Open in Colab

머신 러닝 맛보기 https://teachablemachine.withgoogle.com/ 에서 손톰 사진으로 만들어 볼것. 다 만든 다음 .js 파일로 다운 받아서 https://ml-app.yah.ac/ 이사이트로 가서 3개 파일 올려서 카메라 오하 연동하여 음성가지 나오는 프로그램 해볼 것.

인공지능 정의

인공지능은 일반적으로 인간 지능이 필요하거나 인간이 분석할 수 있는 범위를 벗어난 대규모 데이터를 포함하는 방식으로 추론, 학습 및 행동할 수 있는 컴퓨터와 머신을 빌드하는 과학 분야입니다.

AI는 컴퓨터 공학, 데이터 분석 및 통계, 하드웨어 및 소프트웨어 엔지니어링, 언어학, 신경 과학은 물론 철학과 심리학 등 다양한 학문을 포괄하는 광범위한 분야입니다.

비즈니스의 운영 수준에서 AI는 주로 머신러닝과 딥 러닝을 기반으로 하는 기술로, 데이터 분석, 예측 및 예상, 객체 분류, 자연어 처리, 추천, 지능형 데이터 검색 등에 사용됩니다.

머신러닝 정의 - 기계학습

머신러닝은 인공지능의 하위 집합으로, 많은 양의 데이터를 제공하여 명시적으로 프로그래밍하지 않고 신경망과 딥 러닝을 사용하여 시스템이 자율적으로 학습하고 개선할 수 있게 해줍니다.

머신러닝을 통해 컴퓨터 시스템은 더 많은 '경험'을 쌓으면서 스스로 지속적으로 조정하고 향상시킬 수 있습니다. 따라서 처리할 더 크고 다양한 데이터 세트를 제공함으로써 이러한 시스템의 성능을 향상시킬 수 있습니다.

사용 사례 범위

머신러닝은 거의 모든 산업 및 비즈니스 활동에서 사용되고 있습니다. 머신러닝은 물류 업계에서 발송 및 배송 경로를 최적화하는 데 도움이 되고, 소매업계는 쇼핑 경험을 맞춤설정하고 재고를 관리하며, 제조업체에서는 공장을 자동화하고, 모든 곳에서 조직의 보안을 유지하는 데 도움을 줍니다. 사용자가 음성을 사용해 스마트폰이나 스피커에 쿼리하면 머신러닝이 요청을 이해하고 결과를 찾는 데 도움을 줍니다. 머신러닝의 사용 사례 범위는 방대하며 끊임없이 확장되고 있습니다.

머신러닝의 중요성

데이터 생성 속도는 날로 빨라지고 있습니다. 전 세계는 그 어느 때보다 많은 데이터를 생성하고 있습니다. 머신러닝 없이는 이 모든 데이터를 분석하고 활용하는 것이 거의 불가능할 것입니다. 따라서 머신러닝은 인간이 컴퓨터 및 기타 기계로 할 수 있는 일의 완전히 새로운 영역을 열어가고 있습니다. 머신러닝은 사기 감지, 보안 위협 식별, 맞춤설정 및 추천, 챗봇을 통한 자동화된 고객 서비스, 스크립트 작성 및 번역, 데이터 분석 등과 같은 중요한 기능을 통해 비즈니스에 도움을 줍니다. 또

한 머신러닝은 자율 주행 차량, 드론, 비행기, 증강 현실과 가상 현실, 로봇공학과 같은 미래의 흥미진진한 혁신을 주도하고 있습니다.

머신러닝, 인공지능, 딥 러닝의 차이점은 무엇인가요?

인공지능(AI)과 머신러닝(ML)은 동의어로 사용되는 경우가 많지만 서로 바꿔 사용할 수 있는 용어는 아닙니다.

인공지능은 인간의 지능과 유사한 방식으로 추론하고, 학습하고, 행동할 수 있는 컴퓨터 및 기계 또는 인간이 분석할 수 있는 규모를 넘어서는 데이터를 포함하는 시스템을 구축하는 것과 관련된 컴퓨터 과학의 한 영역입니다. 이 분야에는 데이터 분석, 통계, 하드웨어 및 소프트웨어 엔지니어링, 신경과학, 심지어 철학을 포함한 다양한 학문이 포함됩니다.

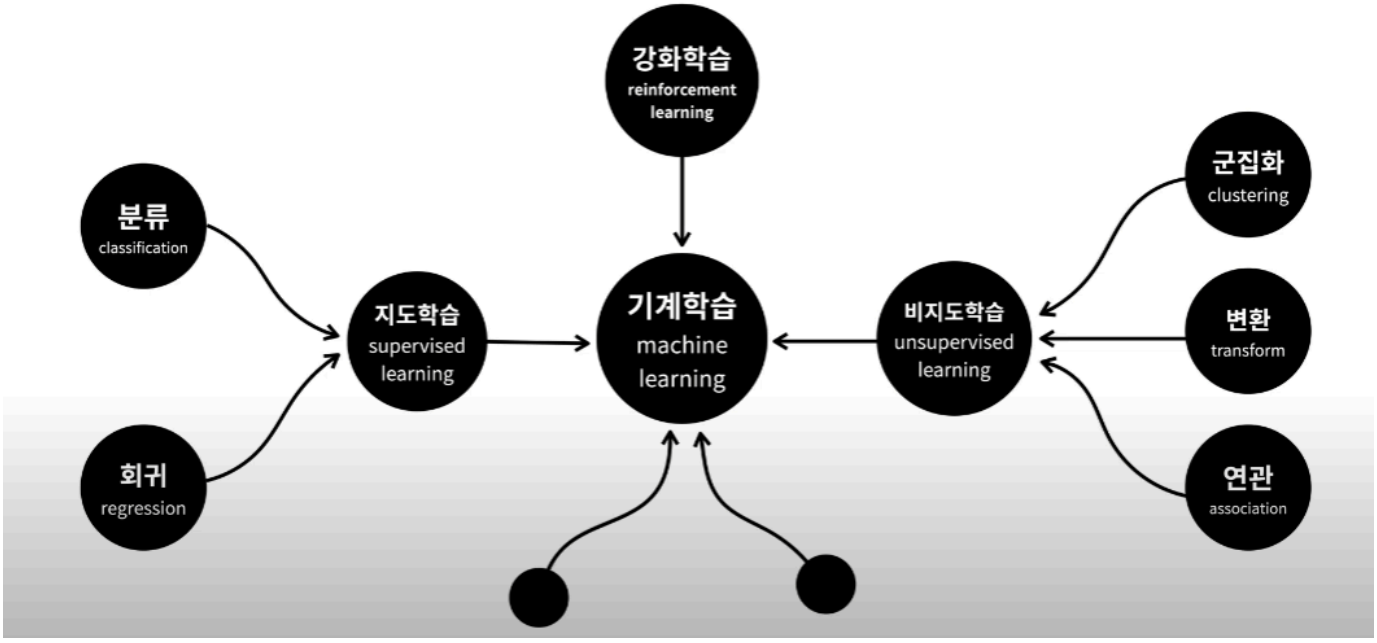
인공지능이 컴퓨터 공학의 광범위한 범주에 해당하는 반면, 머신러닝은 특별히 프로그래밍되지 않은 상태에서 특정 작업을 실행하도록 머신을 학습시키는 AI 응용 분야입니다. 머신러닝은 신경망, 지도 및 비지도 학습, 결정 트리, 선형 회귀와 같은 기술을 통해 데이터에서 지식을 추출하는 수단으로 보다 명시적으로 사용됩니다.

머신러닝이 인공지능의 하위 집합인 것처럼 딥 러닝도 머신러닝의 하위 집합입니다. 딥 러닝은 데이터 세트에 대해 신경망을 학습시키는 방식으로 작동합니다. 신경망은 데이터를 분류하고 분석하는 데 사용되는 계산 노드인 인공 뉴런 시스템을 사용하는 모델입니다. 신경망의 첫 번째 레이어에 데이터가 입력되면 각 노드가 결정을 내리고 이 정보를 다음 레이어의 여러 노드에 전달합니다. 레이어가 3개 넘게 있는 학습 모델을 '심층신경망' 또는 '딥 러닝'이라고 합니다. 일부 최신 신경망에는 레이어가 수백 또는 수천 개 있습니다.

머신러닝의 유형

머신러닝에서 학습 데이터란 무엇인가요? 사용 중인 머신러닝 모델의 유형에 따라 다릅니다.

일반적으로 머신러닝에 사용되는 모델은 세 가지입니다.



