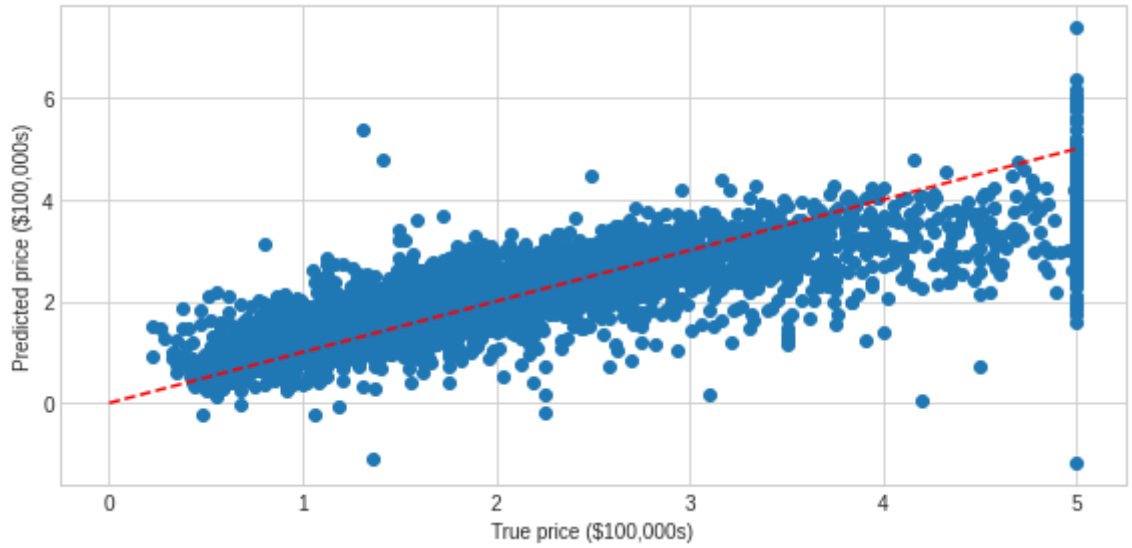
```
Pipeline(memory=None,
         steps=[('polynomialfeatures',
                 PolynomialFeatures(degree=2, include_bias=True,
                                    interaction_only=False, order='C')),
                ('standardscaler',
```

```
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('linearregression',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                                  normalize=False))],
         verbose=False)
```

```
print("학습 데이터 점수: {}".format(model.score(X_train, y_train)))
print("평가 데이터 점수: {}".format(model.score(X_test, y_test)))
```

```
학습 데이터 점수: 0.6863138436064128
평가 데이터 점수: 0.6680010961503731
```

```
predicted = model.predict(X_test)
expected = y_test

plot_california_prices(expected, predicted)
```



참고문헌

- scikit-learn 사이트: https://scikit-learn.org/
- Jake VanderPlas, "Python Data Science Handbook", O'Reilly
- Sebastian Raschka, Vahid Mirjalili, "Python Machine Learning", Packt
- Giuseppe Bonaccorso, "Machine Learning Algorithm", Packt
- Aurelien Geron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems", O'Reilly