인공지능 시스템 설계

2023.09.15(금)

교수: 이현순

LOGO

머신러닝 기초 : 사이킷런과 선형 회귀

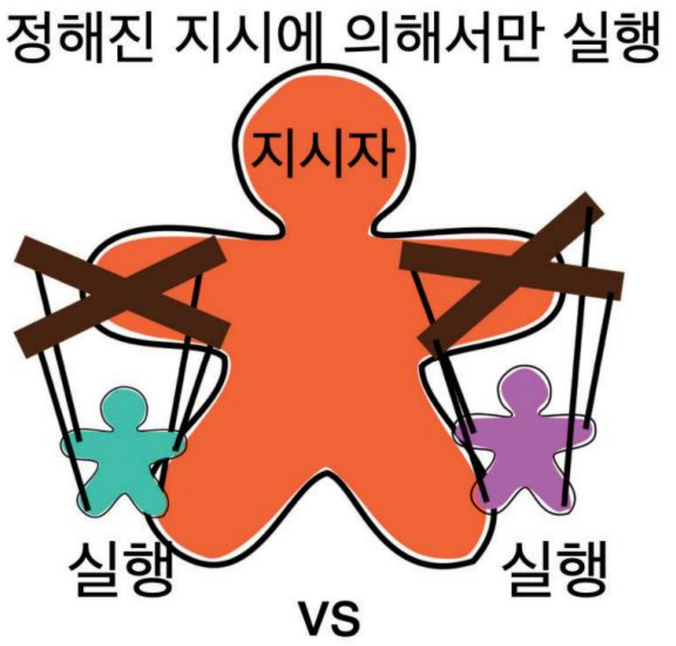
학습목표

- 인간의 지능과 인공지능의 차이점을 이해한다.
- 회귀분석의 개념과 독립변수, 종속변수를 이해한다.
- 사이킷런을 이용한 선형 회귀 알고리즘의 구현 방법을 익힌다.
- 다양한 오차의 개념을 이해하고 경사 하강법을 통해 최적화를 이루는 방법을 익힌다.
- 경사 하강법을 구현하기 위한 미분의 개념을 익힌다.

1.1 경험을 통해서 학습하는 인간을 통해 지능을 정의해 보자

- 학습 능력, 추론 능력, 지각 능력, 언어이해 능력을 통칭하여 **지능**intelligence이라고 한다. 이 지능은 유전적 요인과 함께 **사회화와 학습의 과정에서 형성되는 것**으로 알려져 있다.

- 해결해야 하는 문제를 풀어내기 위해 필요한 행동을 일일이 알려 주는 것은 가르치는 일이라기보다는 **지시**라고 할 수 있다.
- 이 지시를 받는 사람은 아무런 생각 없이 그대로 **실행**하기만 하면 된다.
- 이 일의 책임은 지시를 한 사람에게 있고, 지시를 실행할 수 있는 수준 이상의 지적 능력은 요구되지 않는다, 그리고 **새로운 문제가 발생하면 그에 맞는 지시를 다시 제공**해야 한다.



1.1 경험을 통해서 학습하는 인간을 통해 지능을 정의해 보자

- **문제를 해결하는 일반적 방법**을 가르치는 방법도 있다.
- 이것은 단순한 지시가 아니라 해당 업무에서 만날 수 있는 모든 문제의 해법을 이해하도록 만들어 그 업무를 맡기기 위한 과정이다.
- 복잡하고 큰 문제를 해결하기 위해 우리는 지시를 단순히 실행하는 사람보다는 이렇게 일정한 부분을 맡아서 처리할 **능동적인 행위자**를 더 중요하게 여길 것이다.

문제해결을 위한 능동적 활동

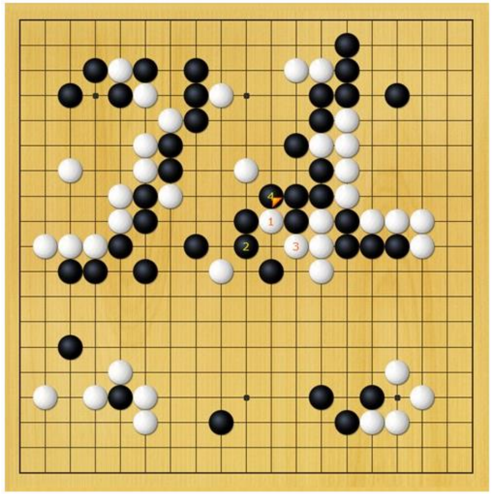
문제 해결을 위한 방법들

1.1 경험을 통해서 학습하는 인간을 통해 지능을 정의해 보자

- 문제를 해결할 때 따라야 하는 일들을 지시하는 것을 **프로그램**program이라고 한다.
- 그런데 때로는 이런 프로그램을 만드는 일이 쉽지 않을 수도 있다. 풀이법을 설명하기가 힘들거나 불가능한 경우이다.
- 풀이법이 존재하더라도 아직 우리가 그 방법을 알지 못한다면 역시 문제 해결 과정을 가르쳐줄 수 없다.
- 문제의 풀이법을 컴퓨터에 알려주지 않아도 컴퓨터가 스스로 잘하는 방법을 찾아내게 할 수 있을까? 컴퓨터가 **데이터를 기반으로 스스로 학습**할 수 있다면 우리는 더욱 복잡한 일을 맡길 수 있을 것이다.

1.1 경험을 통해서 학습하는 인간을 통해 지능을 정의해 보자

- **이세돌을 이긴 컴퓨터 프로그램 "알파고"**
- 규칙 기반 방식 : 컴퓨터에게 어떤 작업을 시키기 위해서 프로그램을 작성하여 지시를 하는 방식
- 기계 학습 방식 : 데이터를 기반으로 컴퓨터가 스스로 학습을 하여 문제를 해결하는 방식



바둑은 경우의 수가
너무 많아서 게임규칙을
모두 규칙화 하기 힘들다

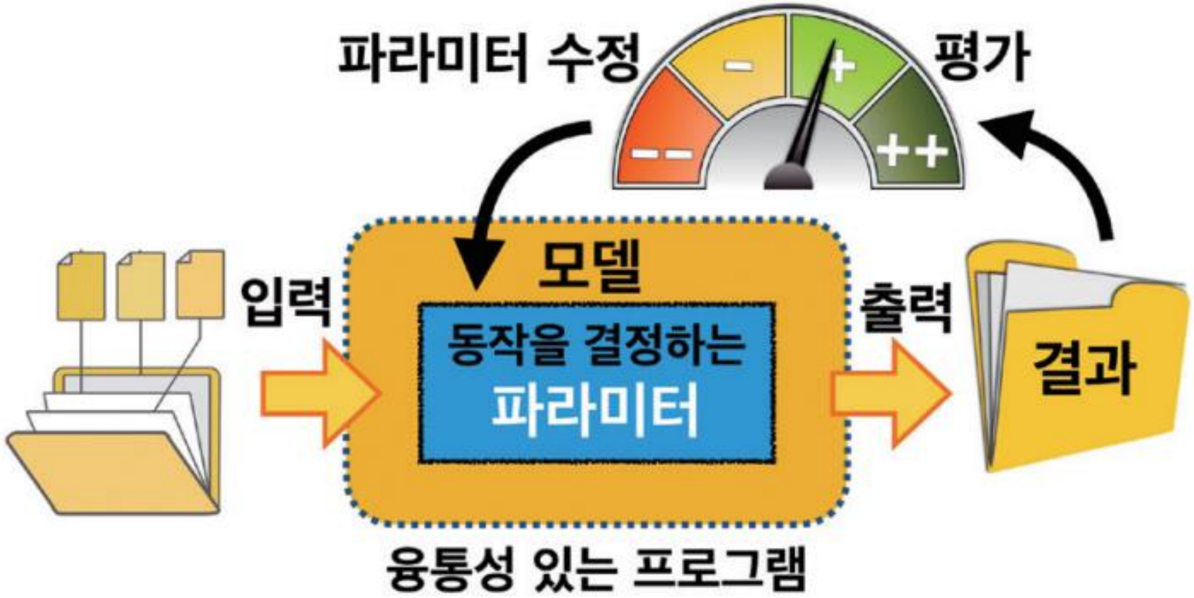
기계학습을 적용

인간이 진 것이 아니라
이세돌이 진 것이다!

- 알파고"처럼 컴퓨터한테 바둑 경기의 규칙과 이전 경기의 **기보**棋譜만을 알려주면, 컴퓨터가 스스로 바둑의 원리를 학습하여 바둑을 둘 수 있다.

1.1 경험을 통해서 학습하는 인간을 통해 지능을 정의해 보자

- 이것이 머신러닝이 하려는 일이다. 머신러닝에서는 그림과 같이 **동작 방식을 일일이 지시하는 프로그램을 설계하지 않는다.**
- 대신 변경 가능한 **파라미터** parameter에 의해 동작이 결정되는 **융통성있는 프로그램**을 만든다. 이것을 **모델** model이라고 부른다.



- 파라미터가 바뀌면 동작도 바뀌는 것이다. 여기에 데이터를 다양하게 제공하여 프로그램이 이 데이터를 얼마나 잘 처리하는지 살펴본다.
- 그리고 좋은 동작이 나오도록 파라미터를 변경하는 일을 하는데, 이 과정을 **학습**이라고 부른다.

1.1 경험을 통해서 학습하는 인간을 통해 지능을 정의해 보자

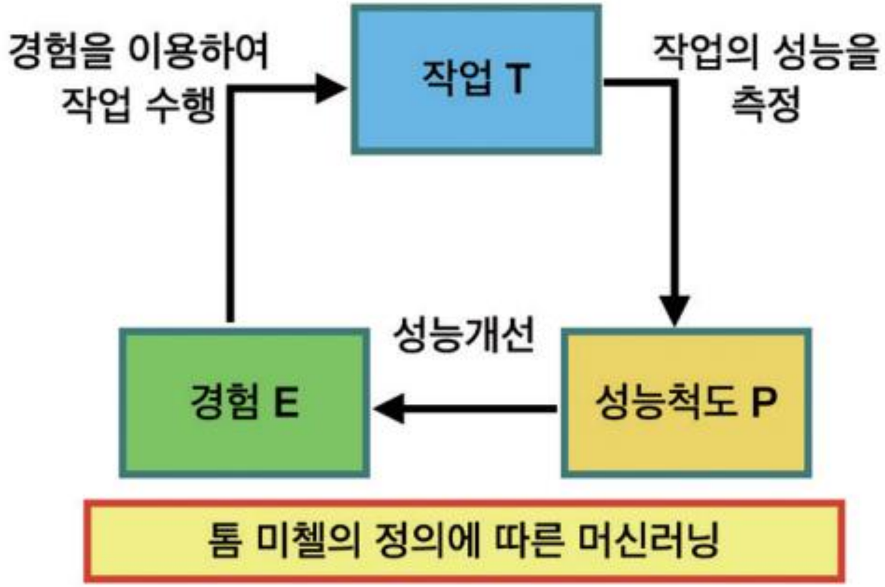
- **인공지능**artificial intelligence이란 **인간의 학습 능력과 추론, 지각 능력을 인공적으로 구현한 것**으로 정의할 수 있다.
- 학습 능력과 추론을 가진 지능이라는 것 자체가 모호성을 포함하고 있기 때문에 인공지능이 매우 큰 범주의 개념이라고 한다면 기계학습 혹은 **머신러닝**machine learning은 매우 **구체적이고 엄밀한 정의**를 가지고 있다.

1.1 경험을 통해서 학습하는 인간을 통해 지능을 정의해 보자

- 예를 들어 0에서 9 사이의 숫자에 대한 사람의 손글씨를 여러 장(수만 장 이상)의 이미지로 입력받아, 이 이미지를 분류하는 작업을 수행하는 경우를 고려해 보자.
- 이때 반복적인 **학습을 통해 분류 성능을 점점 향상시킬 수 있는 효율적인 알고리즘**이 존재한다면, 이 알고리즘은 **기계학습을 수행**한다고 볼 수 있을 것이다.
- 여기서 말하는 기계란 **프로그래밍이 가능한 장치**를 말하며 우리가 생각하는 컴퓨터가 바로 이것이다.
- 그리고 인간의 지능과 유사하게 다양한 상황에 따라 능동적으로 판단하고 자율적으로 주변환경과 상호작용하는 지능을 **범용 인공지능**Artificial General Intelligence:AGI으로 지칭한다.

1.2 머신러닝의 정의와 종류를 알아보자

- 머신러닝을 공학적으로 다루기 위해서는 세 가지 중요한 요소가 필요하다.
- 우선 **해결해야 할 문제(작업)** T이다.
- 그리고 이 일을 수행하는 동작을 P라는 **성능 척도**를 통해 평가할 수 있어야 한다.
- 그리고 지속적인 훈련 **경험** E를 통해 이러한 평가의 점수를 더 나은 상태로 바꿀 수 있어야 하는 것이다.



- 이러한 머신러닝은 인공지능이라는 분야의 매우 중요한 영역으로 간주된다. 그리고 요즘 각광을 받고 있는 인공 신경망을 이용한 **딥러닝**deep learning 분야 역시 이 머신러닝의 한 분야로 볼 수 있다.

1.2 머신러닝의 정의와 종류를 알아보자

- 머신러닝은 일반적으로 기계에게 답을 알려주는 "교사"의 존재 여부에 따라 크게 지도 학습과 비지도 학습으로 나누어진다. 그리고, 에이전트의 액션에 대한 보상을 학습하는 강화 학습을 별도의 영역으로 다룬다.