지도 학습(정답이 있는)은 라벨이 지정된 학습 데이터(구조화된 데이터)를 사용하여 특정 기능을 라벨에 매핑하는 머신러닝 모델입니다. 여기서 지도는 기계를 가르친다는 의미입니다. 마치 문제집을 푸는 것과 비슷해요.문제집에는 문제가 있고 답이 있습니다.문제와 정답을 비교하고 맞추다 보면 점점 문제를 푸는 것에 익숙해 지게 됩니다. 이후에는 비슷한 문제를 만나면 오답에 빠질 확률이 낮아집니다. 이렇듯 문제집으로 학생을 가르치듯 데이터로 컴퓨터를 학습시켜서 모델을 만드는 방식을 지도 학습이라 합니다. 예)손톱감시앱, 레몬에이드 판매량 예측

지도학습은 문류와 회귀로 나누어 집니다. 연속적인 데이터로부터 결과를 예측 할 때(예측결과(종속변수)가 숫자일때)는 **회귀(Regression)** 모델을 주어진 데이터를 정해진 범주에 따라 분류 할 때(예측결과(종속변수)가 숫자가 아닐 때)는 **분류(Classification)** 모델을 사용됩니다.

오늘날 사용되는 가장 일반적인 지도 학습 알고리즘은 다음과 같습니다.

- 선형 회귀
- 다항식 회귀
- K-최근접 이웃
- 나이브 베이즈
- 결정 트리

비지도 학습(정답이 없는)은 라벨이 지정되지 않은 데이터(비정형 데이터)를 사용하여 패턴을 학습하는 머신러닝 모델입니다. 지도 학습에 포함되지 않는 방법들이며 여기에 속하는 도구들은 대체로 기계에게 데이터에 대한 통찰력을 부여하는 것이라고 이야기 할 수 있습니다. 통찰의 사전적인 의미는 예리한 관찰력으로 사물을 꿰뚫어 봄 이라 뜻입니다. 즉 누가 정답을 알려 주지 않았는데

도 무엇가에 대한 관찰을 통해 새로운 의미나 관계를 밝혀 내는 것이라 할수 있습니다. 데이터의 성격을 파악하거나 데이터의 차이를 정리정돈하는 것에 주로 사용됩니다.

비지도학습

unsupervised learning

비지도 학습에는 군집화와 변환, 연관등이 있습니다. 군집화는 비슷한 것들을 찾아서 그룹을 만드는 것입니다. 분류와 혼돈되겠지만 예를 들면 이사를 했는데 엄청나게 많은 물건들이 많이 썻여 있는데 이걸 비슷한 것끼리 모아서 적당한 그룹을 만드는 것을 군집화라고 하고, 그룹을 만들고 난 후에 각각의 물건을 적당한 그룹에 위치시키는 것을 분류라고 합니다. 즉, 어떤 대상들을 구분해서 그룹을 만드는 것이 군집화라면 분류는 어던 대상이 어떤 그룹에 속하는 지를 판단하는 것이라고 할 수 있습니다. 비슷한 행을 그룹핑하는 것을 군집화라고 한다. 연관 규칙은 서로 관련이 있는 특성 다시 말해서 열을 찾아주는 머신러닝 기법이라는 것을 알수 있습니다. 특성을 그룹핑 해주는 것을 연관 규칙이라고 합니다.

오늘날 사용되는 가장 일반적인 비지도 학습 알고리즘은 다음과 같습니다.

- 퍼지 의미
- K-평균 클러스터링
- 계층적 군집화
- 부분 최소 제곱

강화 학습(경험)은 일련의 시행착오 실험을 통해 '실습하여 학습'하는 것으로 설명할 수 있는 머신러닝 모델입니다. 강화 학습은 학습을 통해서 향상 시키는 점에서는 지도학습이랑 비슷합니다. 차이점은 지도학습이 정답을 알려주는 문제집이 있는 것이라면 강화학습은 어떻게 하는 것이 더 좋

은 결과를 낼수 있는지를 스스로 느끼면서 실력 향상을 위해 노력하는 수련과 비슷합니다. 마치 게임을 통해 실력을 키워서 더 좋은 보상을 받는 것과 같습니다. 강화 학습의 한 예는 Google 연구자들이 바둑 게임을 플레이하도록 강화 학습 알고리즘을 학습시킨 경우입니다. 이 모델은 바둑 규칙에 대한 사전 지식이 없었고 단순히 바둑돌을 무작위로 이동하며 가장 좋은 동작을 '학습'했습니다. 이 알고리즘은 긍정적 및 부정적 강화를 통해 머신러닝 모델이 게임에서 인간 플레이어를 이길 수 있을 정도로 학습되었습니다.

머신러닝의 장점

패턴 인식

머신러닝 알고리즘에서 사용하는 데이터가 많을수록 해당 데이터에서 트렌드와 패턴을 더 잘 발견할 수 있습니다. 예를 들어 전자상거래 웹사이트에서는 머신러닝을 사용하여 사람들이 사이트에서 쇼핑하는 방식을 파악하고 해당 정보를 사용하여 사용자에게 더 나은 추천을 제공하거나 새로운 제품 기회로 이어질 수 있는 트렌드 데이터를 찾을 수 있습니다.

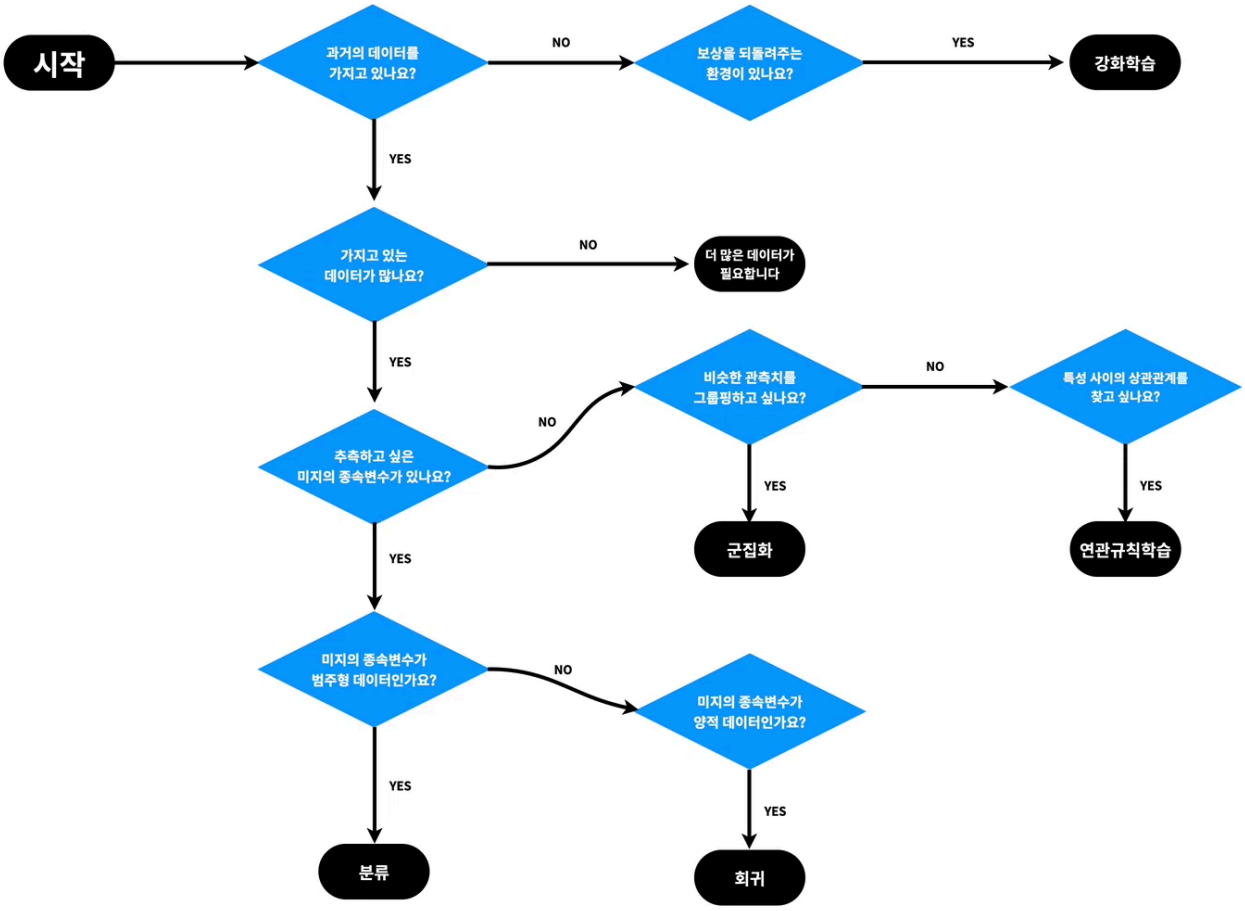
자동화

머신러닝과 인공지능은 노동자의 반복되는 지루한 작업을 상당 부분 덜어줄 수 있습니다. 로봇 프로세스 자동화와 같은 유틸리티는 사람들이 더 의미 있는 작업을 수행하지 못하도록 방해하는 지루한 비즈니스 작업 중 일부를 수행할 수 있습니다. 컴퓨터 비전 및 부정적 반응 감지 알고리즘을 사용하면 로봇이 조립 라인에서 물품을 선택하고 포장하는 데 도움이 될 수 있습니다. 상시 가동되는 사기 감지 및 위협 평가 머신러닝은 문제가 되기 전에 보안 결함을 찾아낼 수 있습니다.

