ch03 선형모델 - linear model

- 선형회귀(linear regression)는 100여 년 전에 개발되었다.
- 선형 모델은 입력 특성에 대한 선형 함수를 만들어 예측을 수행
- 특성이 하나일 때는 **직선**, 두개일 때는 **평면**, 더 높은 차원 **초평면(hyperplane)**
- knnRegressor과 비교해 보면 직선이 사용한 예측이 더 제약이 있음.
- 특성이 많은 데이터 셋이라면 선형 모델은 훌륭한 성능을 갖는다.

In [1]:

```
from IPython.display import display, Image
```

In [2]:

```
### 한글 폰트 설정
import matplotlib
from matplotlib import font_manager, rc
import matplotlib.pyplot as plt
import platform

path = "C:/Windows/Fonts/malgun.ttf"
if platform.system() == "Windows":
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc('font', family=font_name)
elif platform.system()=="Darwin":
    rc('font', family='AppleGothic')
else:
    print("Unknown System")

matplotlib.rcParams['axes.unicode_minus'] = False

%matplotlib inline
```

In [3]:

```
display(Image(filename='img/linear_model01.png'))
```

$$\hat{y} = w1*x1 + w2*x2 + \ldots + wp*xp + b$$

- x1~xp는 데이터 포인트에 대한 특성(feature)
- w와 b는 모델이 학습할 파라미터
- y^ = w1 * x1 + b 는 특성(feature) 하나를 선택한 모델

- 선형회귀 또는 최소제곱법(OLS)은 가장 간단하고 오래된 회귀용 선형 알고리즘.
- 선형 회귀는 예측과 훈련 세트에 있는 **타깃 y사이의 평균제곱오차(mean squared error)를 최소화하는 파라미터 w와 b**를 찾는다.
- 평균 제곱 오차는 예측값과 타깃값의 차이를 제곱하여 더한 후에 샘플의 개수로 나눈 것.

In [4]:

```
display(Image(filename='img/linear_model02_mse.png'))
```

$$\text{MSE(평균제곱오차)} = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

In [5]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
print(np.__version__)
print(matplotlib.__version__)
```

```
1.19.2
3.3.2
```

- mglearn은 numpy 1.16를 필요함.

In [6]:

```
# 설치가 안되어 있을 경우, 설치 필요.
import mglearn
import sklearn
print( sklearn.__version__)
print( mglearn.__version__)