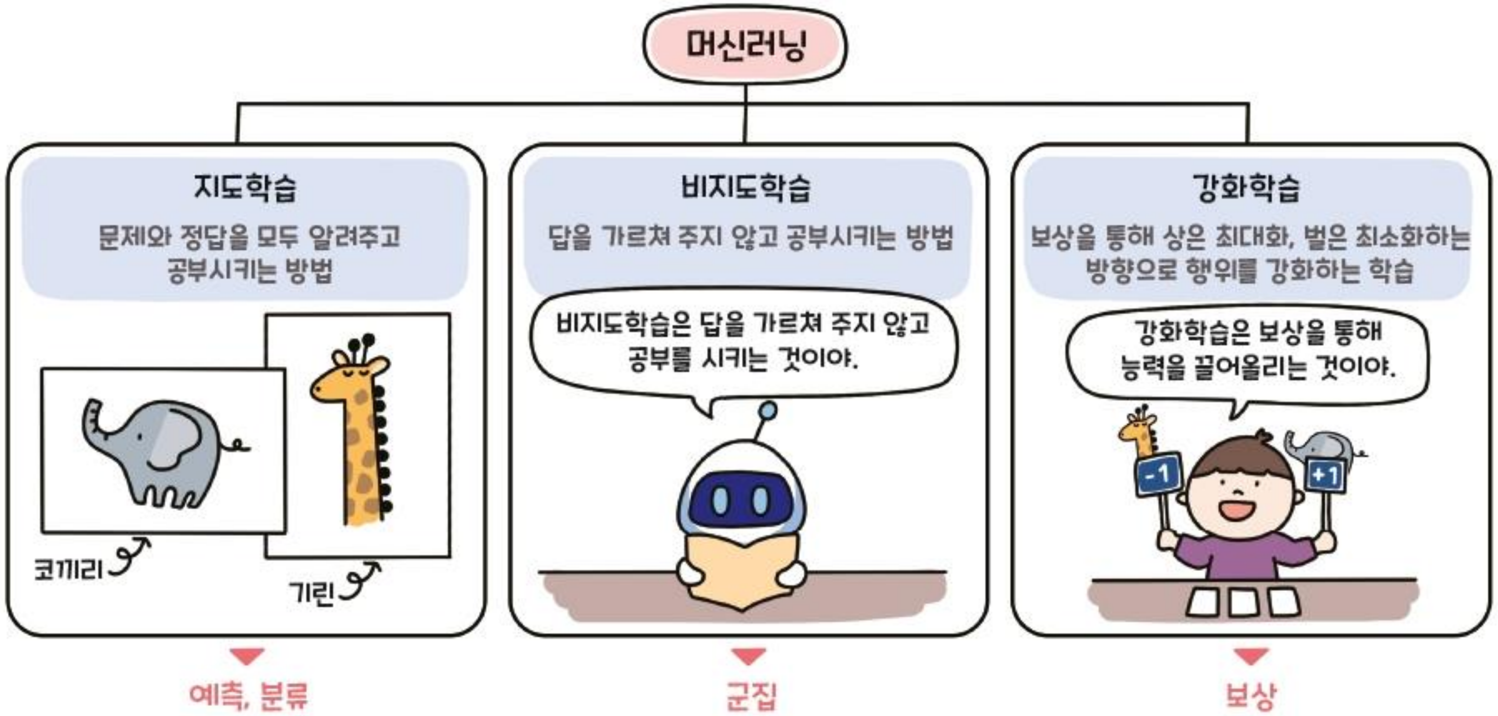


그림 머신러닝 분류

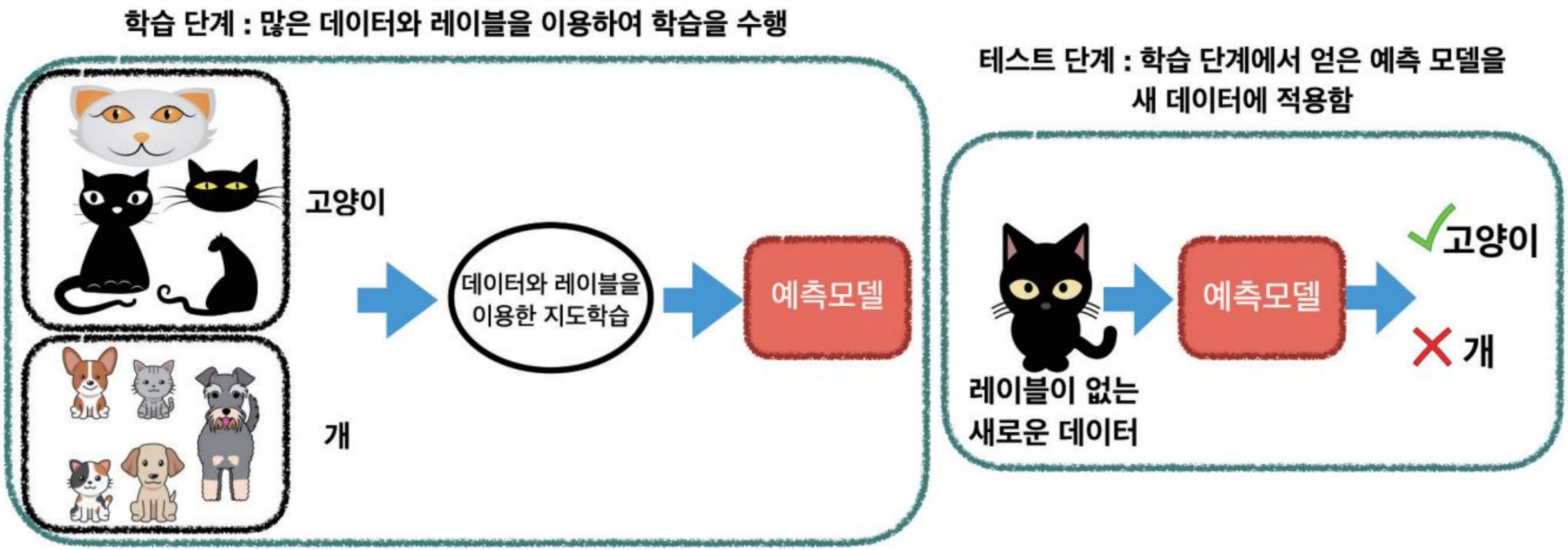
1.2 머신러닝의 정의와 종류를 알아보자

입력값에 대한 정답 또는 결과값을 레이블(Label)이라고 함.

- **지도 학습**supervised learning:
  - 지도 학습에서 컴퓨터는 "교사"에 의해 데이터와 정답의 역할을 하는 **레이블**label을 제공받는다.
  - 지도 학습의 목표는 입력을 출력에 매핑하는 일반적인 규칙을 학습하는 것이다.
  - 예를 들어서 고양이와 개를 구분할 때, 교사가 고양이인지 개인지 레이블링 된 데이터를 충분히 제공한 뒤에 학습을 하도록 하는 과정이 필요하다.

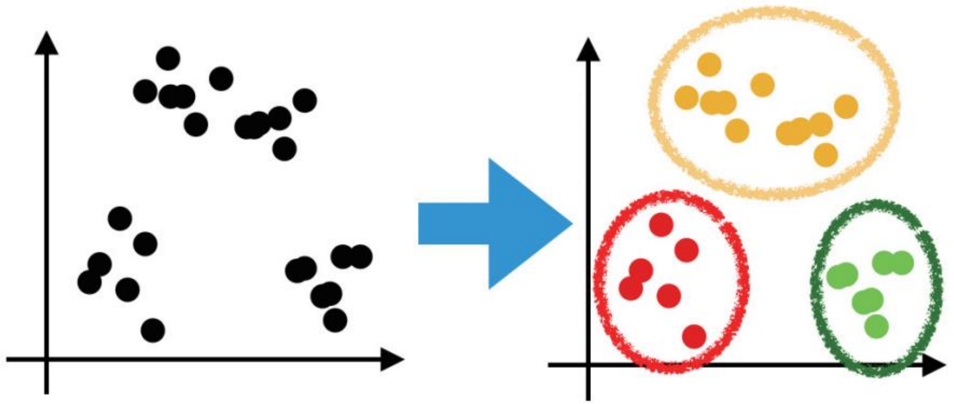
1.2 머신러닝의 정의와 종류를 알아보자

- 그림과 같이 학습 단계에서 만들어진 **예측 모델**은 새로운 데이터에 대하여 이전의 학습을 바탕으로 고양이인지 개인지 맞히게 된다.



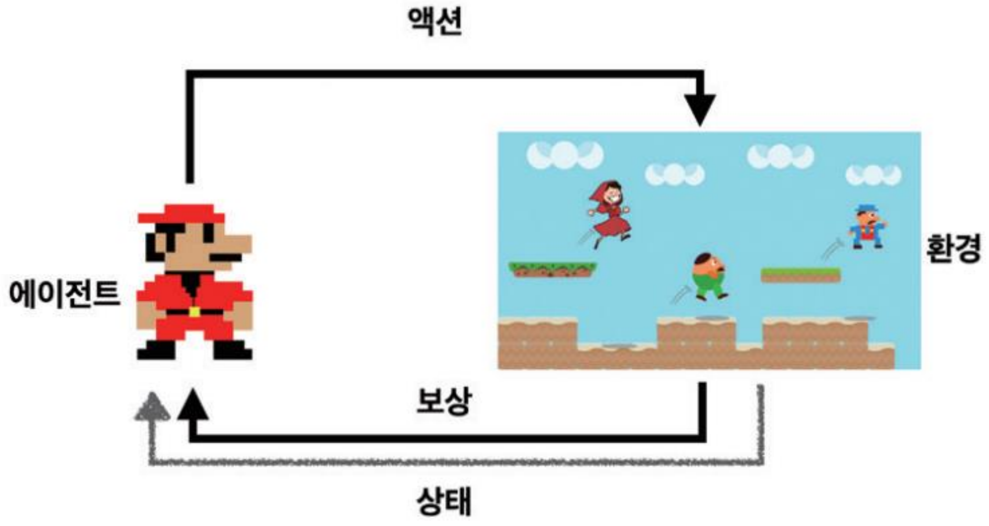
1.2 머신러닝의 정의와 종류를 알아보자

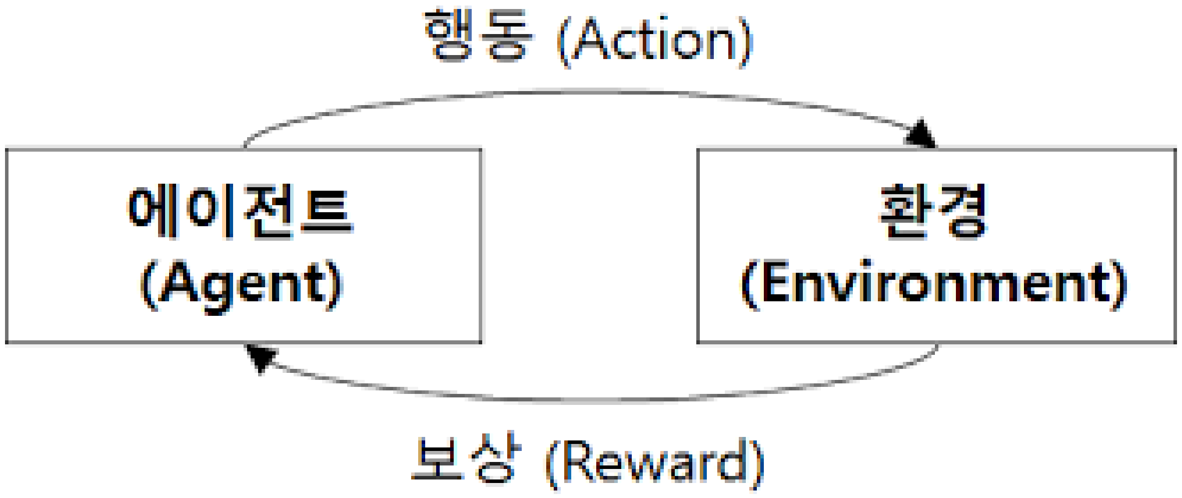
- 비지도 학습unsupervised learning:
  - 지도 학습과는 달리 외부에서 정답(레이블)을 주지 않고 학습 알고리즘이 스스로 입력으로부터 어떤 구조를 발견하는 학습이다.
  - 비지도 학습을 사용하면 데이터에서 숨겨진 패턴을 발견할 수 있다.
  - 비지도 학습의 대표적인 예가 군집화clustering이다. 이 방법은 그림과 같이 **주어진 데이터를 특성에 따라 둘 이상의 그룹으로 나누는 것**이다. 이 특성을 구분하는 방법을 컴퓨터가 스스로 학습하는 것이다. 여러 뉴스를 비슷한 것들끼리 묶어서 제공한다든지 하는 일에 사용될 수 있다.



1.2 머신러닝의 정의와 종류를 알아보자

- **강화 학습**reinforcement learning:
  - 강화 학습은 에이전트, 환경, 액션, 보상과 상태라는 학습 데이터를 준다.
  - 예를 들어서 그림에 나타난 게임 캐릭터(에이전트)가 게임 환경에서 특정한 액션을 수행하고 이에 대한 보상을 통해 행동을 결정하는 **정책**policy을 바꾸어 나가는 방식이다.
  - 게임과 같은 분야에서 높은 수준을 보이는 프로그램들이 이러한 방식으로 만들어지는 경우가 많다.





1.2 머신러닝의 정의와 종류를 알아보자

- 강화 학습reinforcement learning:
  - 강화학습을 이해하기 위해 알아야 할 개념들
    - 에이전트(Agent) : 주어진 문제 상황에서 행동하는 주체
    - 상태(State) : 현재 시점에서의 상황
    - 행동(Action) : 플레이어가 취할 수 있는 선택지
    - 보상(Reward) : 플레이어가 어떤 행동을 했을 때 따라오는 이득
    - 환경(Environment) : 문제 그 자체를 의미
    - 관찰(Observation) : 에이전트가 수집한(보고 듣는) 환경에 대한 정보

1.2 머신러닝의 정의와 종류를 알아보자

- **강화 학습**reinforcement learning을 통한 게임 제어 활용의 예

(a) 슈퍼마리오(https://www.youtube.com/watch?v=WzxmH1Cx2Yg)

슈퍼마리오: https://www.youtube.com/watch?v=WzxmH1Cx2Yg

1.2 머신러닝의 정의와 종류를 알아보자

- 강화 학습reinforcement learning을 통한 로보틱스 제어 활용의 예

(a) 로봇 팔 제어 : 탁구(https://www.youtube.com/watch?v=SH3bADiB7uQ)

로봇 팔 제어(탁구): https://www.youtube.com/watch?v=SH3bADiB7uQ

(b) 로봇 팔 제어 : 팬케이크 뒤집기(https://www.youtube.com/watch?v=W_gxLKSsSIE)

로봇 팔 제어(팬케이크 뒤집기): https://www.youtube.com/watch?v=W_gxLKSsSIE

(c) 보스턴 다이내믹스(Boston Dynamics)의 로봇(https://www.youtube.com/watch?v=NR32ULxbjYc)

보스턴 다이내믹스: https://www.youtube.com/watch?v=NR32ULxbjYc

1.2 머신러닝의 정의와 종류를 알아보자

- **기계 학습의 전형적인 과정**
  - 실제에서는 다양한 형태로 나타난다.