지속적 개선

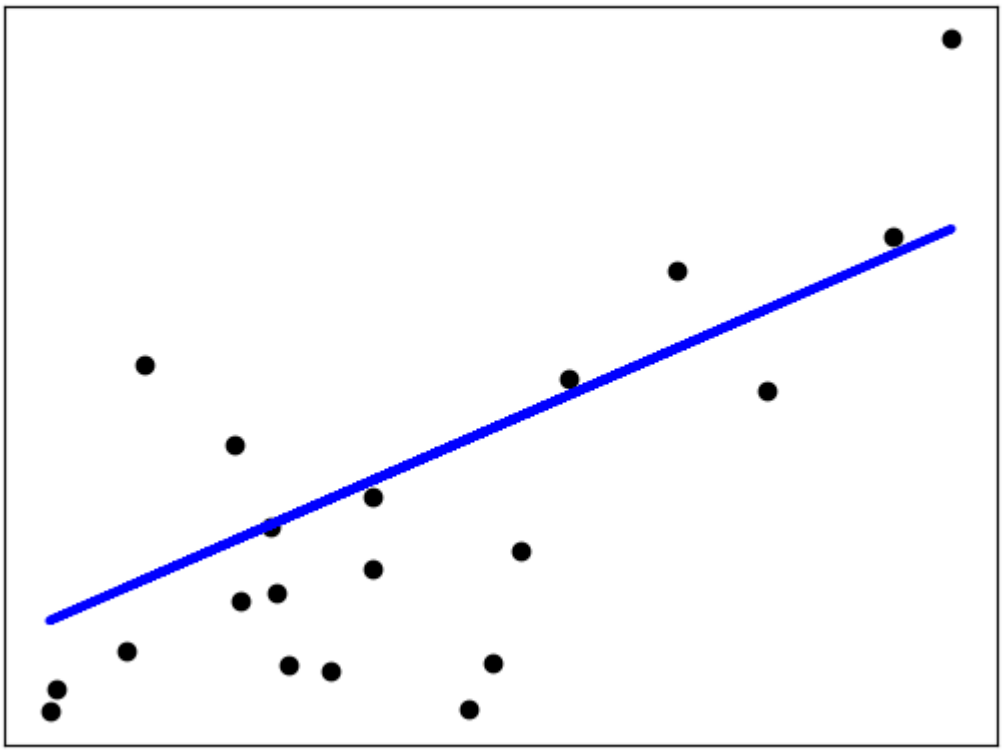
적절한 데이터 종류가 제공되는 경우 머신러닝 알고리즘은 계속해서 더 빠르고 정확하게 개선될 것입니다. 텍스트 생성 방식을 지속적으로 개선하고 있는 GPT-3 데이터 세트가 좋은 예입니다.

머신러닝의 지도



1. Linear Regression(선형 회귀)

단순 선형 회귀(Simple Linear Regression)라고도 한다.
머신러닝의 목적은 데이터의 알려진 속성들을 학습하여 예측 모델을 만드는데 있다. 이때 찾아 낼 수 있는 가장 직관적이고 간단한 모델은 선(line)이다. 선형회귀란 데이터를 가장 잘 대변하는 최적의 선을 찾은 과정이다. 아래 그래프에서 검정색 점이 데이터이다. 이 데이터를 가장 잘 표현하는 선이 파란색 직선이며, 이는 일차 함수(y=ax+b) 형태로 나타난다.



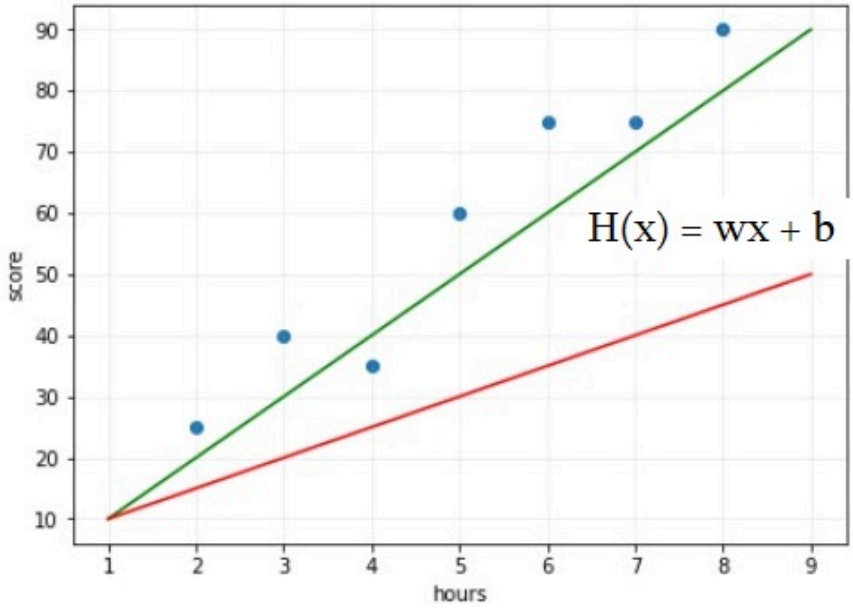
선형회귀 직선은 x와 y의 관계를 요약해서 설명해준다고 볼 수 있다. 이 때 x를 독립 변수라고 하며, x에 의해 영향을 받는 값인 y를 종속 변수라고 한다. 선형 회귀는 한개 이상의 독립 변수 x와 y의 관계를 모델링 하는데, 만약 독립 변수 x가 하나라면 단순 선형 회귀, 2개 이상이면 다중 선형 회귀라고 한다.

- 독립 변수는 예측(Predictor)변수, 설명(Explanatory), 특성(Feature) 등으로 불린다.
- 종속 변수는 반응(Response)변수, 레이블(Label), 타겟(Target) 등으로 불린다.

- 단순 선형 회귀 분석 : $y=\beta_1 x+\beta_0$
- 다중 선형 회귀 분석 : $y=\beta_0+\beta_1 x_1+\beta_2 x_2+...w_n x_n$

공부시간에 따른 시험 점수를 나타낸 표가 있다. 알고있는 데이터로부터 x와 y의 관계를 유추하고, 표에 없는 10시간 공부하였을 때 성적을 예측하고자 한다. 머신러닝에서는 x와 y의 관계를 유추하기 위해 식을 세우게 되는데 이것을 가설(H)이라고 한다.