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

```
0.23.2
0.1.9
```

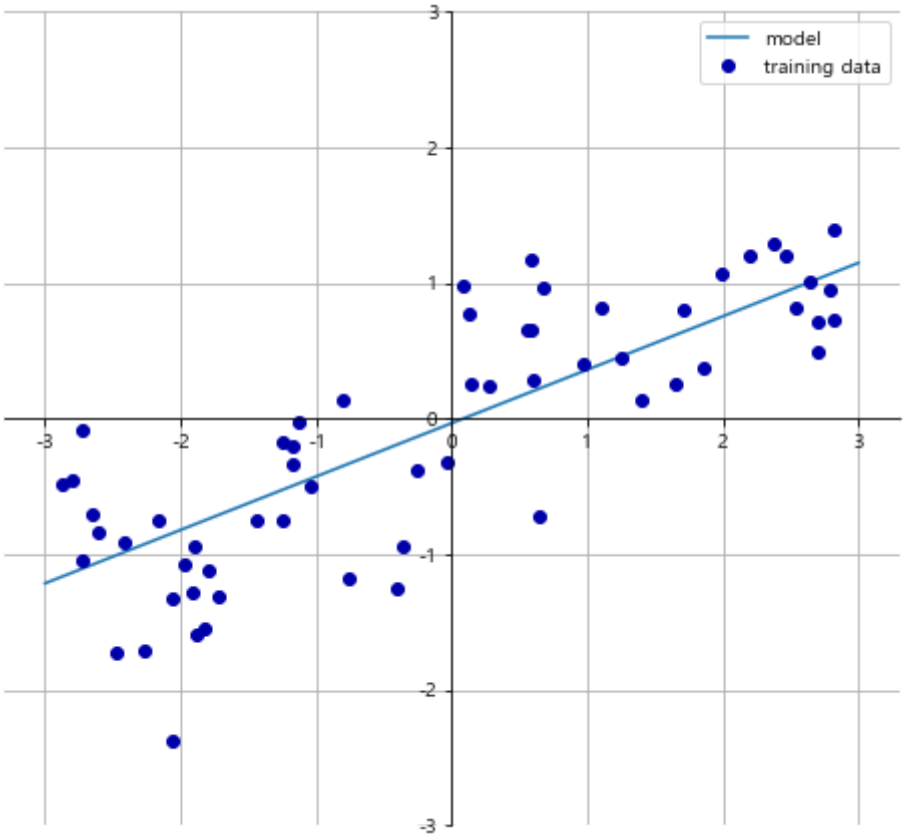
01 회귀 선형 모델 그래프로 살펴보기

특성이 하나일 때의 선형 함수

In [7]:

```
mglearn.plots.plot_linear_regression_wave()
```

```
w[0]: 0.393906  b: -0.031804
```



02 Boston 데이터 셋을 활용한 회귀 모델 만들어보기

데이터 설명

- 1970년대의 보스턴 주변의 주택 평균 가격 예측
- 506개의 데이터 포인트와 13개의 특성

(1) 모델 만들기 [모델명 = 모델객체()]
(2) 모델 학습 시키기 [모델명.fit()]
(3) 모델을 활용한 예측하기 [모델명.predict()]
(4) 모델 평가

In [8]:

```
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
```

In [9]:

```
boston = load_boston()
X = boston.data       # 입력 데이터  - 문제
y = boston.target     # 출력 데이터  - 답
```

데이터 살펴보기

In [12]:

```
print( boston.keys() )
print( boston.feature_names )
```

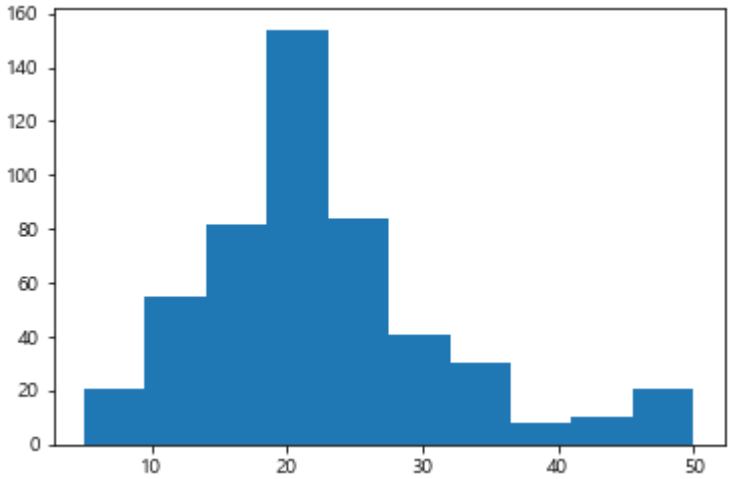
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [10]:

```
plt.hist(y)
```

Out[10]:

```
(array([ 21.,  55.,  82., 154.,  84.,  41.,  30.,   8.,  10.,  21.]),
 array([ 5. ,  9.5, 14. , 18.5, 23. , 27.5, 32. , 36.5, 41. , 45.5, 50. ]),
 <BarContainer object of 10 artists>)
```



- (실습) DataFrame으로 만들어 기본 시각화 등을 통해 확인해 보자.