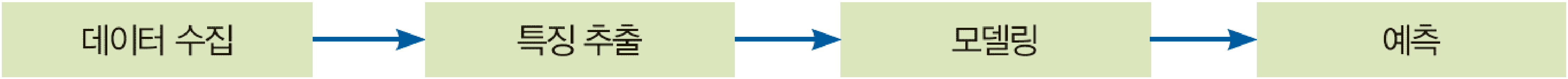


그림 기계 학습의 과정

1.3 회귀분석과 독립변수, 종속변수

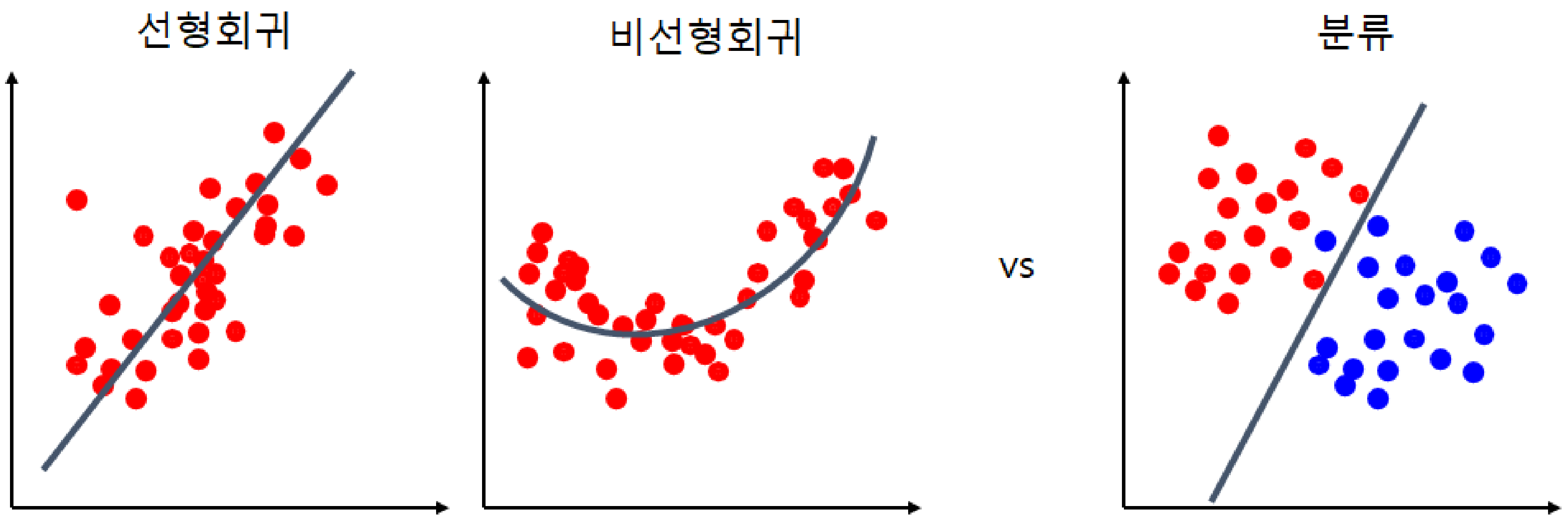
- 일상생활에서는 접하기 힘든 다소 생소한 용어인 **회귀**regression란 어딘가로 돌아 간다는 의미이다.
- 회귀분석은 대표적인 지도 학습 알고리즘으로 관측된 데이터를 통해 독립변수와 종속변수 사이의 숨어 있는 관계를 추정하는 것이다. 각 용어의 의미는 아래와 같다.

용어	해설
변수	변경될 수 있는 양이나 조건
독립변수	연구자가 임의로 조절할 수 있는 변수로 실험 영역에 있어 다른 변수에 영향을 받지 않는 변수
종속변수	관측이나 측정이 가능한 변수로 독립변수에 영향을 받아서 변화하는 변수

- 선형 회귀는 임의의 변수 x(독립 변수)와 이 변수에 따른 또 다른 변수 y(종속 변수)와의 상관관계를 모델링하는 기법
- 두 변수의 관계를 알아내거나 이를 이용하여 y가 없는 x값에 대하여 y를 예측하는데 사용되는 통계학의 기법

1.3 회귀분석과 독립변수, 종속변수

- 선형회귀 및 비선형회귀 vs 분류의 개념



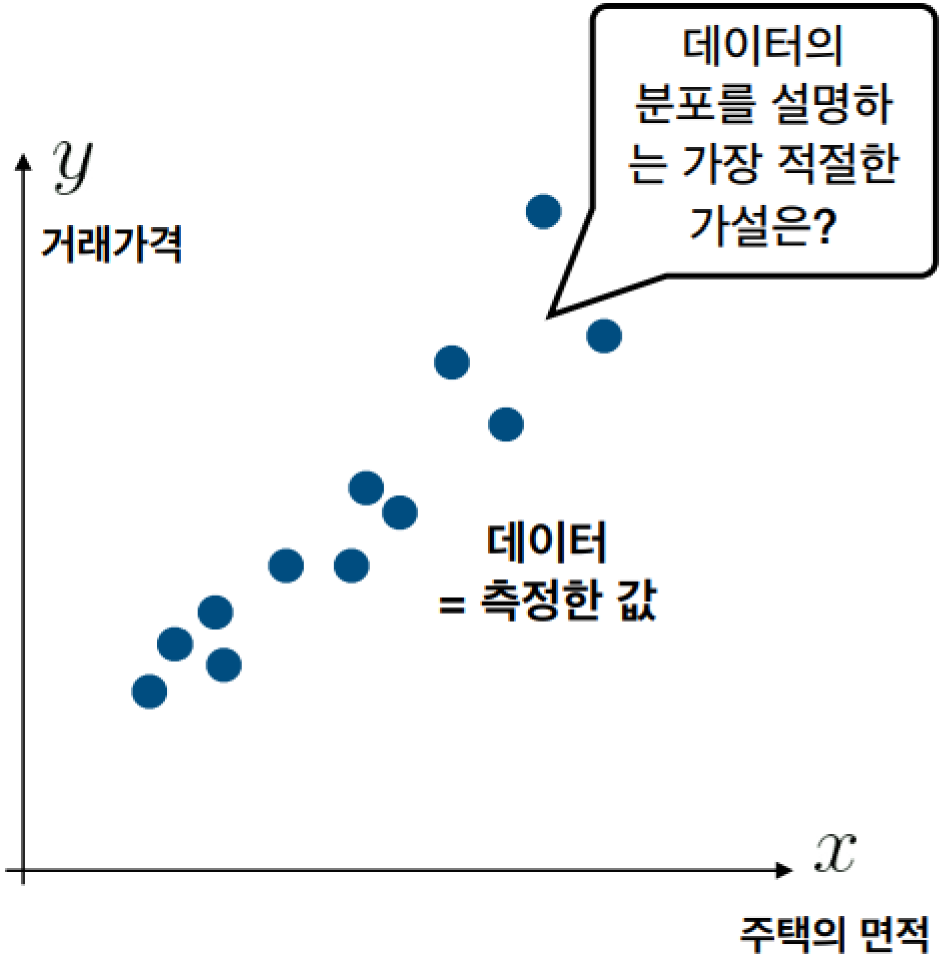
1.3 회귀분석과 독립변수, 종속변수

- 어떤 연구자가 특정 지역의 주택 면적과 최근 2년의 거래가격의 관계를 조사하는 경우를 생각해보자.
- 일반적으로 주택의 면적이 큰 경우 판매가격도 높은 경우가 많은데, 여기서 다른 변수에 영향을 덜 받는 변수인 **주택의 면적은 독립변수**가 되며, 이에 영향을 받아서 변화할 수 있는 **거래가격이 종속변수**가 될 것이다.

- 주택의 가격은 면적에도 영향을 받지만, 일조량 및 접근성 등에도 영향을 받을 수 있기 때문에, 이렇게 관측된 데이터를 바탕으로 **다차원 공간에 존재하는 데이터들을 가장 잘 설명하는 수학 함수를 찾는 것**이 바로 회귀분석이 해야할 일이다.

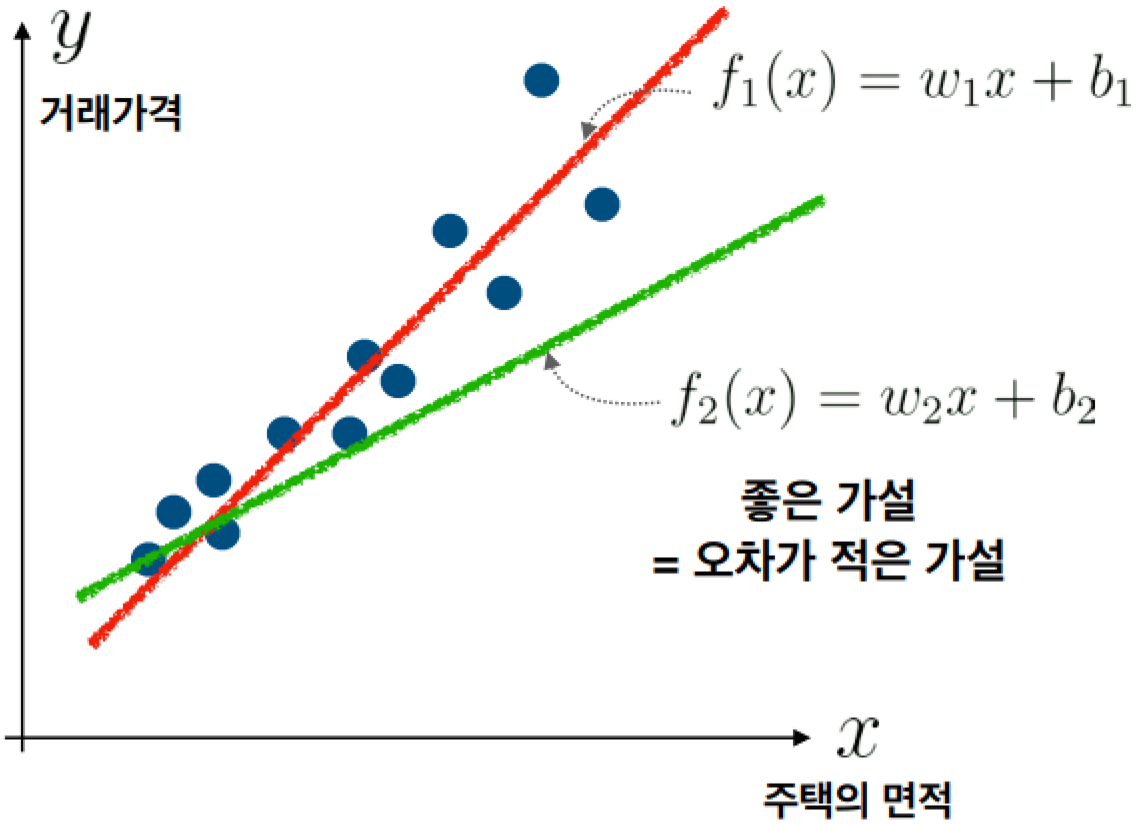
1.3 회귀분석과 독립변수, 종속변수

- 즉, 다른 표현으로 y=f(x)에서 입력 x와 출력 y를 보면서 함수 (x)를 예측하는 것을 **회귀 기법**이라고 할 수 있다.
- 예를 들어서 대상의 "면적"을 x 좌표에 입력하고 이들의 "거래가격"을 y 좌표에 매핑시킨 후, 이 상관관계를 가장 잘 설명하는 직선을 찾는 문제가 될 수 있을 것이다.
- 그림을 보면 x로 나타낸 특정한 지역의 주택면적과 거래가격 y를 확인한 결과인데 파란색 점들이 측정값들(데이터)이다.



1.3 회귀분석과 독립변수, 종속변수

- 이 둘 사이의 상관관계가 1차 방정식으로 표현될 수 있는 선형관계라고 가정하면 주택 면적 x와 거래가격 y의 관계는 오른쪽 그림과 같이 y=wx+b로 표현될 것이다.
- 기울기 w과 절편 b를 어떻게 정하는가에 따라 $f_1(x)=w_1x+b_1$ 혹은, $f_2(x)=w_2x+b_2$ 로 표현할 수 있을 것이다.
- 이때 데이터에 숨겨진 관계를 표현하고, 종속변수가 어떤 값을 가질지 예측하는 $f_1(x)$와 $f_2(x)$를 **가설hypothesis** 혹은 모델이라고 부른다.



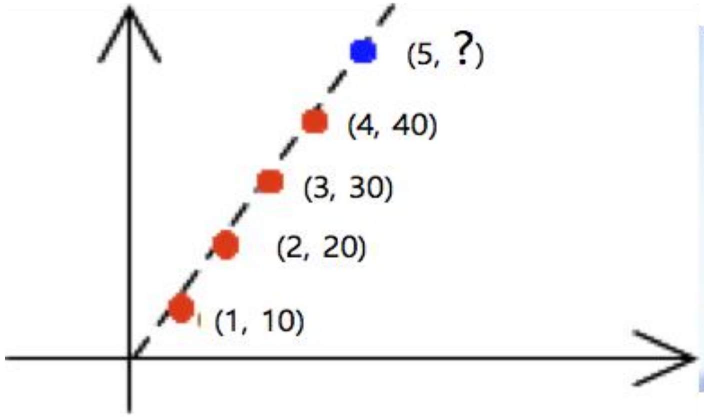
- 그림에는 두 개의 가설이 나타나 있다. 어떤 가설이 더 좋은 것일까? 그것은 **오차error**가 작은 가설이다. 따라서, $f_1(x)$ 이 $f_2(x)$보다 더 나은 가설이 될 것이다.

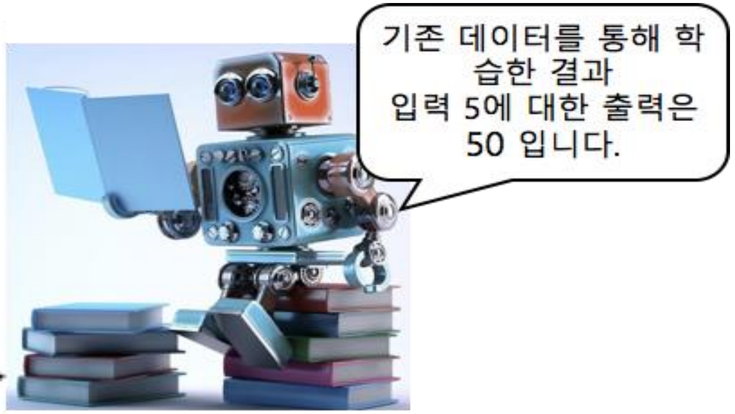
1.3 회귀분석과 독립변수, 종속변수

- 이 회귀 문제에서 y=f(x) 함수의 입력 x에 대응되는 실수 y들이 주어지고 추정한 함수 f( )가 만들어 내는 오차를 측정하여, 이 **오차를 줄이는 방향으로 함수의 계수 w b , 를 최적화하는 과정을 수행한다면 이는 톰 미첼이 정의한 기계학습으로 볼 수 있다.**
- 이때 작업 T는 독립변수에 대응하는 종속변수를 추정하는 일이며, 주어진 데이터가 경험 E에 해당한다.
- 성능 척도 P는 예측한 값 $\hat{y}$과 데이터로 제공되는 목표값 y의 차이가 작을수록 높은 점수를 부여한다.

1.3 회귀분석과 독립변수, 종속변수

- (x, y) 형태의 입력 데이터로 점 (1, 10), (2, 20), (3, 30), (4, 40)들이 주어져 있다고 하자. 컴퓨터는 x 값에 대하여 y 값이 $y = 10x$의 방정식으로 표현될 수 있는 데이터라는 것을 아직 모르는 상태이다. 주어진 데이터 4개를 학습하여 학습이 끝난 후에 x=5를 입력하면 컴퓨터가 50이라는 답을 할 수 있도록 만들고 싶다.
- 입력된 값을 바탕으로 컴퓨터가 스스로 이 입력을 설명할 수 있는 가장 좋은 함수를 찾는 것이 바로 지도학습이며, 이 문제는 지도학습 중에서도 **회귀분석**regression이라 할 수 있다.