hours	score
2	25
3	40
4	35
5	60
6	75
7	75
8	90

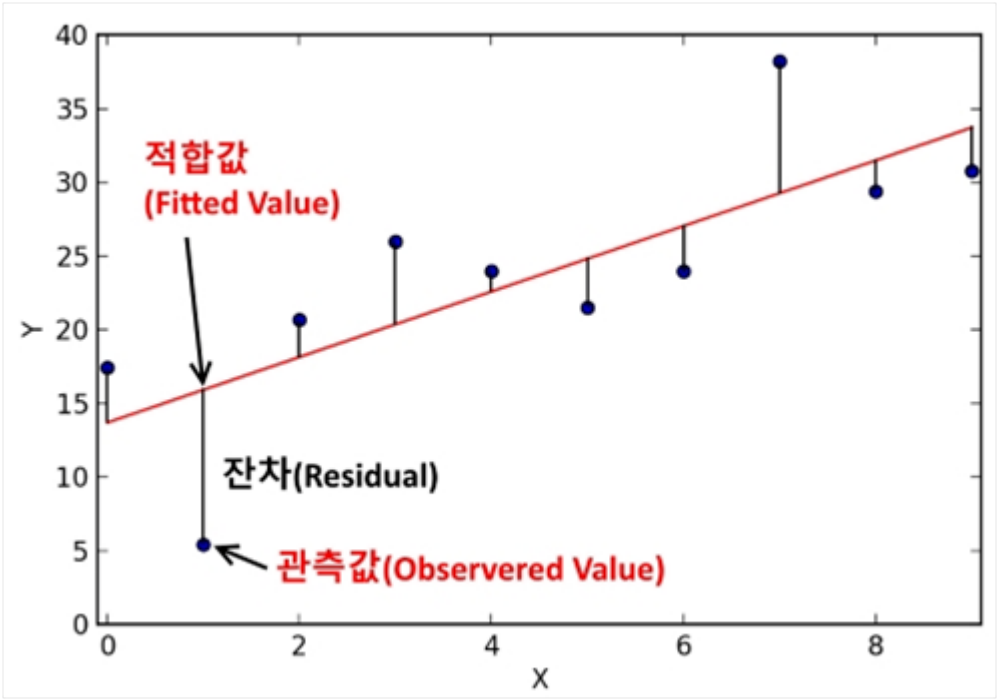


예측 모델

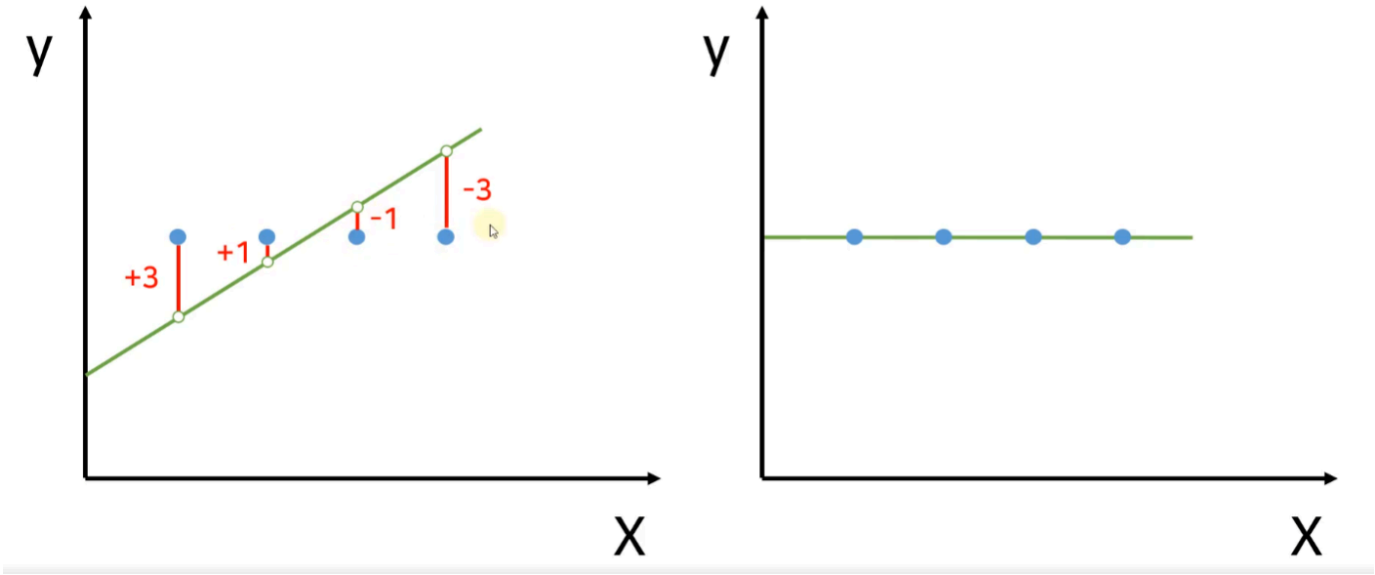
⭐최소자승법(Ordinary Least Square, OLS) : 위의 scatter plot에 Best fit Line을 그리면, 그것이 회귀 예측 모델이 된다. 이 회귀 직선을 그리기 위해서는 최소자승법(=최소제곱법)을 이용해야 한다. 최소자승법은 잔차제곱의 합(RRS, residual sum of squares)를 최소화하는 가중치 벡터를 구하는 방법이다. 선형 회귀식 : y=α+βX (α=y절편(intercept), β=회귀계수(Coefficients))

💭 잔차란 예측값과 관측값의 차이이다. 잔차 제곱의 합을 RSS 혹은 SSE(Sum of Square Error)라고도 하며, 이 값이 회귀모델의 비용 함수(Cost function)이 된다. 머신러닝에서 이 Cost Function을 최소화 하는 모델을 찾는 과정을 학습이라고 한다.

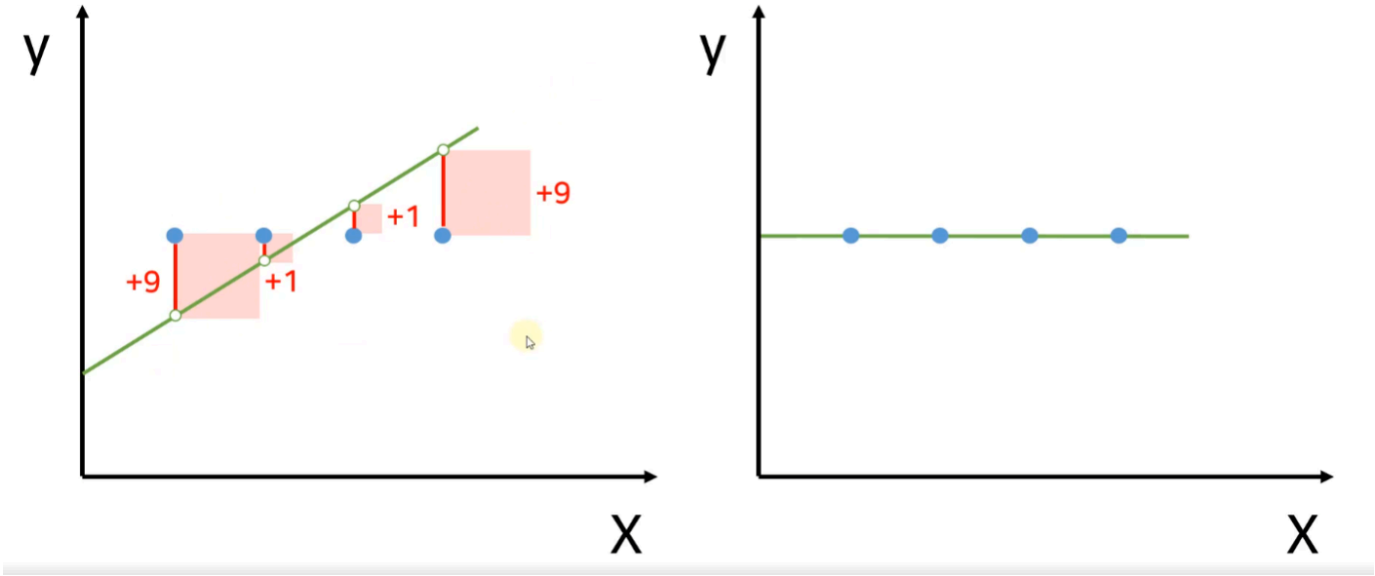
RSS(SSE)= i=0 ∑ n(y i−f(x i)) 2 = i=0 ∑ n(y i−(αx i+β)) 2



둘중에 데이터를 더 잘표현한 것은 어느 것인가.

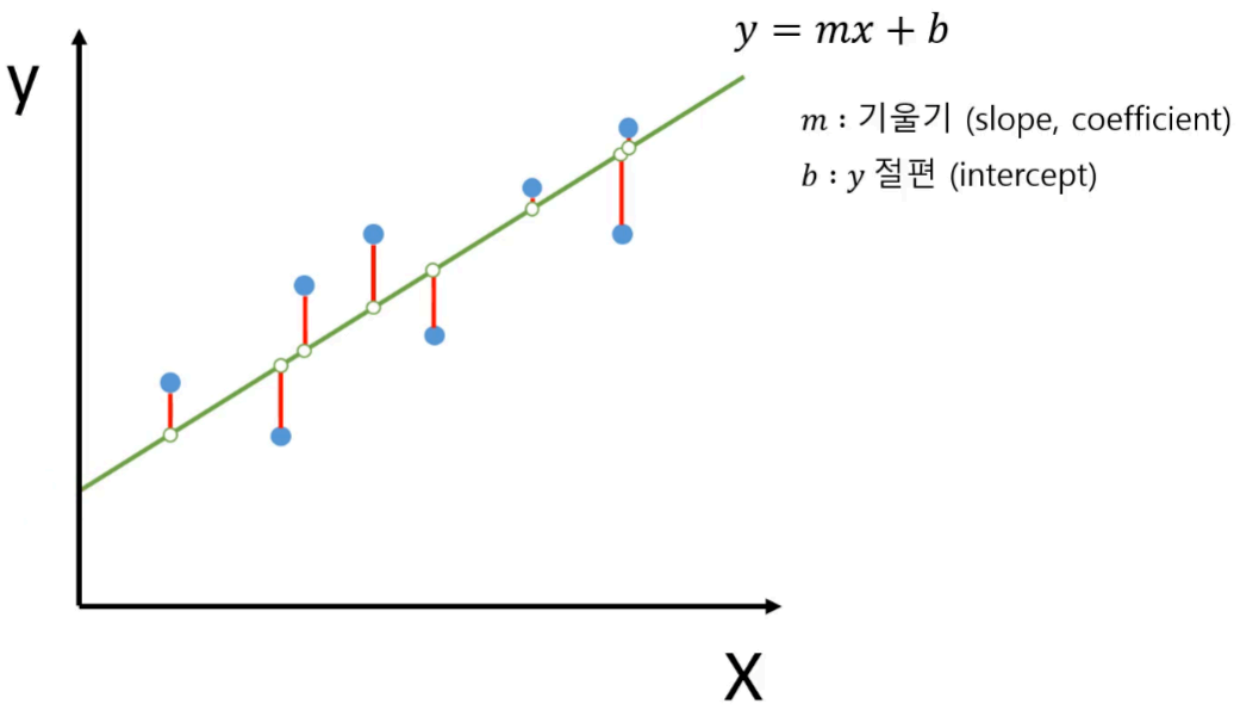


왼쪽 그래프에서 예측값과 실제값을 다 더하면 0로가 됨.