데이터 준비하기

In [14]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=42)
```

In [15]:

```
model = LinearRegression().fit(X_train, y_train)   # 학습
pred = model.predict(X_test)
pred
```

Out[15]:

```
array([28.83885359, 36.00783288, 15.08324755, 25.23090886, 18.87864064,
       23.21398327, 17.5931124 , 14.30508093, 23.05438985, 20.62008346,
       24.78514683, 18.66833668, -6.9788951 , 21.83575737, 19.20898992,
       26.2868054 , 20.54379176,  5.65713224, 40.42358065, 17.64146116,
       27.32258958, 30.05056174, 11.15013704, 24.11530393, 17.89145648,
       15.79348591, 22.94743453, 14.2586068 , 22.26731194, 19.24709013,
       22.26897546, 25.24344002, 25.69165643, 17.98759507, 16.70286649,
       17.11631225, 31.19643534, 20.17835831, 23.71828436, 24.79196868,
       13.94575895, 32.00389982, 42.53869791, 17.44523722, 27.15354457,
       17.07482215, 13.89272021, 26.06440323, 20.36888769, 29.97813037,
       21.35346608, 34.32287916, 15.88498671, 26.17757739, 39.50970314,
       22.84123308, 18.95049088, 32.68913818, 25.02057949, 12.90539147,
       22.76052302, 30.53884316, 31.60797905, 15.92162168, 20.50670563,
       16.50798147, 20.50202198, 26.00723901, 30.63860954, 11.42877835,
       20.53765181, 27.56249175, 10.85162601, 15.96871769, 23.87570192,
        5.66369672, 21.47818991, 41.2820034 , 18.56559986,  9.08857252,
       20.97848452, 13.0630057 , 20.99054395,  9.34050291, 23.13686588,
       31.80106627, 19.10245917, 25.59186169, 29.14490119, 20.17571514,
       25.5962149 ,  5.20301905, 20.16835681, 15.08546746, 12.8601543 ,
       20.80904894, 24.68556943, -0.77450939, 13.33875673, 15.62703156,
       22.21755358, 24.58188737, 10.77302163, 19.50068376, 23.23450396,
       11.77388822, 18.36777924, 25.4383785 , 20.89079232, 24.08440617,
        7.3658717 , 19.16424347, 21.93734133, 27.41191713, 32.50857196,
       14.86885244, 35.05912525, 12.86075113, 20.83043572, 28.42077138,
       15.65853688, 24.67196362,  3.28420892, 23.79879617, 25.73329894,
       23.04815612, 24.73046824])
```

In [16]:

```
import pandas as pd
```

In [17]:

```
dict_dat = {"실제값":y_test, "예측값":pred, "오차":y_test - pred}
dat = pd.DataFrame(dict_dat )
dat
```

Out[17]:

	실제값	예측값	오차
0	23.6	28.838854	-5.238854
1	32.4	36.007833	-3.607833
2	13.6	15.083248	-1.483248
3	22.8	25.230909	-2.430909
4	16.1	18.878641	-2.778641
...	...	...	...
122	8.8	3.284209	5.515791
123	19.2	23.798796	-4.598796
124	25.3	25.733299	-0.433299
125	20.4	23.048156	-2.648156
126	23.1	24.730468	-1.630468

127 rows × 3 columns

In [18]:

```
dat['오차절대값'] = abs(dat['오차'])
dat['오차제곱'] = dat['오차'] ** (2)
dat
```

Out[18]:

	실제값	예측값	오차	오차절대값	오차제곱
0	23.6	28.838854	-5.238854	5.238854	27.445587
1	32.4	36.007833	-3.607833	3.607833	13.016458
2	13.6	15.083248	-1.483248	1.483248	2.200023
3	22.8	25.230909	-2.430909	2.430909	5.909318
4	16.1	18.878641	-2.778641	2.778641	7.720844
...	...	...	...	...	...
122	8.8	3.284209	5.515791	5.515791	30.423951
123	19.2	23.798796	-4.598796	4.598796	21.148926
124	25.3	25.733299	-0.433299	0.433299	0.187748
125	20.4	23.048156	-2.648156	2.648156	7.012731
126	23.1	24.730468	-1.630468	1.630468	2.658427

127 rows × 5 columns

평가 지표

- MAE(mean absolute error)
- MSE(mean squared error)
- RMSE(root mean squared error)

MAE (mean absolute error)

- 각각의 값에 절대값을 취한다. 이를 전부 더한 후, 갯수로 나누어주기

In [19]:

```
### MSE, MAE, RMSE, RMLSE
sum(dat['오차절대값'])/len(dat['오차절대값'])
```

Out[19]:

3.060939595437035

In [20]:

```
np.mean(dat['오차절대값'])
```

Out[20]:

3.060939595437035

MSE (mean squared error)

- (실제값-예측값) ^ 2 의 합를 데이터의 샘플의 개수로 나누어준것

In [21]:

```
value = np.mean(dat['오차제곱'])
value
```

Out[21]:

22.098694827098043

In [22]:

```
mse_value = sum(dat['오차'] ** 2) / len(dat['오차'])
mse_value
```

Out[22]:

22.098694827098043

In [23]:

```
from sklearn.metrics import mean_squared_error
```

In [24]:

```
mean_squared_error(y_test, pred)
```

Out[24]:

22.098694827098036

RMSE (root mean squared error)

- (실제값-예측값) ^ 2 의 합을 데이터의 샘플의 개수로 나누어 준 이후에 제곱근 씌우기

In [25]:

```
# (1) 제곱에 루트를 씌워구하기  (2) 제곱한 값을 길이로 나누기
rmse = np.sqrt(mse_value)
# rmse = mse_value ** 0.5  # 다른 방법
print(rmse)
```

4.700924890603767

결정계수

- 결정계수는 회귀모델에서 모델의 적합도를 의미하는 것으로 0~1사이의 값을 갖는다.
- 1에 가까우면 가까울수록 이 모델은 좋다라고 볼수 있다.

In [26]:

```
# R^2의 값을 구하기- 결정계수 구하기
print("훈련 데이터 세트 점수 : {:.2f}".format(model.score(X_train, y_train)))
print("테스트 데이터 세트 점수 : {:.2f}".format(model.score(X_test, y_test)))
```

```
훈련 데이터 세트 점수 : 0.75
테스트 데이터 세트 점수 : 0.68
```

In [27]:

```
for i in range(1, 6, 1):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=(i/10), random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    pred[:5]

    mae = np.abs(y_test - pred).sum() / len(pred)
    mse = ((y_test - pred)**2).sum()/len(pred)
    rmse = (((y_test - pred)**2).sum()/len(pred))**0.5

    print("test_size : ",(i/10))
    print("MAE : {:.3f}".format(mae))
    print("MSE : {:.3f}".format(mse))
    print("RMSE : {:.3f}".format(rmse))
    print("")
```

```
test_size :  0.1
MAE : 2.834
MSE : 14.996
RMSE : 3.872

test_size :  0.2
MAE : 3.189
MSE : 24.291
RMSE : 4.929

test_size :  0.3
MAE : 3.163
MSE : 21.517
RMSE : 4.639

test_size :  0.4
MAE : 3.298
MSE : 21.833
RMSE : 4.673

test_size :  0.5
MAE : 3.398
MSE : 25.175
RMSE : 5.018
```

실습과제1

- 데이터를 나누는 것에 따라 RMSE는 어떻게 되는지 확인해 보자.
  - 70:30, 90:10, 80:20, 75:25 등

실습 과제 1

- 아래 대회에서 데이터 셋을 다운로드 후, 다중선형 회귀 모델을 만들어보자.

    - URL : https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr/data (https://www.kaggle.com/c/2019-2nd-ml-month-with-kakr/data)
- MAE, MSE, RMSE를 구해보자

도전 ¶

- 나만의 데이터 셋을 선택하여 다중 선형 회귀 모델을 만들고 이를 예측을 수행한 후, 제출해 보자.