1.4 사이킷런을 이용한 선형 회귀

- **사이킷런**scikit-learn은 2007년도 구글 하계 코드 프로젝트 모임에 참여한 몇몇 개발자들이 중심이 되어 시작된 라이브러리다.
- 머신러닝을 위해서는

1) 특징과 레이블(선택사항)로 이루어진 데이터

2) 데이터를 바탕으로 동작이 결정되는 모델

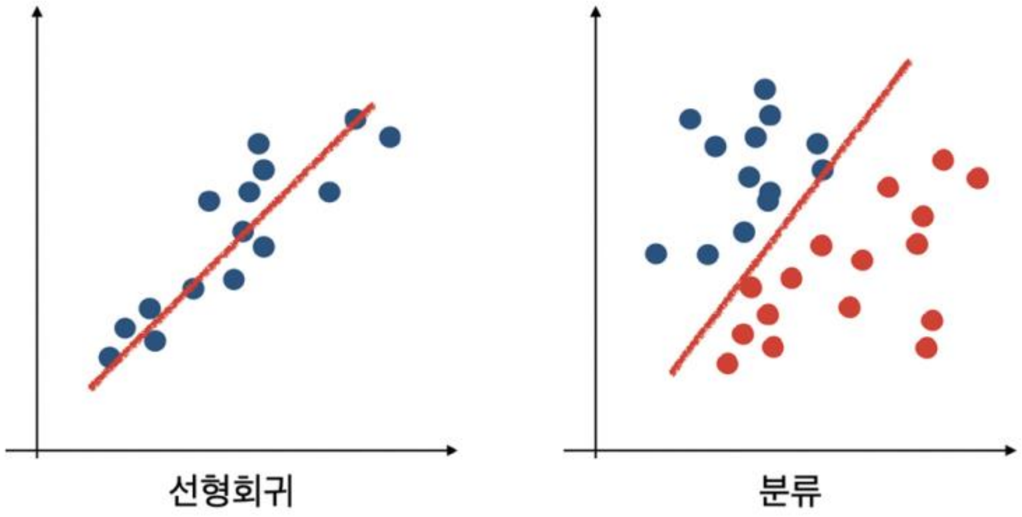
3) 모델을 위한 적절한 **하이퍼파라미터**hyperparameter

4) 학습을 위한 훈련단계

5) 검증의 여러 단계가 필요하다.

1.4 사이킷런을 이용한 선형 회귀

- 사이킷런은 지도 학습, 비지도 학습을 위한 다양한 모델을 제공하며, 이 모델을 위한 시각화 도구, 교차 검증 도구들까지 매우 광범위한 기능을 제공한다.
- 가장 단순한 모델 중의 하나인 선형 회귀에 대하여 살펴보고 사이킷런으로 구현해볼 것이다. 그리고 다음 시간에는 분류 문제를 상세히 다룰 것이다.



1.4 사이킷런을 이용한 선형 회귀

- 데이터를 학습시킬 때, 데이터를 원형 그대로 사용하는 경우도 있지만 일반적으로는 데이터에서 어떤 특성을 추출하여 이것으로 학습시키고 테스트하게 된다. 그렇다면 **특징, 특성**features이란 무엇인가?
- 특징이란 **관찰되는 현상에서 측정할 수 있는 개별적인 속성**을 의미한다.
- 리고 기계에게 이 현상을 학습하게 한다는 것은 이 특징을 입력으로 사용하여 학습한다는 것을 의미한다.
- Y=f(x)의 함수를 찾는다고 할 때, 입력 데이터로 사용되는 x가 바로 특징이다.
- 기계학습에서 특징이라는 것은 학습의 결과를 결정하는 데에 영향을 미치는 입력 데이터라고 할 수 있다.

1.4 사이킷런을 이용한 선형 회귀

<기계학습에서 다룰 수 있는 특징(특성)의 예>

- 사람의 **키**와 **몸무게** : 일반적으로 키가 큰 사람이 몸무게가 더 많이 나가는 경우가 많다. 키와 몸무게는 사람의 특징을 표현하기 위한 좋은 특성이 될 수 있다.
- 개의 **몸통 길이**와 **높이** : 말티즈와 같은 작은 개와 사모예드 같은 큰 개를 구분하기 위한 방법으로 개의 몸통 길이와 높이를 입력으로 주고 학습을 시킨다면 말티즈와 사모예드를 잘 구별할 수 있게 될 것이다.
- **주택 가격**과 **주택의 면적** : 주택의 가격에 영향을 주는 특징으로는 주택의 면적, 지하철역과의 거리, 마트까지의 거리, 주택의 건축연도, 화장실의 수와 같은 것들이 있을 것이다.

1.5 선형 회귀 모델의 계수와 절편

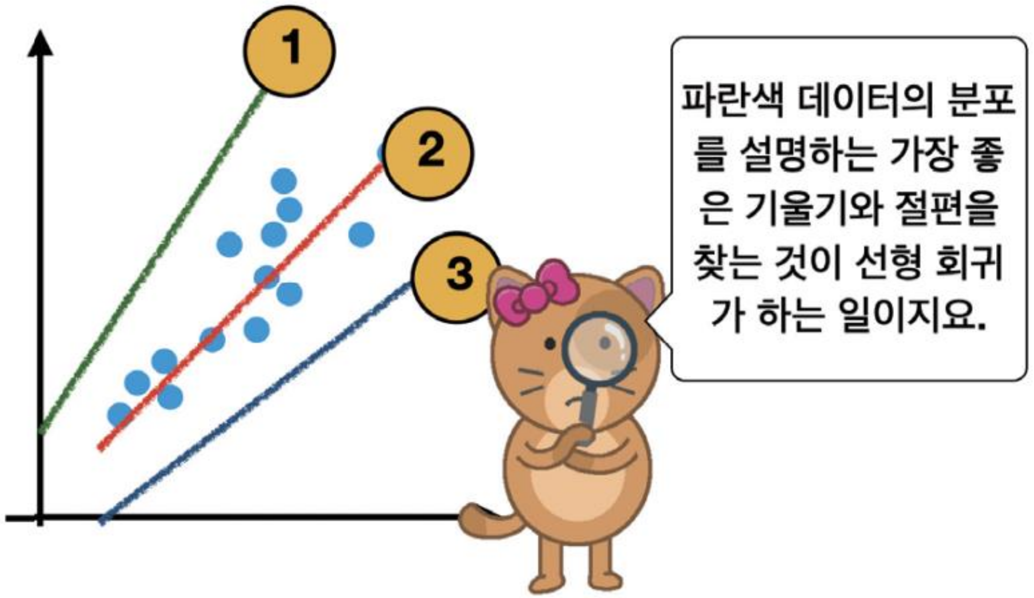
- 선형 회귀는 임의의 독립 변수 x와 이 변수에 따른 종속 변수 y와의 상관관계를 모델링하는 기법으로, 이 두 변수의 관계를 알아내거나 이 모델을 이용하여 y가 없는 x 값에 대해 y를 예측하는 데 사용할 수 있다.
- 우선 가장 간단한 모델로 이차원 평면상에 있는 직선의 방정식을 생각해 보면, 이 직선의 방정식은 기본적으로 다음과 같다.

$$y = mx + b$$

- 여기서 m은 직선의 기울기이고 입력 변수 x에 곱해지는 **계수**coefficient이다.
- 그리고, x와 관계없이 y에 영향을 주는 값 b는 **절편**intercept이다.
- 절편은 y=mx 라는 회귀선을 위 또는 아래로 얼마나 평행이동 시킬지를 결정한다.

1.5 선형 회귀 모델의 계수와 절편

- 기본적으로 선형 회귀 알고리즘은 데이터를 설명하는 가장 적절한 기울기와 절편값을 찾는 것이다.
- x 변수는 데이터 특성이므로 변경할 수 없고 우리가 제어할 수 있는 값은 기울기와 절편이다.
- 기울기와 절편의 값에 따라 여러 개의 직선이 있을 수 있다.



- 기본적으로 선형 회귀 알고리즘은 데이터 요소에 여러 직선을 맞추어 본 후에 가장 적은 오류를 발생시키는 직선을 반환한다.
- 그림을 살펴보면 ①, ②, ③ 중에서 **②가 가장 적은 오류를 발생시키는 직선**이라고 볼 수 있다.

1.5 선형 회귀 모델의 계수와 절편

- 이 개념은 2개 이상의 변수가 있는 경우까지 확장될 수 있다. 이를 다중 회귀분석이라고 한다.
- 예를 들어 주택의 면적, 침실 수, 해당 지역의 사람들의 평균 소득, 주택의 노후화 등을 기준으로 **주택 가격을 예측해야 하는 시나리오**를 생각해 보자.
- 이 경우 종속 변수 y는 여러 독립 변수에 종속된다. p + 1 개의 독립 변수가 포함된 **다중 회귀모델**은 다음과 같이 나타낼 수 있다.

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

변수가 2개 이상인 경우 다중 선형 회귀라 한다.

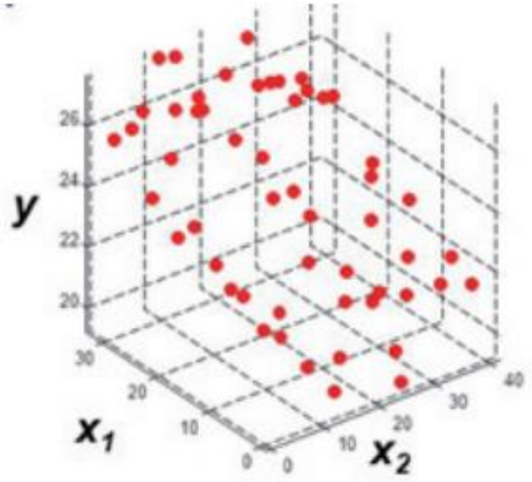
1.5 선형 회귀 모델의 계수와 절편

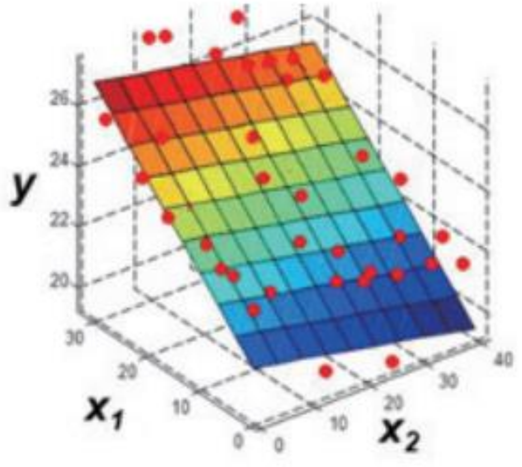
- 이 개념은 2개 이상의 변수가 있는 경우까지 확장될 수 있다. 이를 다중 회귀분석이라고 한다.
- 예를 들어 주택의 면적, 침실 수, 해당 지역의 사람들의 평균 소득, 주택의 노후화 등을 기준으로 **주택 가격을 예측해야 하는 시나리오**를 생각해 보자.
- 이 경우 종속 변수 y는 여러 독립 변수에 종속된다. p + 1 개의 독립 변수가 포함된 **다중 회귀모델**은 다음과 같이 나타낼 수 있다.

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

1.5 선형 회귀 모델의 계수와 절편

- 여기서 w와 x는 모두 벡터이고 w_0 을 제외한 $w=(w_0, w_1, \ldots, w_p)$ 를 계수, w_0 를 절편이라고도 한다.
- 이것은 사실 평면의 방정식이다. 2차원 공간에서 선형 회귀 모형은 직선이고 3차원에서는 평면이고, 3차원 이상에서는 **초평면hyperplane**이다.
- 아래 그림은 2개의 변수를 가진 입력 데이터와 레이블값 사이의 관계를 잘 표현하는 평면을 선형 회귀로 찾은 결과이다.





1.5 선형 회귀 모델의 계수와 절편

- 다중 회귀 모델의 예제
- [프로그램]: iris 데이터셋 읽기

프로그램 1 iris 데이터셋 읽기

```
01 from sklearn import datasets
02
03 d=datasets.load_iris()      # iris 데이터셋을 읽고
04 print(d.DESCR)              # 내용을 출력
```

- 01행: sklearn 모듈의 datasets 클래스를 불러옴
- 03행: load_iris 함수를 호출해 iris 데이터셋을 읽어 객체 d에 저장
- 04행: 객체 d의 DESCR 변수를 출력

1.5 선형 회귀 모델의 계수와 절편

```
Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================
:Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
...
```

150개의 샘플

네 개의 특징(feature)

세 개의 부류

그림 iris의 세 가지 품종(왼쪽부터 Setosa, Versicolor, Virginica)

sepal(꽃받침)
petal(꽃잎)

<네 개의 특징>
꽃받침 길이(sepal length)
꽃받침 너비(sepal width)
꽃잎 길이(petal length)
꽃잎 너비(petal width)

1.5 선형 회귀 모델의 계수와 절편

- [프로그램]: iris 데이터셋 내용 살피기

프로그램 2 iris의 내용 살펴보기

```
for i in range(0,len(d.data)):         # 샘플을 순서대로 출력
    print(i+1,d.data[i],d.target[i])
```

```
1 [5.1 3.5 1.4 0.2] 0
2 [4.9 3.  1.4 0.2] 0
3 [4.7 3.2 1.3 0.2] 0
4 [4.6 3.1 1.5 0.2] 0
...
51 [7.  3.2 4.7 1.4] 1
52 [6.4 3.2 4.5 1.5] 1
53 [6.9 3.1 4.9 1.5] 1
54 [5.5 2.3 4.  1.3] 1
...
101 [6.3 3.3 6.  2.5] 2
102 [5.8 2.7 5.1 1.9] 2
103 [7.1 3.  5.9 2.1] 2
104 [6.3 2.9 5.6 1.8] 2
...
```

Setosa : 0
Versicolor : 1
Virginica : 2 로 레이블 되어 있다.

d.target(레이블)

d.data(특징 벡터)

1.5 선형 회귀 모델의 계수와 절편

- 샘플을 특징 벡터와 레이블로 표현
  - 특징 벡터는 **x**로 표기(d는 **특징의 개수**에서 **특징 벡터의 차원**이라 부름)

    특징 벡터: $\mathbf{x}=(x_1,x_2,\cdots,x_d)$

  - d가 4이고 가중치가 w인 경우의 선형 결합

    $y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_0 \times 1$

  - 레이블은 0,1,2,…,c-1의 값 또는 1,2,…,c-1,c의 값 또는 원핫 코드
    - 원핫 코드는 한 요소만 1인 이진열
      - 예) Setosa는 (1,0,0), Versicolor는 (0,1,0), Virginica는 (0,0,1)로 표현

	특징 벡터 $\mathbf{x}=(x_1, x_2,\cdots, x_d)$	레이블(참값) y
샘플 1:	(5.1, 3.5, 1.4, 0.2)	0
샘플 2:	(4.9, 3.0, 1.4, 0.2)	0
…	…	…
샘플 51:	(7.0, 3.2, 4.7, 1.4)	1
샘플 52:	(6.4, 3.2, 4.5, 1.5)	1
…	…	…
샘플 101:	(6.3, 3.3, 6.0, 2.5)	2
샘플 102:	(5.8, 2.7, 5.1, 1.9)	2
…	…	…
샘플 n:	(5.9, 3.0, 5.1, 1.8)	2

iris 데이터셋
(n=150, d=4)

그림 일반적인 데이터셋 표현 방법(iris 데이터셋 예시)

1.5 선형 회귀 모델의 계수와 절편

- 통계학에서는 보통 예측값을 나타낼 때 $\hat{y}$와 같이 나타내는데, 이 책에서는 이러한 표기법을 그대로 사용할 것이다.
- 선형 회귀분석을 위해서는 다음과 같은 4가지의 기본 가정이 필요하다.
- **선형성**: 독립변수와 종속변수 간의 분포 관계가 선형의 관계를 가진다.
- **독립성**: 독립성은 다중 회귀분석의 중요한 기본 가정으로 독립변수와 다른 독립변수 간의 상관관계가 적을 경우 선형 회귀 모델의 예측력이 좋아진다.
- **등분산성**: 분산이란 데이터의 분포 정도에 대한 척도인데, 데이터가 특정한 패턴 없이 고르게 분포하는 것이, 특정한 좁은 구간에만 집중해서 분포하는 것보다 더 나은 예측을 보인다.
- **정규성**: **잔차residual**란 회귀직선과 관측값과의 차이인데, **오차error**라고도 한다. 이 차이가 정규성을 만족해야 한다.