올바른 값을 도출해 내기 위해 보통 제곱을 해줍니다. 이렇게 하고 나면 이제 모든 값들이 다 양수로 바뀔 거고 이 값들 을 다 더한 값이 최소화가 되는 직선을 그어주면 되는 겁니다. 여기서는 우리가 이렇게 지금 제곱했으니까 사각형으로 표시 되며 이면적들이 합이 최소화되는게 최적의 선이 되는 거죠. 다시 말해 실제 값에서 예측 값을 뺀 값을 제곱해 주니까 모두 양수가 될 거고 그러면 지금처럼 사각형이 되고 이 사각형 면적들의 합이 최소가 되

는 직선 하나를 찾으면 되는 겁니다. 그래서 우리가 원하는 것은 아래 그림 처럼 최적의 선을 찾는 것입니다.



이것이 선형회귀의 이론입니다.

```
import sklearn
sklearn.__version__
# 최신버전으로 업그레이드 후 할 것.
```

```
'1.3.2'
```

공부 시간에 따른 시험 점수

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
# 구글 드리이브에 있는 파일을 사용하려면 이것을 먼저 실행 할 것.
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/활용편7-머신러닝/ScikitLearn/LinearF
```

```
dataset.head() # head() - 데이터 상위 5개까지만 보여주는 함수
```

	hour	score
0	0.5	10
1	1.2	8
2	1.8	14
3	2.4	26
4	2.6	22

```
# 독립변수와 종속 변수의 관계를 인과관계라고 한다. 인과관계는 상관관계에 포함된다.  예) 온도(독립ᄇ
X = dataset.iloc[:, :-1].values # 처음부터 마지막 컬럼 직전까지의 데이터 (독립 변수 - 원인)
y = dataset.iloc[:, -1].values # 마지막 컬럼 데이터 (종속 변수 - 결과)
```

```
X, y
```

```
(array([[ 0.5],
        [ 1.2],
        [ 1.8],
        [ 2.4],
        [ 2.6],
        [ 3.2],
        [ 3.9],
        [ 4.4],
        [ 4.5],
        [ 5. ],
        [ 5.3],
        [ 5.8],
        [ 6. ],
        [ 6.1],
        [ 6.2],
        [ 6.9],
        [ 7.2],
        [ 8.4],
        [ 8.6],
        [10. ]]),
 array([ 10,   8,  14,  26,  22,  30,  42,  48,  38,  58,  60,  72,  62,
         68,  72,  58,  76,  86,  90, 100]))
```

선형회귀 모델 생성

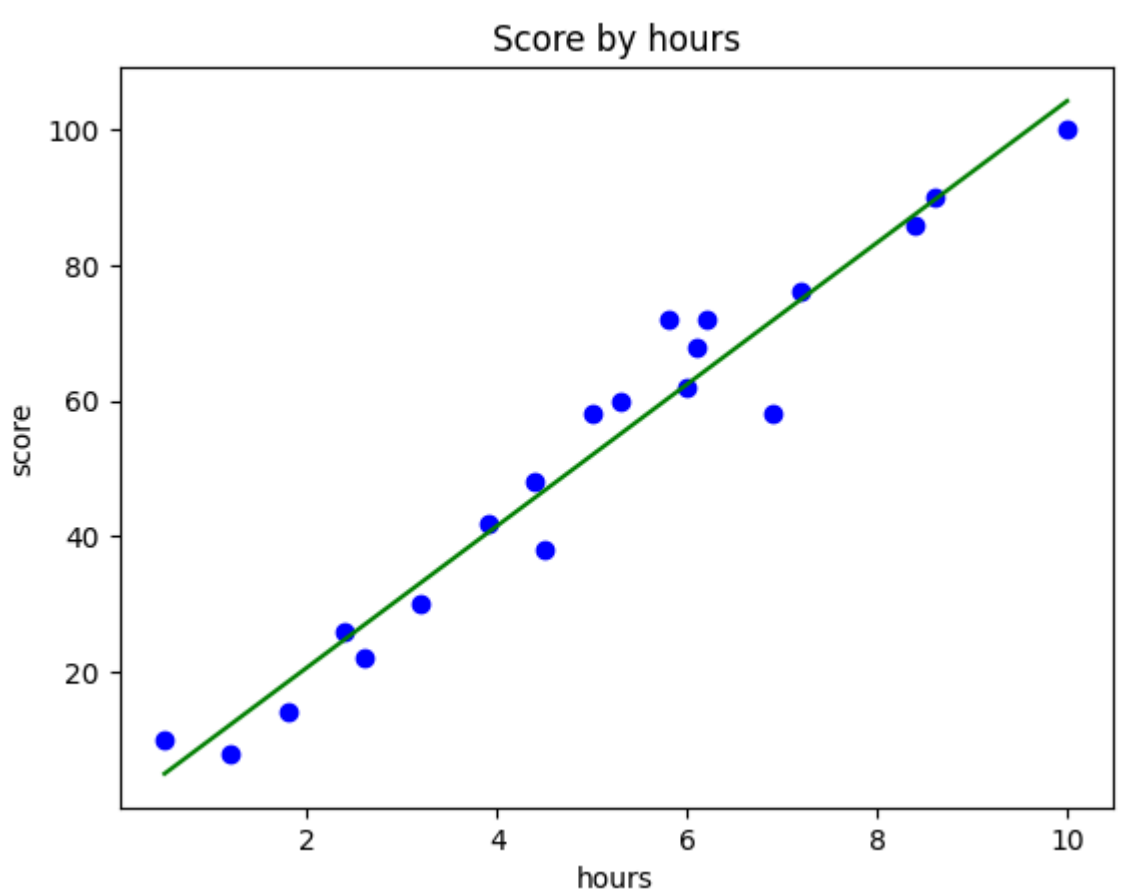
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression() # 객체 생성
reg.fit(X, y) # 학습 (모델 생성)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = reg.predict(X) # X 에 대한 예측 값
y_pred
```

```
array([  5.00336377,  12.31395163,  18.58016979,  24.84638795,
        26.93512734,  33.20134551,  40.51193337,  45.73378184,
        46.77815153,  52.        ,  55.13310908,  60.35495755,
        62.44369694,  63.48806663,  64.53243633,  71.84302419,
        74.97613327,  87.5085696 ,  89.59730899, 104.2184847 ])
```

```
plt.scatter(X, y, color='blue') # 산점도
plt.plot(X, y_pred, color='green') # 예측 선 그래프
plt.title('Score by hours') # 제목
plt.xlabel('hours') # X 축 이름
plt.ylabel('score') # Y 축 이름
plt.show()
```




```
print('9시간 공부했을 때 예상 점수 : ', reg.predict([[9]])) # 2차원 배열 형태로 -> [[9], [8], [7]]
```

```
9시간 공부했을 때 예상 점수 :  [93.77478776]
```

```
reg.coef_ # 기울기 (m)
```

```
array([10.44369694])
```

```
reg.intercept_ # y 절편 (b)
```

