ch03 선형모델 - linear model

학습 목표

- 선형 모델(Linear Regression)에 대해 이해합니다.

학습 내용

- Boston 데이터 셋 불러오기
- 집값 예측 선형모델 구축해 보기

- 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
```

matplotlib의 한글 폰트 설정

In [15]:

```
### 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 [16]:

```
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 [17]:

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

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

In [18]:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
print("numpy 버전 : ", np.__version__)
print("matplotlib 버전 : ", matplotlib.__version__)
```

```
numpy 버전 :  1.22.3
matplotlib 버전 :  3.3.2
```

In [19]:

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

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

```
sklearn 버전 :  0.23.2
mglearn 버전 :  0.1.9
```

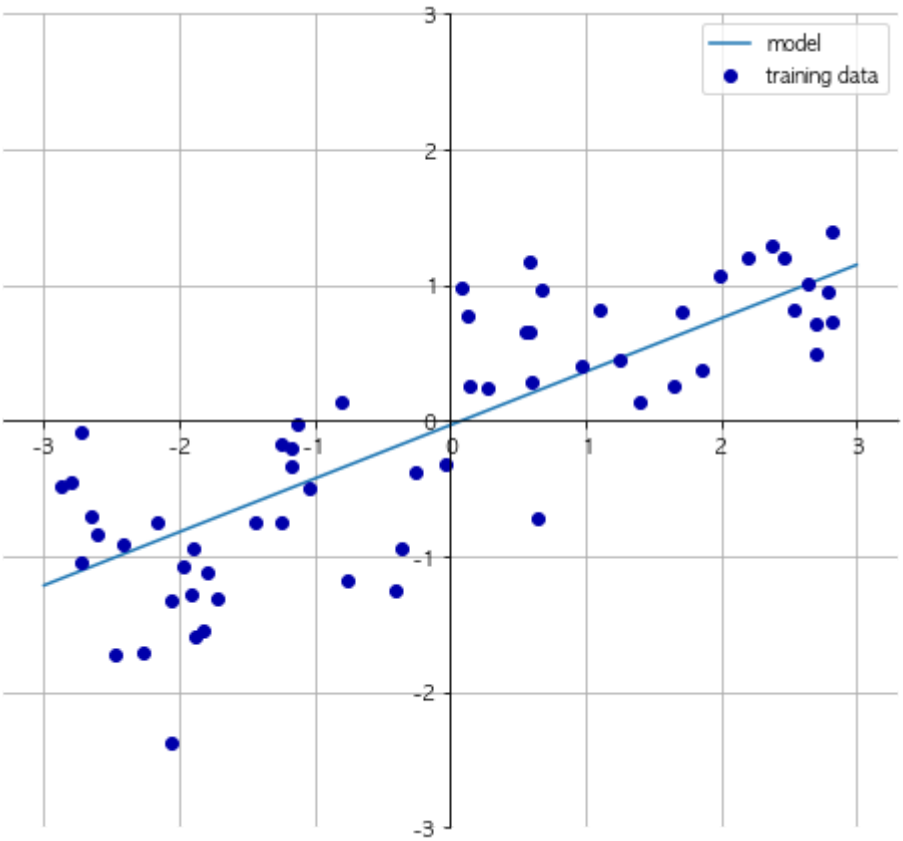
01 회귀 선형 모델 그래프로 살펴보기

특성이 하나일 때의 선형 함수

In [20]:

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

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



- [그래프 설명] w는 기울기, b는 y절편을 의미.

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 [21]:

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

In [22]:

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

데이터 살펴보기

features	내용	값
crim	마을별 1인당 범죄율	-
zn	25,000 평방 피트 이상의 대형 주택이 차지하는 주거용 토지의 비율	-
indus	소매상 이외의 상업 지구의 면적 비율	-
chas	Charles River(찰스강 접한 지역인지 아닌지) (강 경계면=1, 아니면=0	-
nox	산화 질소 오염도(1000만분 율)	-
rm	주거 당 평균 방수	-
age	1940년 이전에 지어진 소유주 집들의 비율	-
dis	보스턴 고용 센터 5곳까지의 가중 거리	-
rad	도시 순환 고속도로에의 접근 용이 지수	-
tax	만 달러당 주택 재산세율	-
ptratio	학생 - 선생 비율	-
black-(B)	흑인 인구 비율(Bk)이 지역 평균인 0.63과 다른 정도의 제곱	-
lstat	저소득 주민들의 비율 퍼센트	-
(target) MEDV	소유주가 거주하는 주택의 중간 가치($ 1000)	-

In [25]:

```
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' 'PTRATI
O'
 'B' 'LSTAT']
```

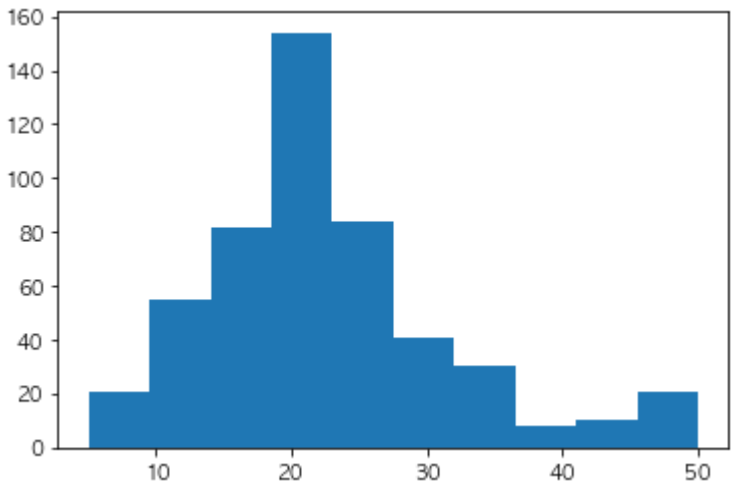
주택 가격 - 히스토 그램

In [27]:

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

Out[27]:

```
(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, 5
0. ]),
 <BarContainer object of 10 artists>)
```



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

데이터 준비하기

In [35]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=42)
```

In [36]:

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

Out[36]:

```
array([28.64896005, 36.49501384, 15.4111932 , 25.40321303, 18.85527
988,
       23.14668944, 17.3921241 , 14.07859899, 23.03692679, 20.59943
345,
       24.82286159, 18.53057049, -6.86543527, 21.80172334, 19.22571
177,
       26.19191985, 20.27733882,  5.61596432, 40.44887974, 17.57695
918,
       27.44319095, 30.1715964 , 10.94055823, 24.02083139, 18.07693
812,
       15.934748  , 23.12614028, 14.56052142, 22.33482544, 19.32576
27 ,
       22.16564973, 25.19476081, 25.31372473, 18.51345025, 16.62232
86 ,
       17.50268505, 30.94992991, 20.19201752, 23.90440431, 24.86975
466,
       13.93767876, 31.82504715, 42.56978796, 17.62323805, 27.01963
242,
       17.19006621, 13.80594006, 26.10356557, 20.31516118, 30.08649
576,
       21.3124053 , 34.15739602, 15.60444981, 26.11247588, 39.31613
646,
       22.99282065, 18.95764781, 33.05555669, 24.85114223, 12.91729
352,
       22.68101452, 30.80336295, 31.63522027, 16.29833689, 21.07379
993,
       16.57699669, 20.36362023, 26.15615896, 31.06833034, 11.98679
953,
       20.42550472, 27.55676301, 10.94316981, 16.82660609, 23.92909
733,
        5.28065815, 21.43504661, 41.33684993, 18.22211675,  9.48269
245,
       21.19857446, 12.95001331, 21.64822797,  9.3845568 , 23.06060
014,
       31.95762512, 19.16662892, 25.59942257, 29.35043558, 20.13138
581,
       25.57297369,  5.42970803, 20.23169356, 15.1949595 , 14.03241
742,
       20.91078077, 24.82249135, -0.47712079, 13.70520524, 15.69525
576,
       22.06972676, 24.64152943, 10.7382866 , 19.68622564, 23.63678
009,
       12.07974981, 18.47894211, 25.52713393, 20.93461307, 24.69559
41 ,
        7.59054562, 19.01046053, 21.9444339 , 27.22319977, 32.18608
828,
       15.27826455, 34.39190421, 12.96314168, 21.01681316, 28.57880
911,
       15.86300844, 24.85124135,  3.37937111, 23.90465773, 25.81792
146,
       23.11020547, 25.33489201, 33.35545176, 20.60724498, 38.47726
65 ,
```

```
       13.97398533, 25.21923987, 17.80946626, 20.63437371,  9.80267
398,
       21.07953576, 22.3378417 , 32.32381854, 31.48694863, 15.46621
287,
       16.86242766, 28.99330526, 24.95467894, 16.73633557,  6.12858
395,
       26.65990044, 23.34007187, 17.40367164, 13.38594123, 39.98342
478,
       16.68286302, 18.28561759])
```

In [37]:

```
import pandas as pd
```

In [38]:

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

Out[38]:

	실제값	예측값	오차
0	23.6	28.648960	-5.048960
1	32.4	36.495014	-4.095014
2	13.6	15.411193	-1.811193
3	22.8	25.403213	-2.603213
4	16.1	18.855280	-2.755280
...	...	...	...
147	17.1	17.403672	-0.303672
148	14.5	13.385941	1.114059
149	50.0	39.983425	10.016575
150	14.3	16.682863	-2.382863
151	12.6	18.285618	-5.685618

152 rows × 3 columns

In [39]:

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

Out[39]:

	실제값	예측값	오차	오차절대값	오차제곱
0	23.6	28.648960	-5.048960	5.048960	25.491998
1	32.4	36.495014	-4.095014	4.095014	16.769138
2	13.6	15.411193	-1.811193	1.811193	3.280421
3	22.8	25.403213	-2.603213	2.603213	6.776718
4	16.1	18.855280	-2.755280	2.755280	7.591567
...	...	...	...	...	...
147	17.1	17.403672	-0.303672	0.303672	0.092216
148	14.5	13.385941	1.114059	1.114059	1.241127
149	50.0	39.983425	10.016575	10.016575	100.331779
150	14.3	16.682863	-2.382863	2.382863	5.678036
151	12.6	18.285618	-5.685618	5.685618	32.326247

152 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 [40]:

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

Out[40]:

```
3.162709871457379
```

In [41]:

```
np.mean(dat['오차절대값'])
```

Out[41]:

```
3.162709871457379
```

MSE (mean squared error)

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

In [42]:

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

Out[42]:

```
21.517444231177006
```

In [43]:

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

Out[43]:

```
21.517444231177006
```

In [44]:

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

In [45]:

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

Out[45]:

```
21.517444231176995
```

RMSE (root mean squared error)

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

In [46]:

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

```
4.638689926172799
```

결정계수

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

In [47]:

```
# 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 : 0.74
테스트 데이터 세트 점수 : 0.71
```

In [48]:

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

    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.