1.6 간단한 선형 회귀를 수행해 보자

- 이번 절에서는 사이킷런 라이브러리를 사용하여 회귀 함수를 구현하는 방법을 살펴볼 것이다.
- 사이킷런을 코드에 가져오기 위해서는 sklearn이라는 이름으로 가져와야 한다.
- 선형 회귀를 위해 가장 먼저 해야 할 작업은 사이킷런 라이브러리와 넘파이를 코드에 import시키는 일이다.
- 선형 회귀를 구현하기 위해 다음과 같이 선형 모델 linear_model을 import한 뒤에 LinearRegression() 생성자를 통해 선형 회귀 모델을 생성한다. 이 선형 회귀 모델을 참조하는 변수는 regr로 지정하도록 하자.

```
import numpy as np
from sklearn import linear_model   # scikit-learn 모듈을 가져온다

regr = linear_model.LinearRegression()
```

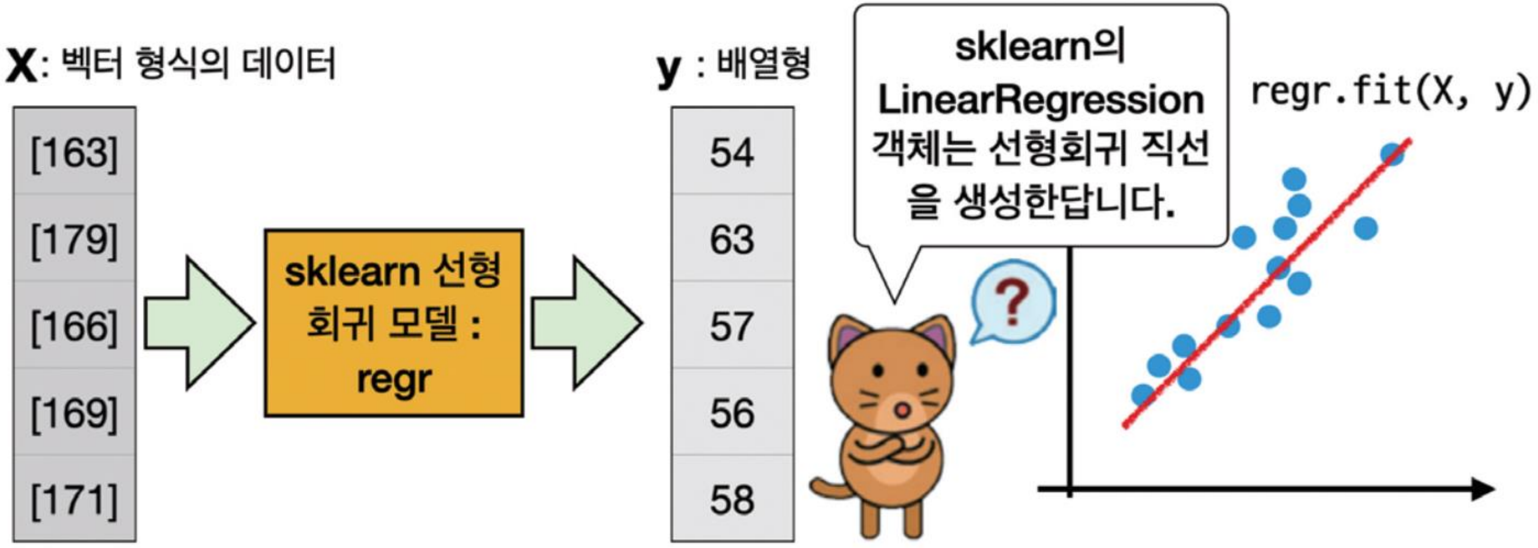
1.6 간단한 선형 회귀를 수행해 보자

- 이제 선형 회귀를 위한 입력 데이터 집합 X를 만들도록 하자.
- 입력 데이터는 [[163], [179], [166], [169], [171]]과 같은 2차원 리스트로 만들도록 한다.
- 다음으로 정답에 해당하는 y 변수를 [54, 63, 57, 56, 58]과 같이 초기화하도록 하자.
- 이제 이 데이터를 이용하여 선형 회귀 학습을 시작해 보자. regr.fit(X, y)와 같이 선형 회귀 모델에 입력과 출력을 지정하면 된다.

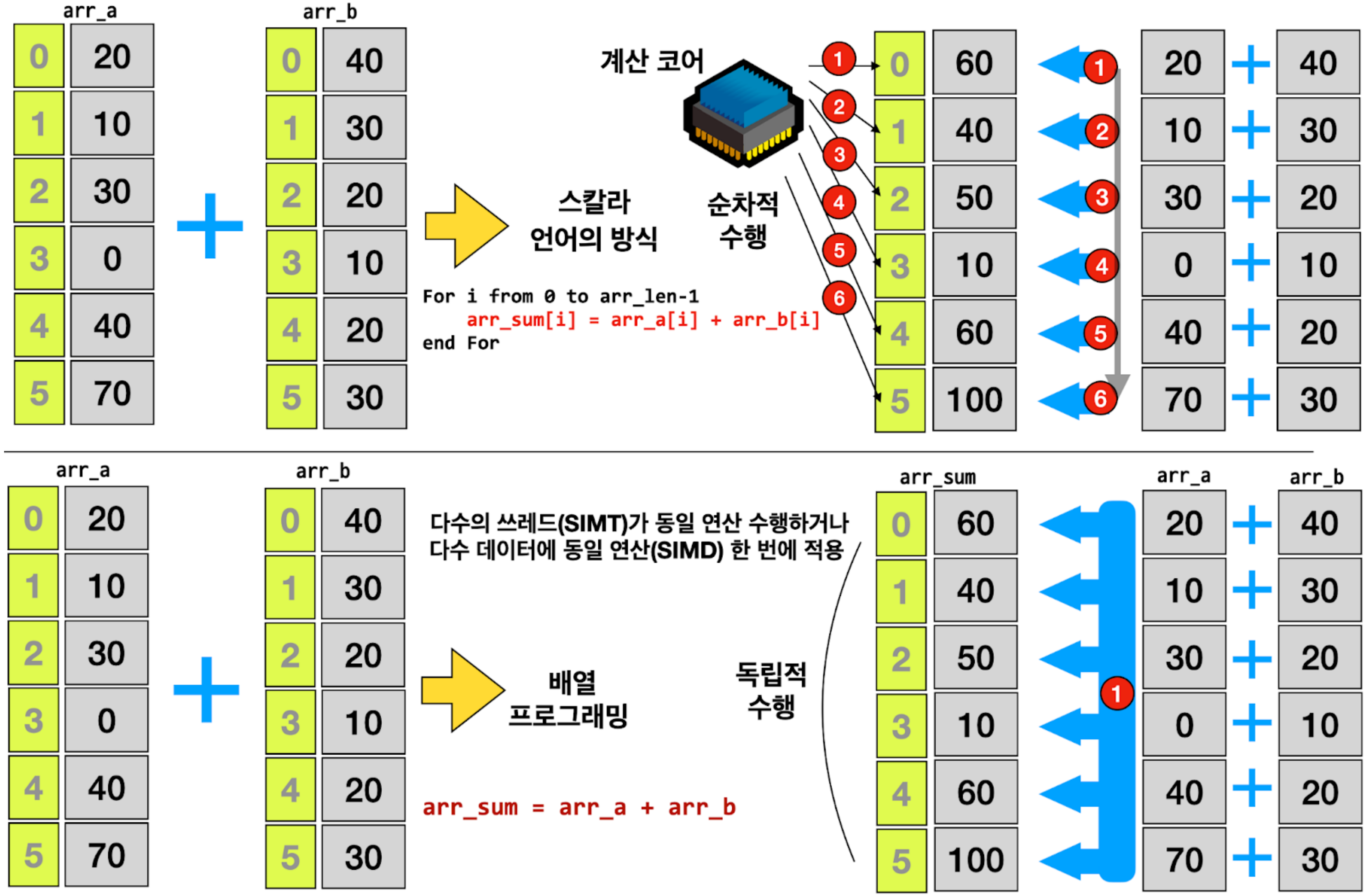
```
X = [[163], [179], [166], [169], [171]]
y = [54, 63, 57, 56, 58]
regr.fit(X, y)
```

1.6 간단한 선형 회귀를 수행해 보자

- 여기서 주의할 점은 학습 데이터는 **반드시 2차원 배열**이어야 한다는 점이다. 그 이유는 사이킷런의 LinearRegression() 모델은 **다중 회귀분석을 실시하기 위해서** 설계되었기 때문이다.
- 이 때문에 X의 각 항목을 스칼라값이 아닌, 다수의 독립 변수를 포함하는 벡터로 간주한다. 따라서 입력의 차원이 1차원인 경우에도 163이 아닌 [163]과 같은 배열 형태로 만들어야 한다. 그러나, y 값은 목표값으로 1차원 배열형 자료를 사용한다.



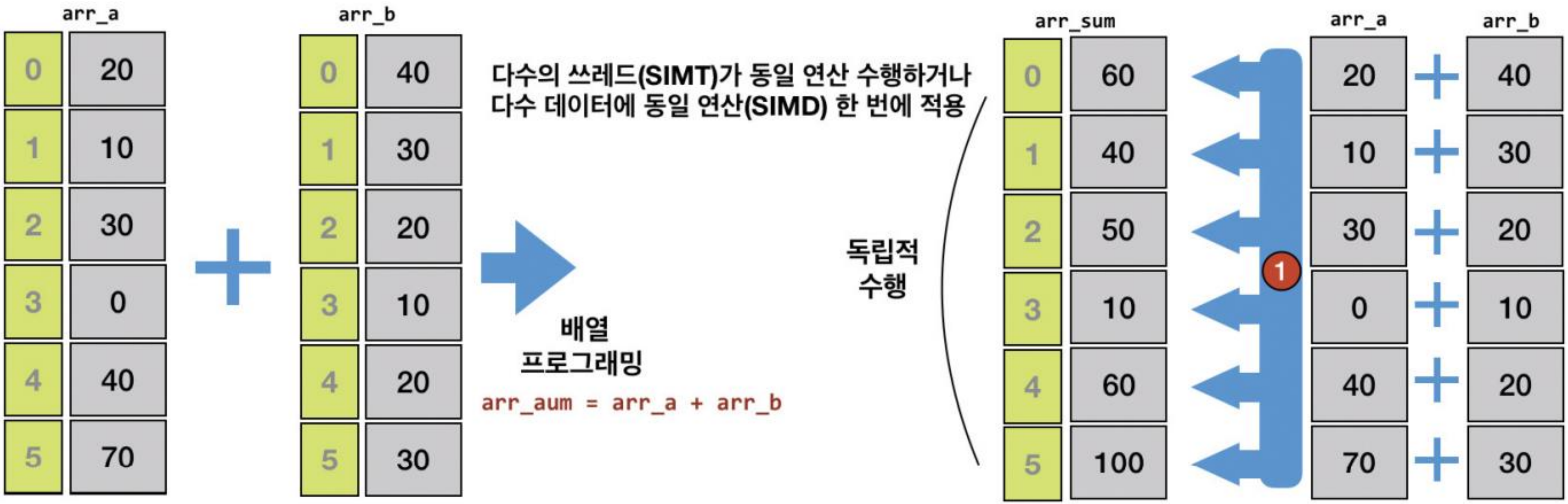
1.6 간단한 선형 회귀를 수행해 보자



*GPU는 다수의 **쓰레드**thread를 만들어 동일한 연산을 수행하게 하는데, 이러한 구조를 **단일 명령어 다중 쓰레드**single instruction multiple thread:SIMT 방식이라고 한다.

*CPU는 GPU와 달리 쓰레드를 생성하지 않고, 하나의 연산이 다수의 데이터에 동시에 적용되도록 하는 **단일 명령어 다중 데이터**single instruction multiple data:SIMD 방식을 사용한다.

1.6 간단한 선형 회귀를 수행해 보자



단일 명령어 다중 쓰레드single instruction multiple thread:SIMT
단일 명령어 다중 데이터single instruction multiple data:SIMD

1.6 간단한 선형 회귀를 수행해 보자

- 이제 이 직선의 식과 선형 회귀 직선이 실제 데이터를 얼마나 잘 설명하는 모델인가를 구해보도록 하자.

```
coef = regr.coef_             # 직선의 기울기
intercept = regr.intercept_   # 직선의 절편
score = regr.score(X, y)      # 학습된 직선이 데이터를 얼마나 잘 따르나
                         #round() 함수 지정된 소수점 자리에서 반올림한 값 리턴
print("y = {}* X + {:.2f}".format(coef.round(2), intercept))
print("데이터와 선형 회귀 직선의 관계점수: {:.1%}".format(score))
```

```
y = [0.53]* X + -32.50
데이터와 선형 회귀 직선의 관계점수: 91.9%
```

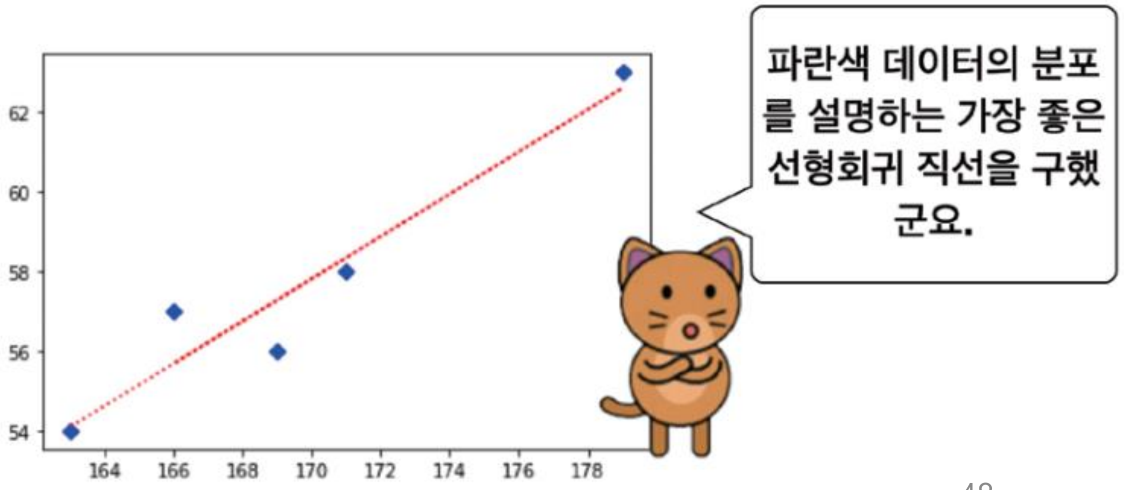
- 직선의 기울기는 regr 모델의 coef_ 속성값으로 얻을 수 있으며, 직선의 절편은 intercept_ 속성값으로 얻을 수 있다. score() 메소드를 통해서는 이 모델의 점수를 알아볼 수 있다.