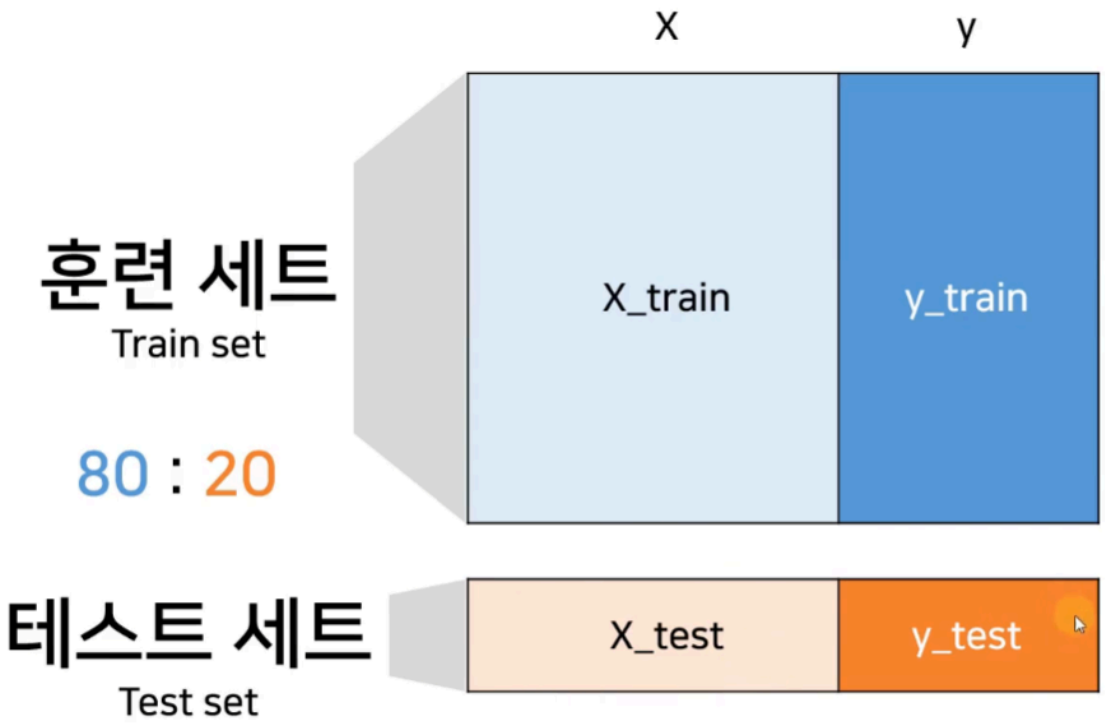
```
-0.218484702867201
```

우리의 식

y = mx + b -> y = 10.4436x - 0.2184

데이터 세트 분리

머신러닝에서 이미 가지고 있는 데이터를 가직도 모델을 평가 하기 위해서 데이터 세트를 분리하는 작업을 한다.



대체로 훈련세트 80 가지고 나머지 20의 테스트 세트로 잘 동작하는 본다.(또는 70:30)

```
import matplotlib.pyplot as plt
import pandas as pd
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```

```
dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/활용편7-머신러닝/ScikitLearn/LinearF
dataset
```

	hour	score
0	0.5	10
1	1.2	8
2	1.8	14
3	2.4	26
4	2.6	22
5	3.2	30
6	3.9	42
7	4.4	48
8	4.5	38
9	5.0	58
10	5.3	60
11	5.8	72
12	6.0	62
13	6.1	68
14	6.2	72
15	6.9	58
16	7.2	76
17	8.4	86
18	8.6	90
19	10.0	100

```
X = dataset.iloc[:, :-1].values  # 독립변수
y = dataset.iloc[:, -1].values # 종속 변수
```

```
from sklearn.model_selection import train_test_split # 테스트 세트 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0) # 훈련 80
```

```
X, len(X) # 전체 데이터 X, 개수
```

```
(array([[ 0.5],
        [ 1.2],
        [ 1.8],
        [ 2.4],
        [ 2.6],
        [ 3.2],
        [ 3.9],
        [ 4.4],
        [ 4.5],
```

```
            [ 5. ],
            [ 5.3],
            [ 5.8],
            [ 6. ],
            [ 6.1],
            [ 6.2],
            [ 6.9],
            [ 7.2],
            [ 8.4],
            [ 8.6],
            [10. ]]),
     20)
```

```
X_train, len(X_train) # 훈련 세트 X, 개수 -> 80% 이므로 16개
```

```
(array([[5.3],
        [8.4],
        [3.9],
        [6.1],
        [2.6],
        [1.8],
        [3.2],
        [6.2],
        [5. ],
        [4.4],
        [7.2],
        [5.8],
        [2.4],
        [0.5],
        [6.9],
        [6. ]]),
 16)
```

```
X_test, len(X_test) # 테스트 세트 X, 개수 -> 20% 이므로 4개
```

```
(array([[ 8.6],
        [ 1.2],
        [10. ],
        [ 4.5]]),
 4)
```

```
y, len(y) # 전체 데이터 y
```

```
(array([ 10,   8,  14,  26,  22,  30,  42,  48,  38,  58,  60,  72,  62,
         68,  72,  58,  76,  86,  90, 100]),
 20)
```

```
y_train, len(y_train) # 훈련 세트 y -> 80% 이므로
```

```
(array([60, 86, 42, 68, 22, 14, 30, 72, 58, 48, 76, 72, 26, 10, 58, 62]), 16)
```

```
y_test, len(y_test) # 테스트 세트 y -> 20%
```

```
(array([ 90,   8, 100,  38]), 4)
```

분리된 데이터를 통한 모델링

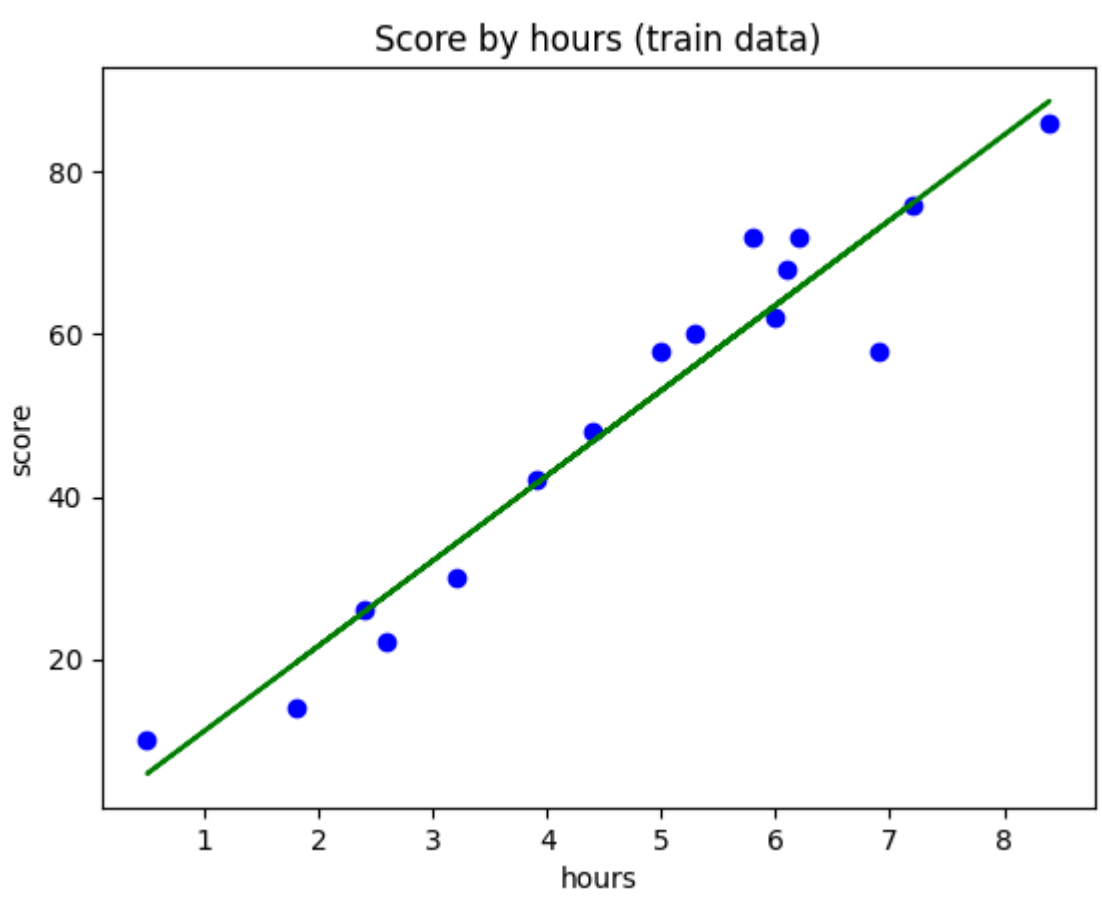
```
# 선형회귀 모델
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
```

```
reg.fit(X_train, y_train) # 훈련 세트로 학습
```

```
▾ LinearRegression
LinearRegression()
```

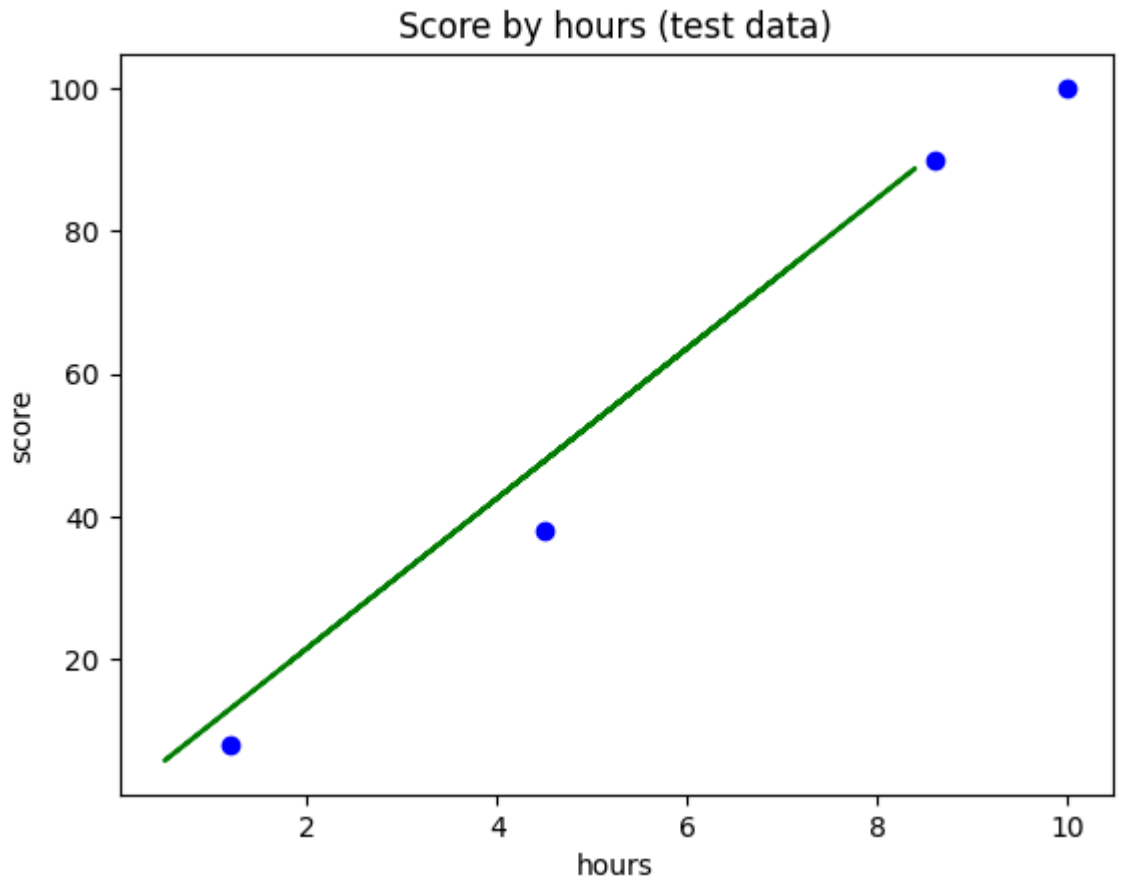
데이터 시각화 (훈련 세트)

```
plt.scatter(X_train, y_train, color='blue') # 산점도
plt.plot(X_train, reg.predict(X_train), color='green') # 선 그래프
plt.title('Score by hours (train data)') # 제목
plt.xlabel('hours') # X 축 이름
plt.ylabel('score') # Y 축 이름
plt.show()
```



데이터 시각화 (테스트 세트)

```
plt.scatter(X_test, y_test, color='blue') # 산점도
plt.plot(X_train, reg.predict(X_train), color='green') # 선 그래프 -> 모델을 가지고 만들었기 때문(
plt.title('Score by hours (test data)') # 제목
plt.xlabel('hours') # X 축 이름
plt.ylabel('score') # Y 축 이름
plt.show()
```




```
reg.coef_ # 기울기 -> 위에는 array([10.44369694])
```

```
array([10.49161294])
```

```
reg.intercept_ # y 절편 -> 위에는 -0.218484702867201
```

```
0.6115562905169369
```

모델 평가

```
reg.score(X_test, y_test) # 테스트 세트를 통한 모델 평가 -> 97점 정도의 점수
```

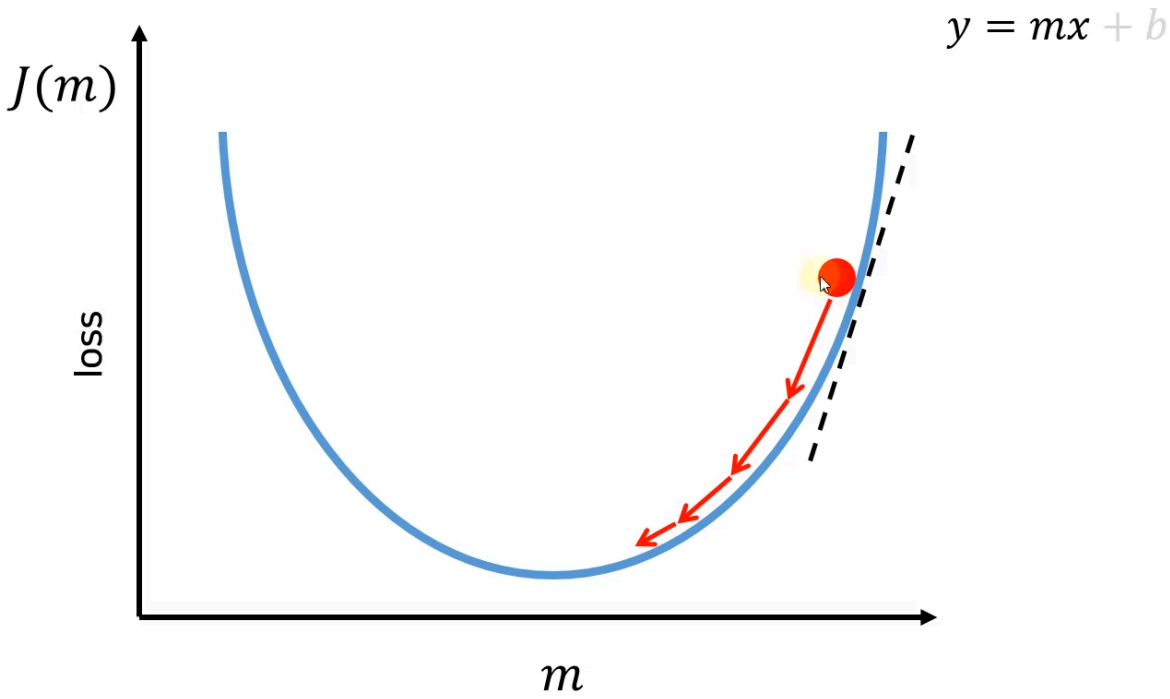
```
0.9727616474310156
```