1.7 데이터를 시각화하고 차원을 증가시키자

- 앞 절의 데이터를 다음과 같은 방식으로 시각화해 본다면 보다 더 직관적으로 이해하기 좋을 것이다.

```
import matplotlib.pyplot as plt

# 학습 데이터와 y 값을 산포도로 그린다.
plt.scatter(X, y, color='blue', marker='D')
# 학습 데이터를 입력으로 하여 예측값을 계산한다.
y_pred = regr.predict(X)
# 계산된 기울기와 y 절편을 가지는 점선을 그려보자
plt.plot(X, y_pred, 'r:')
```

#marker='D' 는 다이아몬드



1.7 데이터를 시각화하고 차원을 증가시키자

- 이제 이 모델에 다음과 같이 키가 167인 동윤이의 키를 넣어서 그 추정값을 출력해 보자(출력 데이터의 차원을 일단 무시하자).
- 이를 위하여 predict()라는 메소드를 사용하는 것을 볼 수 있다.

```
unseen = [[167]]
result = regr.predict(unseen)
print('동윤이의 키가 {}cm 이므로 몸무게는 {}kg으로 추정됨'.format(\
                    unseen, result.round(1)))
```

```
동윤이의 키가 [[167]]cm 이므로 몸무게는 [56.2]kg으로 추정됨
```

- format() 함수는 중괄호로 된 부분을 대입시켜주는 역할

1.7 데이터를 시각화하고 차원을 증가시키자

- 이제 여자/남자의 체중 차이를 반영한 선형 회귀 모델을 생성해 보도록 하자.
- 이를 위하여 동윤이네 반 학생들의 키와 몸무게를 다음과 같이 남학생 8명, 여학생 8명으로 나누어서 측정한 데이터를 새로 만들고 선형 회귀 모델에 적용시켜 보도록 하자.

남학생								
키	168	166	173	165	177	163	178	172
몸무게	65	61	68	63	68	61	76	67

여학생								
키	163	162	171	162	164	162	158	173
몸무게	55	51	59	53	61	56	44	57

1.7 데이터를 시각화하고 차원을 증가시키자

- 남학생과 여학생을 구분해야 하므로 간단하게 남학생은 0, 여학생은 1의 구분 값을 입력값에 넣어주도록 하자. 따라서, 입력 데이터의 차원을 2차원으로 증가시켜서 키가 167cm인 남학생은 [167, 0]로, 여학생은 [167, 1]이 되도록 데이터를 만들자.

```
from sklearn import inear_model
regr = linear_model.LinearRegression()
X = [[168, 0], [166, 0], [173, 0], [165, 0], [177, 0], [163, 0], \
     [178, 0], [172, 0], [163, 1], [162, 1], [171, 1], [162, 1], \
     [164, 1], [162, 1], [158, 1], [173, 1], ]     # 2차원 입력 데이터
y = [65, 61, 68, 63, 68, 61, 76, 67, 55, 51, 59, 53, 61, 56, 44, 57]
regr.fit(X, y)          # 학습시키기
print('계수 :', regr.coef_ )
print('절편 :', regr.intercept_)
print('점수 :', regr.score(X, y))
print('동윤이와 은지의 추정 몸무게 :', regr.predict([[167, 0], [167, 1]]))
```

LinearRegression의 score() 메소드는 참값과 예측값의 차이인 결정 계수 R^2를 반환한다. 결정 계수 값이 1이면 현재 모델이 완벽한 모델이라는 것을 의미하며, 0이면 현재 모델이 변수를 설명하는데 전혀 도움이 되지 않음을 의미한다

```
계수 : [ 0.74803397 -7.23030041]
절편 : -61.2277838943063 84
점수 : 0.8425933302504423
동윤이와 은지의 추정 몸무게 : [63.69388959 56.46358918]
```

167 → regr.predict() → 56.5

1.7 데이터를 시각화하고 차원을 증가시키자(고급)

- 선형 회귀 모델의 성능을 평가하는 척도
  - MSE(평균 제곱 오차) :

$$E_{mse} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

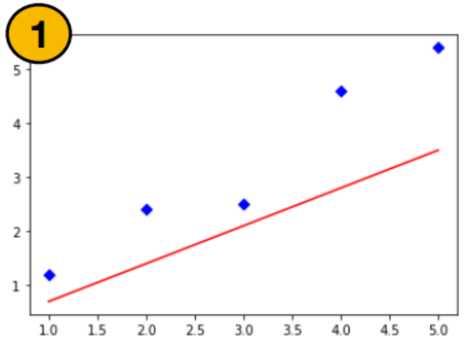
    - 예측값($\hat{y}_i$)과 실제값(y_i)의 차이의 제곱을 구하고 이 값을 데이터의 개수 m으로 나눈다
  - RMSE(평균 제곱근 오차)
    - $E_{rmse} = \sqrt{E_{mse}}$
    - 평균 제곱 오차에 제곱근을 취한다
    - MSE, RMSE 모두 절대적인 숫자로 나타나는데 이것만으로 모델이 좋은지 나쁜지를 판단하기가 어렵다.

1.7 데이터를 시각화하고 차원을 증가시키자(고급)

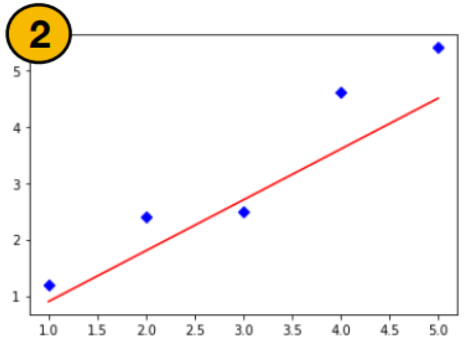
- 선형 회귀 모델의 성능을 평가하는 척도
  - R-square(R^2) : sklearn의 LinearRegression 모델의 score 메소드가 사용하는 척도
    - **결정 계수**라고 한다.
    - 이 척도는 전체 분포가 있을 때 모델에 의해 설명되는 정도를 의미한다
    - $R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i - \widehat{y_i})^2}{\sum_{i=1}^{m}(y_i - \overline{y_i})^2}$ = 1 - $\frac{unexplained\ variance}{total\ varianc\ of\ Y}$
    - 이 척도가 1이면 모델의 설명이 완벽함을 의미함
    - 0이면 모델이 완전히 어긋남을 의미함

1.8 가설의 정확도를 평가하는 오차

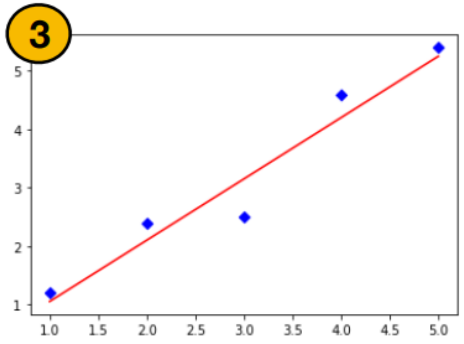
- 이제 데이터의 분포를 파란색 점으로, 선형방정식을 빨간색 직선으로 그려보면 그림과 같이 나타나서 이 방정식이 데이터의 분포를 잘 설명하지 못한다는 것을 알 수 있을 것이다.
- 이제 2) y=0.9x 로 선형방정식을 사용할 경우 데이터의 분포를 이전보다 더 정확하게 설명하는 직선을 얻을 수 있을 것이며, 3) y=1.05 라는 선형방정식은 더욱 더 나은 결과를 보여주는 것을 눈으로 확인할 수 있을 것이다.



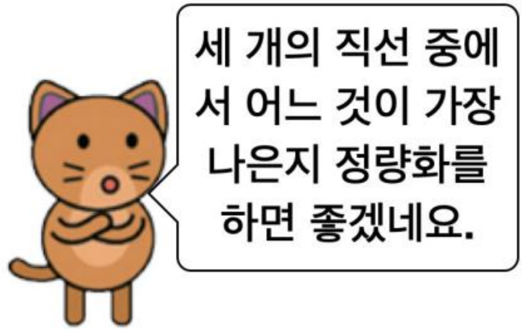
$y = 0.7x$



$y = 0.9x$



$y = 1.05x$



1.8 가설의 정확도를 평가하는 오차

- '더 나은' 선형방정식(혹은 '더 좋은' 가설)은 너무나 주관적인 표현이므로 이를 정량화하는 것이 필요한데 이때 사용되는 것이 바로 오차함수이다.
- 오차의 합을 그대로 사용하지 않고 별도의 오차함수를 사용하는 이유는 실제값이 {1, 2, 3}이고 예측값이 {1, 4, 1}로 나타날 경우 (1-1) + (2-4) + (3-1) = 0이 되는 경우가 발생하기 때문이다.

1.8 가설의 정확도를 평가하는 오차

- **평균 절대 오차**mean absolute error:MAE
- 머신러닝에서 사용 가능한 오차함수 중에서 비교적 단순한 오차함수로 예측값 $\hat{y}$과 관측값 y의 차이값의 절대값을 구한 후 이 값들의 평균값을 사용한다.
- 이 오차함수는 오차값을 그대로 보여주는 특징이 있으며 다음과 같이 정의할 수 있다.

$$E_{mae} = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}_i - y_i|$$

- 평균 절대 오차는 직관적이며 계산이 편리한 반면 다음의 문제가 있다.
- 첫째, 축적을 보정하지 않기 때문에 앞의 값의 10배에 해당하는 (10, 12), (20, 24), (30, 25), (40, 46), (50, 54) 값에 대해서 동일한 10%오차가 발생하더라도 10배의 차이가 나는 문제가 있다.
- 둘째, 절대값의 사용으로 인해 미분이 불가능한 지점이 발생한다는 문제가 있다.

1.8 가설의 정확도를 평가하는 오차

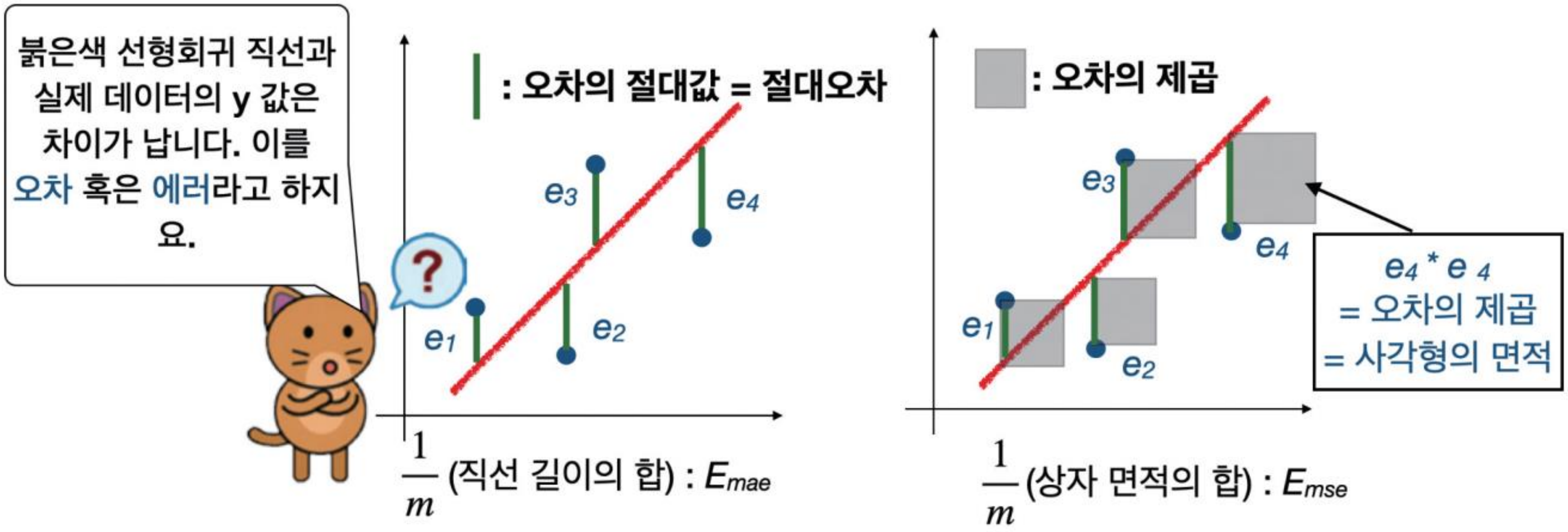
- **평균 제곱 오차**mean square error:MSE
- 머신러닝에서 사용하는 대표적인 오차 척도는 **평균 제곱 오차**이다.
- 이 방법은 예측치 $\hat{y}$ 와 정답 레이블 y 사이의 차이를 제곱하여 모두 더한 뒤에 전체 데이터의 개수 m으로 나누는 것인데, 다음과 같은 식으로 표현할 수 있다.

$$E_{mse} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

- 머신러닝의 문제를 해결하는 데 **주로 사용되는 오차는 평균 제곱 오차**로 우리는 이 오차 측정 방법이 왜 유용한지 집중적으로 살펴볼 것이다.

1.8 가설의 정확도를 평가하는 오차

- 아래 그림을 보면, 파란색 점으로 표시된 레이블과 붉은색 가설 직선의 y 값은 차이가 난다.
- 이를 오차라고 하는데, e1에서 e4까지 전체 에러의 합이 최소가 되는 모델이 가장 바람직한 모델이 될 것이며 우리는 이 직선을 찾는 것이다.



1.9 오차 함수의 구현과 파라미터 공간의 최적값

- 넘파이를 이용하여 앞서 살펴 다룬 오차 함수를 쉽게 구현할 수 있다.
- 우선 다음과 같이 [1.2, 2.4, 2.5, 4.6, 5.4]의 값을 가지는 데이터가 y에 저장되어 있고, 예측 모델이 y x =1 꼴로 되어 x가 1에서 5까지 증가할 때, y_hat이 [1, 2, 3, 4, 5]인 경우를 가정해 보자.
- 이 경우에 대해 평균 제곱 오차는 아래와 같이 구할 수 있다.
- 그리고 동일한 기능을 sklearn의 mean_squared_error() 함수를 호출하여 실행시켜 볼 수 있다.