```
reg.score(X_train, y_train) # 훈련 세트를 통한 모델 평가 -> 93.56점 정도의 점수
```

```
0.9356663661221668
```

경사 하강법 (Gradient Descent)

- 시각적으로 어느 부분이 목적지인지 전혀 확인되지 않는 산 속에서 산 밑으로 내려가야 하거나 산 정상까지 도달해야 한다면? 시각적으로 목적지 확인이 불가능하기 때문에 주변의 지형 등 형태 파악을 통해 이를 근거로 목적지까지 가야 하는 판단이 필요하게 됩니다.
- 목적지가 산의 정상이라면, 현재 자신의 위치에서 가장 경사가 높은 곳으로 계속 이동하다 보면 일반적으로 산의 정상에 도달할 수 있을 것이고, 목적지가 산 밑이라면 현재 위치에서 계속해서 가장 낮은 지점을 찾아 이동하다 보면 산 밑으로 내려갈 수 있게 됩니다. 산의 정상까지 가는 방법을 반대로 "경사 상승법(Gradient ascent)", 산 밑으로 내려가는 것을 "경사 하강법(Gradient descent)"이라고 부를 수 있습니다.

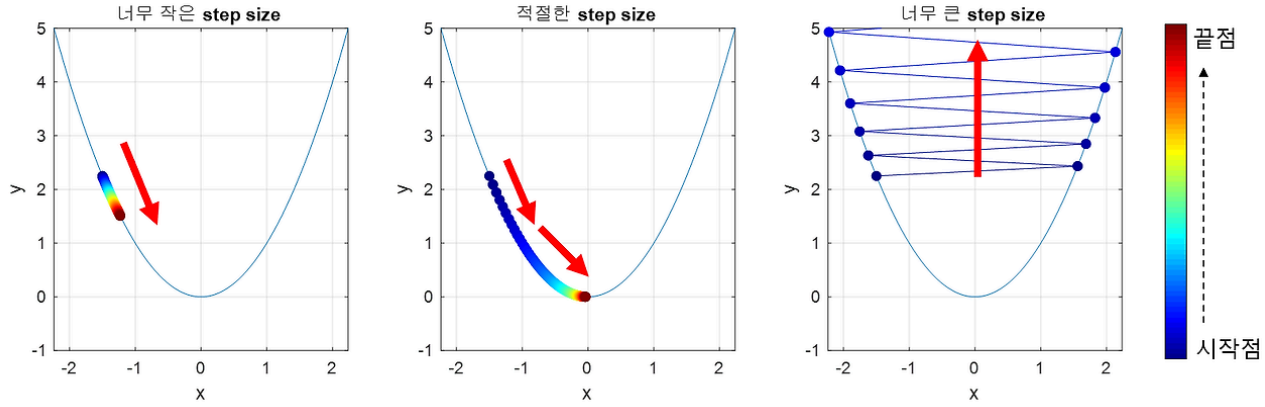


- 그림과 같이 기울기가 0이 되는 지점까지 가는게 목표이다.

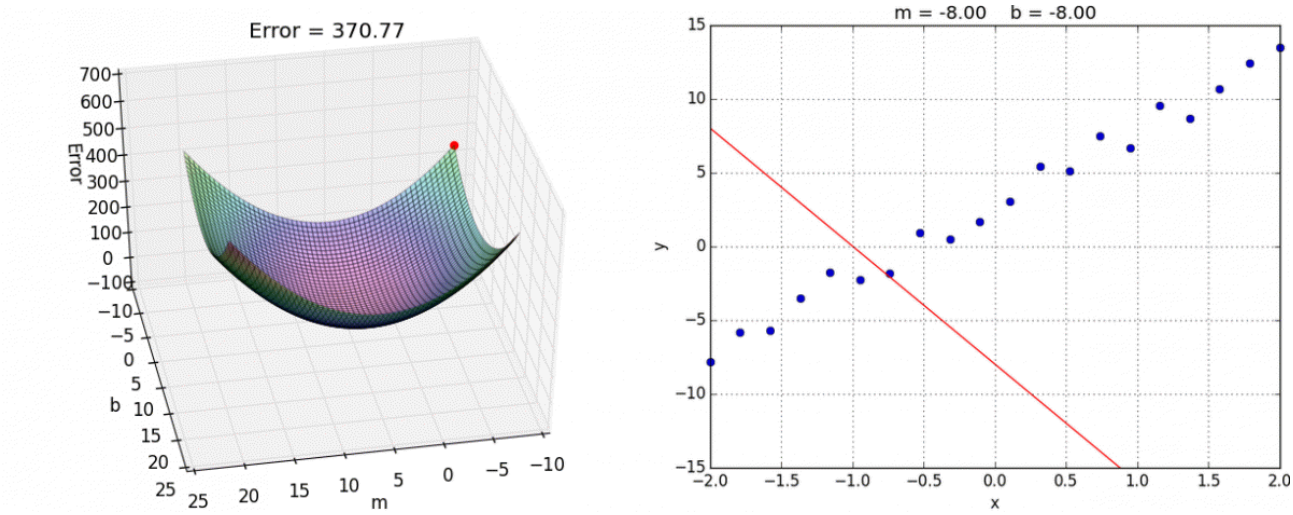
경사 하강법의 문제점

- 이런 경사 하강법도 몇 가지 문제점을 가지고 있습니다.
- 대표적으로 [적절한 학습률(Learning Rate), Local Minimum]이 있습니다.

- 통상적으로 학습률과 기울기 정보를 혼합하여 내가 나아갈 방향과 거리를 결정합니다.
- 따라서 학습률이 높으면 높을수록 더욱더 많이 나아갑니다.
- 그러므로 학습률이 높으면 한 번에 이동하는 거리가 커지므로 최적값에 빨리 수렴할 수 있다는 장점이 있습니다.
- 하지만 너무 크게 설정하면 최적값에 수렴하지 못하고 다른 곳으로 발산하는 현상이 나타날 수 있습니다.
- 정 반대로 학습률이 낮으면 발산하지 않는 대신 최적값에 수렴하는 시간이 오래 걸릴 수 있습니다.



- 위의 그림에서 볼 수 있듯이, 학습률을 적절히 조정하는 것이 매우 중요합니다.
- 일반적으로 => —0.001, 0.003, 0.01, 0.03, 0.1, 0.3 값들을 많이 사용한다.
- 최적의 파라미터를 찾기 위해서 훈련세트에 있는 모든 데이터들을 한번씩 다 사용하는 과정을 에포크(Epoch)라고 부릅니다.
- 데이터가 많으면 시간이 많이 걸리므로 확률적 경사하강법을 사용한다. 사이킷 런에서는 이 확률적 경사하강법 클래스를 제공한다. 이것에 시각화 자료가 아래와 같다.



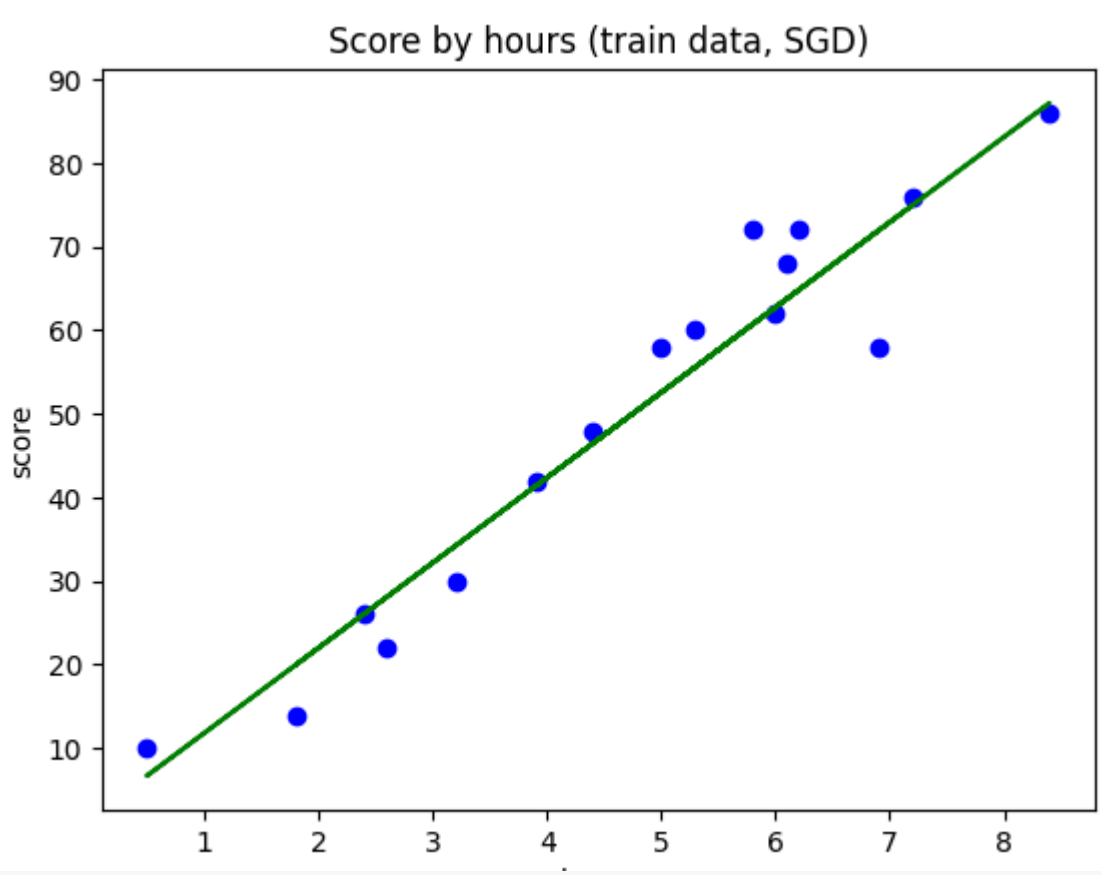
- Error가 로스를 말함.

```
from sklearn.linear_model import SGDRegressor # SGD : Stochastic Gradient Descent 확률적 경사 하강

sr = SGDRegressor() # 객체 생성
sr.fit(X_train, y_train)
```

```
▾ SGDRegressor
SGDRegressor()
```

```
plt.scatter(X_train, y_train, color='blue') # 산점도
plt.plot(X_train, sr.predict(X_train), color='green') # 선 그래프
plt.title('Score by hours (train data, SGD)') # 제목
plt.xlabel('hours') # X 축 이름
plt.ylabel('score') # Y 축 이름
plt.show()
```




```
sr.coef_, sr.intercept_
# 주의 : SGDRegressor() 객체를 생성할 때 random_state 값을 지정하지 않았으므로 결과가 다르게 나타날
# 위에서의 값은 array([10.44369694]) -0.218484702867201
```

```
(array([10.2062811]), array([1.95017289]))
```

```
sr.score(X_test, y_test) # 테스트 세트를 통한 모델 평가
```

```
0.9759945339358915
```

```
sr.score(X_train, y_train) # 훈련 세트를 통한 모델 평가
```

```
0.934497230156066
```

훈련 회수를 정한 경사 하강법

max_iter : 훈련 세트 반복 횟수 (Epoch 횟수) 많을 수로 손실률이 적어짐

eta0 : 학습률 (learning rate)

```
# 지수표기법
# 1e-3 : 0.001 (10^-3)
# 1e-4 : 0.0001 (10^-4)
# 1e+3 : 1000 (10^3)
```