Import numpy as np

```
# 넘파이를 이용하여 구현한 평균 제곱 오차
y = np.array([1.2, 2.4, 2.5, 4.6, 5.4])
y_hat = np.array([1, 2, 3, 4, 5])
diff = (y_hat - y) ** 2      # y_hat과 y의 차이값의 제곱
e_mse = diff.sum() / len(y)
print('평균 제곱 오차 =', e_mse)
```

```
평균 제곱 오차 = 0.19399999999999995
```

1.9 오차 함수의 구현과 파라미터 공간의 최적값

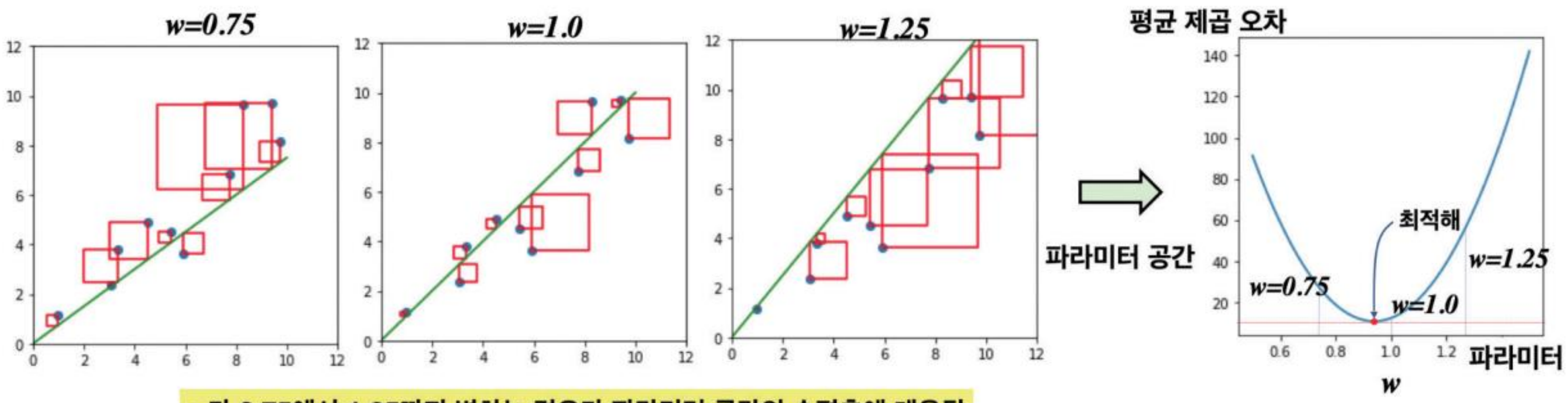
```
from sklearn.metrics import mean_squared_error

# sklearn에서 제공하는 함수를 사용해보자. 위의 결과와 동일하다.
print('평균 제곱 오차 =', mean_squared_error(y_hat, y))
```

```
평균 제곱 오차 = 0.19399999999999995
```

- 사실 오차를 제곱하는 데에는 더욱 중요한 이유가 있는데, 이것은 **오차 합 곡면의 기울기를 따라 내려가 최소 오차에 접근하기 위해서**이다.
- 양수 오차가 많으면 그 합이 무한히 커질 수도 있고, 음수 오차가 많으면 무한히 작은 값을 가질 수도 있다.
- 하지만, **오차를 제곱하면 가장 좋은 파라미터에서 최소값을 갖는 볼록한 그릇 모양의 곡면**을 만들 수 있다.

1.9 오차 함수의 구현과 파라미터 공간의 최적값

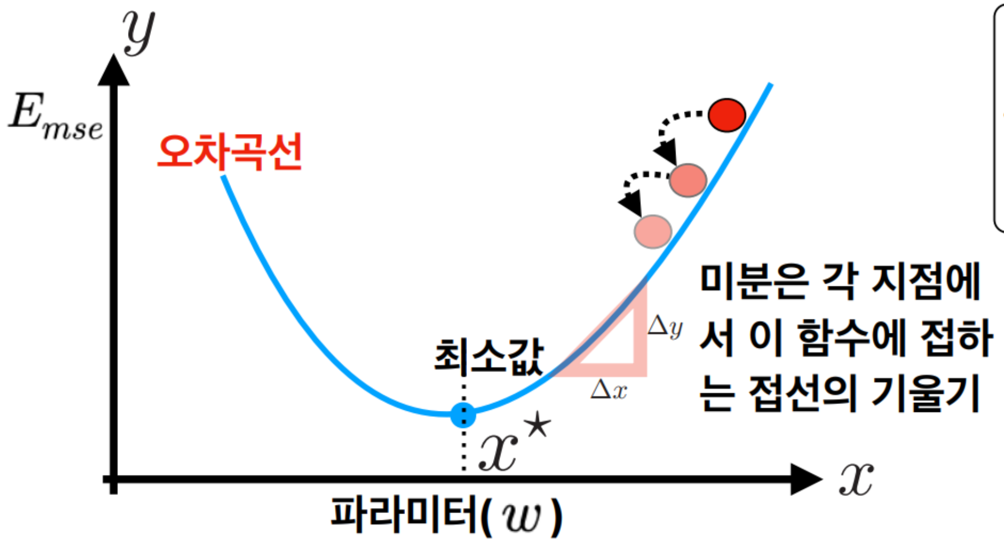
- 아래 그림의 왼쪽과 같은 데이터가 있을 때, 아래 그림의 가운데와 같이 y=wx 는 w에 따라 여러 가지 모델이 될 수 있다.
- 여기서 가장 좋은 모델은 w=1.0 인 모델인 것 같다.
- 하지만, 이것이 정말로 가장 좋은 모델인지 계산할 수 있는 방법은 무엇일까?
- 그림의 왼쪽에서는 w가 0.75에서 1.25까지 점점 커지는데, 이 값을 오른쪽 그림의 수평축에 대응시키고 각 상황에 대해 평균 제곱 오차를 수직축에 대응시켜 보도록 하자.

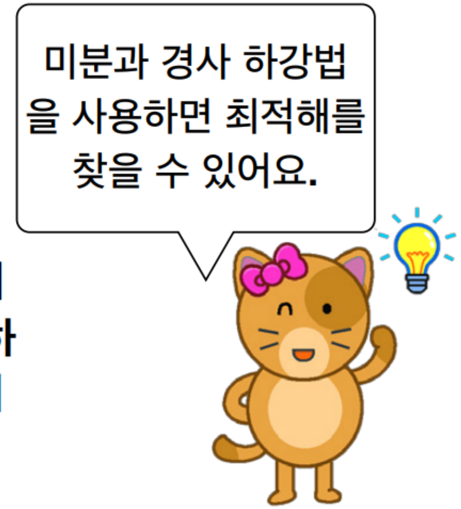


w가 0.75에서 1.25까지 변하는 경우가 파라미터 공간의 수평축에 대응됨
이때, 평균 제곱 오차는 파라미터 공간의 수직축에 대응됨

1.9 오차 함수의 구현과 파라미터 공간의 최적값

- 우리는 수학자들이 아주 좋아하는 미분이라는 도구를 사용할 것이며, 오차 곡선의 미분을 이용하여 곡선의 변화율을 구하고 이 변화율을 이용하여 최적해를 찾을 것이다.
- 그림과 같은 오차 곡선에서 최소값(혹은 최적해)을 구하기 위하여 오차 곡선 혹은 곡면의 기울기를 따라 내려가며 해를 구하는 **경사 하강법**gradient descent method이라는 방법으로 해를 찾아볼 것이다.





1.10 미분과 경사 하강법

- 머신러닝에 사용되는 여러 기법들을 이해하기 위해 필요한 수학적인 개념 중에서 가장 중요한 개념은 바로 **미분**derivative이다.
- 미분이란 순간 변화량을 구하는 것으로 다음과 같이 독립 변수값의 변화량의 비의 극한으로 구성된다.
- y=f(x) 와 같은 함수의 한 입력값 a에서의 미분 f′(a)는 다음과 같이 정의된다.

$$f'(a) = \lim_{\Delta x \to 0} \frac{f(a+\Delta x) - f(a)}{\Delta x}$$

- 함수 f(x)의 1차 미분 f′(x)는 x가 매우 조금 변화한 정도에 대해 함수값이 어떤 비로 변화하는지 알려준다. 이것을 변화율이라고 한다.
- 이러한 성질을 이용하면 머신러닝에서 매우 중요한 **최적화**optimization 작업을 할 수 있다. 최적화를 위하여 다음과 같은 용어를 명확하게 정의하도록 하자.

1.10 미분과 경사 하강법

- **목적함수**
- 다음 페이지 그림의 파란색 곡선으로 표시된 함수는 변수 혹은 파라미터를 매개변수로 갖는 함수이다.
- 이 함수를 목적함수라고 하는데, 이를 f(x)라고 할 때, 이 함수를 가장 작은 값으로 만드는 최적의 변수 x*를 찾는 일을 최적화라고 한다.
- 앞 절에서 다룬 평균 제곱 오차의 최소값을 구할 경우 평균 제곱 오차식이 목적함수가 될 것이다.

- **평균 변화율**
- 그림의 가운데에 있는 붉은 삼각형을 보면 변수 x가 Δx 만큼 변할 때, 목적함수는 Δy 만큼 변한다. 이것의 비가 이 구간의 평균 변화율이다.

1.10 미분과 경사 하강법

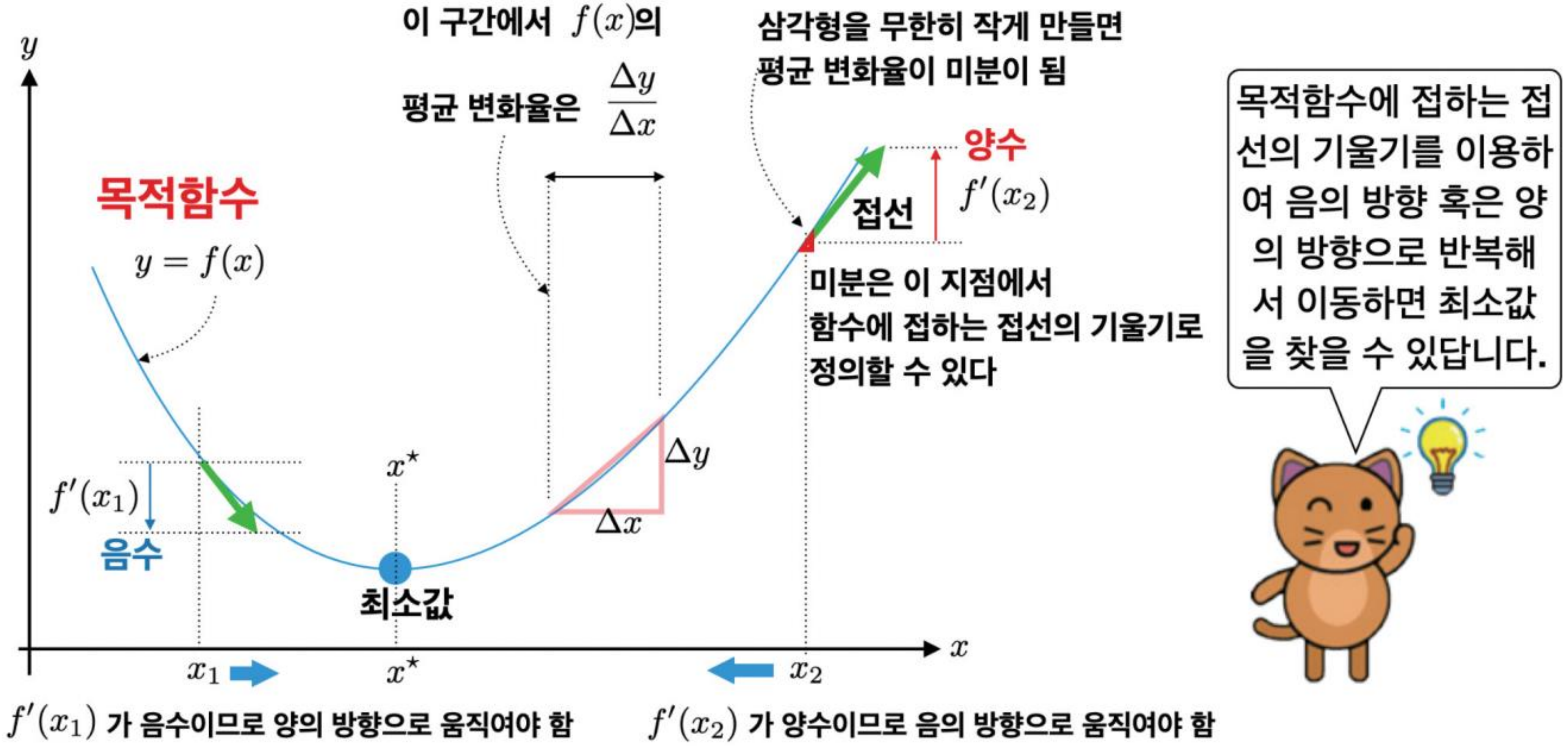
- **미분과 접선의 기울기**
- $x = x_2$ 인 지점에서 이 삼각형을 매우 작게 만들었다. 이 삼각형을 무한히 작게 만들면 이것이 바로 x_2 지점에서의 목적함수 미분이다.
- 그리고 이 값은 그 지점에서 목적함수 곡선에 접하는 선, 즉 접선의 순간변화율이다. 이것은 접선의 기울기이다. 이 값의 크기는 경사가 급할수록 더 큰 값이 된다.

- **경사 하강법**
- 접선의 기울기를 따라 기울기 부호의 반대 방향으로 조금씩 내려오면 해당 지점에서 목적함수의 값이 더 작은 쪽으로 이동할 수 있다.
- x_2 의 위치에서 x가 증가하면 y도 증가하므로 접선의 기울기는 양수이다.
- 따라서 x_2 에서 음의 방향으로 움직이면 목적함수의 값이 줄어든다. 즉 $f'(x_2)$의 반대 방향인 $-f'(x_2)$ 로 이동해야 한다.

1.10 미분과 경사 하강법

- 반대로, x_1 지점에서 미분은 음수이므로 이 위치에서는 양의 방향으로 이동해야 한다.
- 이 과정을 반복하여 최소값을 구하는 방법을 **경사 하강법**gradient descent이라고 한다.
- 경사 하강법은 반복적으로 조금씩 조금씩 최소값에 접근하는 방법인데 이 과정이 **바로 학습의 과정**이며 변화되는 변수 x의 **양의 학습률**learning rate이라고 한다.
- 경사 하강법에서의 최소값은 변화율이 0이 되는 지점으로, 변화율이란 곧 미분 값이라는 것을 그림을 통해 알 수 있다.

1.10 미분과 경사 하강법



1.10 미분과 경사 하강법(도전문제)

도전문제 (난이도 : 하)

1. 다음 함수의 미분 함수 $f'(x)$를 구하여라.

1) $f(x)=10$

2) $f(x)=3x+2$

3) $f(x)=5x^2+2x$

4) $f(x)=e^x$

2. 다음과 같이 정답값, 모델 A의 예측값, 모델 B의 예측값이 있다. 모델 A와 모델 B의 평균 절대 오차(MAE), 평균 제곱 오차(MSE)를 각각 구하여라.

정답값	모델 A	모델 B
1	0.9	0.5
2	1.3	1.9
3	3.3	3.4
4	3.8	4.4

1.10 미분과 경사 하강법(도전문제)

- 정답코드(2)

```
import numpy as np

model_A = np.array([0.9, 1.3, 3.3, 3.8])
model_B = np.array([0.5, 1.9, 3.4, 4.4])
model_true = np.array([1, 2, 3, 4])
n = 4

# 다음과 같이 차를 구하고 절대값을 취한 후 이 값들의 sum을 한다.
# 그 결과에 1/n을 한다
print('model A의 MAE =', np.abs(model_A - model_true).sum()/n )
print('model B의 MAE =', np.abs(model_B - model_true).sum()/n )
```

```
model A의 MAE = 0.32499999999999996
model B의 MAE = 0.3500000000000001
```

1.10 미분과 경사 하강법(도전문제)

- 정답코드(2)

```
# 다음과 같이 차를 구하고 절대값을 취한 후 이 값들의 평균을 구한다
# 위의 방법과 동일하지만 n으로 나누는 일을 np.mean()이 수행한다
print('model A의 MAE =', np.mean(np.abs(model_A - model_true)) )
print('model B의 MAE =', np.mean(np.abs(model_B - model_true)) )
```

```
model A의 MAE = 0.32499999999999996
model B의 MAE = 0.3500000000000001
```

```
# 다음과 같이 차들의 제곱을 구하고 이 값들의 sum을 한다
# 그 결과에 1/n을 한다
print('model A의 RMSE =', ((model_A - model_true)**2).sum()/n )
print('model B의 RMSE =', ((model_B - model_true)**2).sum()/n )
```

```
model A의 RMSE = 0.15749999999999997
model B의 RMSE = 0.14500000000000005
```

1.11 경사 하강법과 학습의 원리

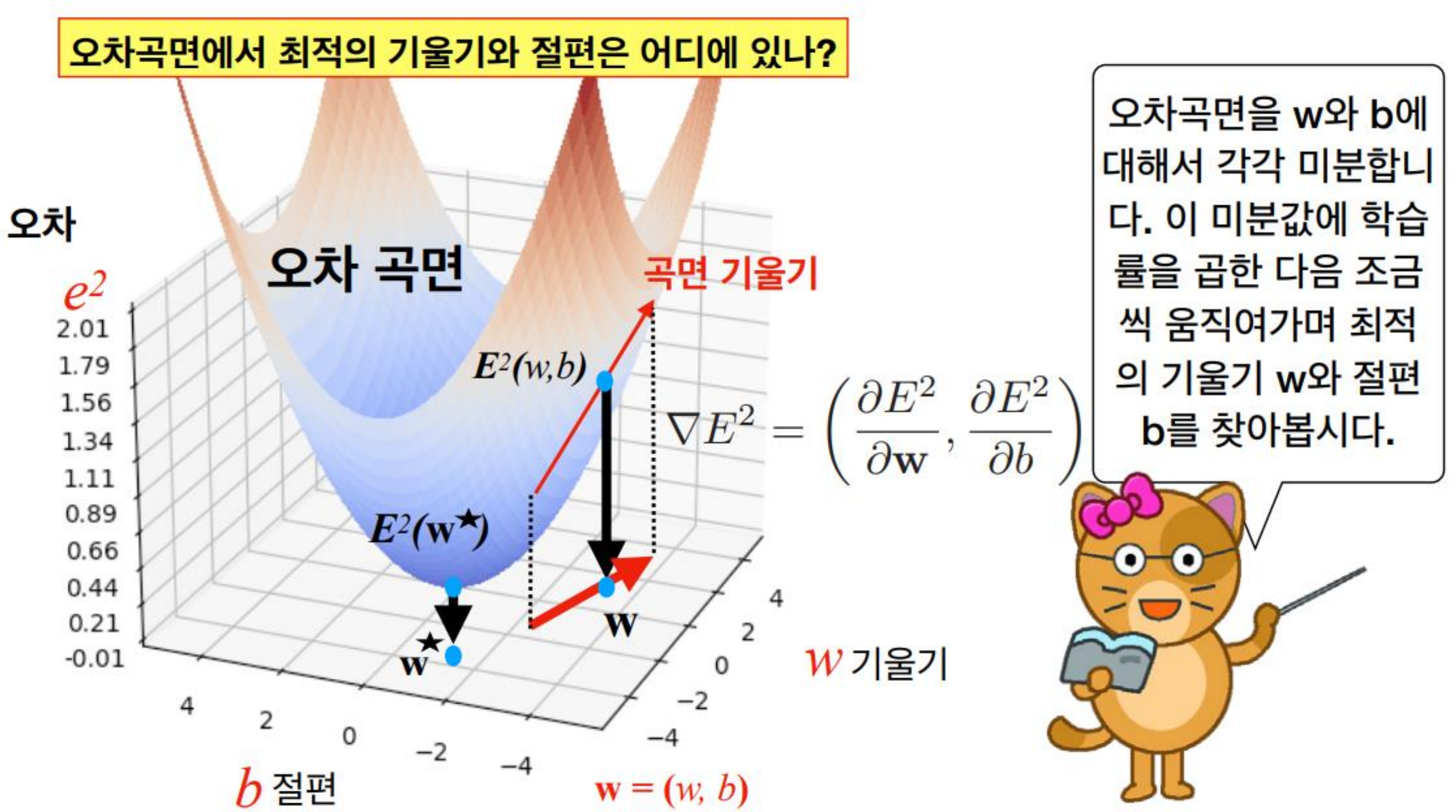
- 직선의 기울기 w와 절편 b에 의해 결정되는 오차의 제곱 $E^2(w, b)$이 그림과 같은 밥그릇 모양의 곡면이라면, 최적의 w와 b를 찾기 위한 오차 곡면의 기울기 방향은 다음과 같이 오차의 제곱값을 기울기 w와 절편 b에 대해 각각 미분하여 생성된 벡터가 될 것이다.
- 따라서 다음과 같은 식이 성립한다.

$$\nabla E^2 = \left(\frac{\partial E^2}{\partial w}, \frac{\partial E^2}{\partial b} \right)$$

$$\frac{\partial E^2}{\partial w} = \frac{\partial (wx+b-y)^2}{\partial w} = 2(wx+b-y)x = 2Ex$$

$$\frac{\partial E^2}{\partial b} = \frac{\partial (wx+b-y)^2}{\partial b} = 2(wx+b-y)\cdot 1 = 2E$$

1.11 경사 하강법과 학습의 원리



1.11 경사 하강법과 학습의 원리

- 훈련 모델을 구현하기 위하여 모델에 설정되는 학습에 사용되는 파라미터를 **하이퍼파라미터**라고 한다.
- 하이퍼파라미터에는 학습률, 훈련 반복 횟수, 가중치 초기화 값들이 될 수 있다.
- 여기서는 학습률 값을 0.005로 사용하였으며, 그리스 문자 η(에타)로 표기하였다.
- n개의 데이터 x_i 에 대한 예측 오차가 E_i 라고 할 때 다음과 같이 기울기 w와 절편 b를 오차를 이용하여 수정할 수 있다.

$$w \leftarrow w - \eta \sum_{i=1}^{n} E_i x_i , \quad b \leftarrow b - \eta \sum_{i=1}^{n} E_i$$

- 학습을 위해서는 전체 데이터를 모두 넣어서 에러를 구하는데 이렇게 전체 데이터를 한 번 사용하는 것을 1 **에폭**epoch이라고 한다.

1.11 경사 하강법과 학습의 원리

```
X = np.array([1, 4.5, 9, 10, 13])
y = np.array([0, 0.2, 2.5, 5.4, 7.3])

w, b = 0, 0    # w, b의 초기값을 0으로 두자
learning_rate, epoch = 0.005, 1000 # 학습률과 학습 횟수(에폭)
n = len(X)  # 입력데이터 개수

for i in range(epoch):       # 학습 루프
    y_pred = w*X + b         # 현재 w, b를 이용한 작업 T
    error = y_pred - y       # 성능척도 P
    w = w - learning_rate * (error * X).sum() # 경험 E로 개선
    b = b - learning_rate * error.sum()

print('w =', w.round(2), ', b =', b.round(2))
```

```
w = 0.63 , b = -1.65
```

7.12 경사 하강법과 학습률

- 이전 절의 과정은 sklearn의 LinearRegression 클래스에 구현되어 있는데 이를 다음과 같은 코드로 확인해 보도록 하자. 이전 절에서 경사 하강법으로 구한 w와 b과 같은 값이 출력되는 것을 볼 수 있다.

```
from sklearn import linear_model
import numpy as np

X = np.array([1, 4.5, 9, 10, 13])
y = np.array([0, 0.2, 2.5, 5.4, 7.3])
regr = linear_model.LinearRegression() # 절편값 b는 0으로 둔다
X = X[:, np.newaxis]
regr.fit(X, y)          # 학습

print('w =', regr.coef_.round(2), \
      ', b =', regr.intercept_.round(2))
```

```
w = [0.63] , b = -1.65
```

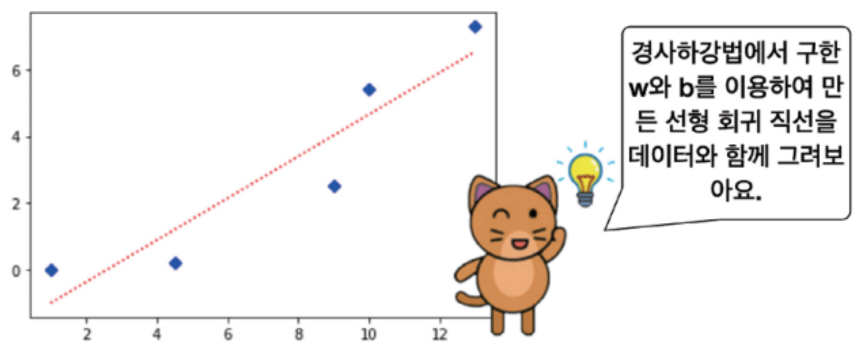
```
import numpy as np

# 배열 생성
arr1 = np.array([1, 2, 3, 4], dtype=int)
# 일차원 배열
print(arr1[np.newaxis])
print(arr1[:, np.newaxis])

# 결과
[[1 2 3 4]]
[[1]
 [2]
 [3]
 [4]]
```

***np.newaixs: np 행렬의 차원을 확장하는 함수**

1.12 경사 하강법과 학습률

- 우리가 구한 가설 함수의 w와 b가 데이터의 분포를 제대로 설명하는가 시각화하는 코드를 만들어 보자.

```
import matplotlib.pyplot as plt
import numpy as np

X = np.array([1, 4.5, 9, 10, 13])
y = np.array([0, 0.2, 2.5, 5.4, 7.3])
plt.scatter(X, y, color='blue', marker='D')
# 계산으로 구한 w, b를 이용하여 선형 회귀 직선을 그리자
y_pred = 0.63 * X - 1.65
plt.plot(X, y_pred, 'r:')
```



1.12 경사 하강법과 학습률

- 만일 학습의 하이퍼파라미터 중 하나인 학습률을 매우 작은 0.00001로 둔다면 어떤 결과가 나타날까?

68 페이지 참고

```
learning_rate, epoch = 0.00001, 1000 # 학습률과 학습횟수(에폭)

print('w =', w.round(2), ', b =', b.round(2))
```

```
w = 0.45 , b = 0.03
```

- 매 단계에서 사용해야할 학습률이 너무 작을 경우 경사를 타고 내려오는 간격이 너무나 작아서 안타깝게도 정답에 제대로 수렴하지 못하는 것을 볼 수 있다.
- 물론 학습 횟수를 1,000,000번 정도로 충분히 많이 준다면 언젠가는 정답에 수렴할 수도 있겠지만 이 경우 학습에 너무 많은 시간이 걸릴 것이다.

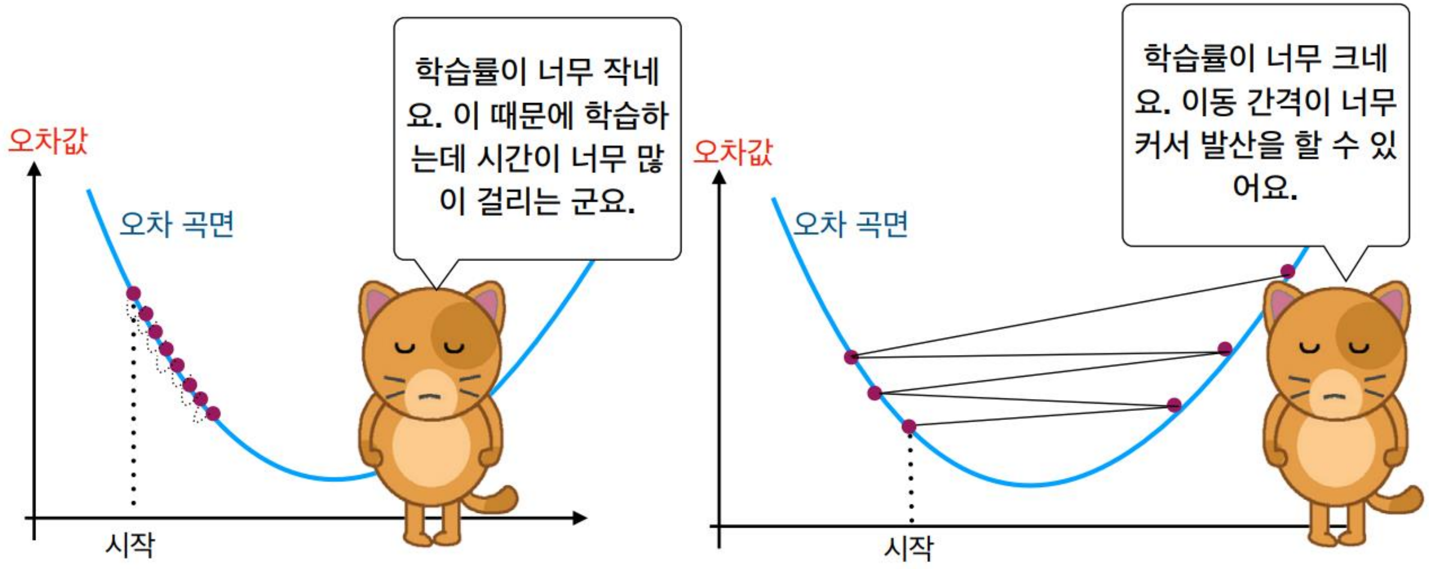
1.12 경사 하강법과 학습률

- 반대로 학습률을 1.0으로 둔다면 어떤 결과가 나올까?

68 페이지 참고

```
learning_rate, epoch = 1.0, 1000 # 학습률과 학습횟수(에폭)

print('w =', w.round(2), ', b =', b.round(2))
```

```
w = nan , b = nan
```



THANK YOU