

머신러닝

- 김백섭 : 033-248-2313, 010-5372-4186,
bskim@hallym.ac.kr
- 교재 : 핸즈온 머신러닝 2판, 2020.05
- 강의(이론) / 실습
- 도구 : Python, Numpy/Pandas/Matplotlib,
Scikit-learn, Tensorflow, Keras,
Jupyter Notebook (Colab, Anaconda)
- 기말프로젝트

인공지능이란

- Artificial Intelligence :

- The science and engineering of building **machines** that can make decisions like humans.
- It is the **ability of machines** to perform cognitive tasks like humans.



- Weak vs Strong AI

- **Weak AI** : System that emulates human cognitive abilities **in a specific task**.
- Strong AI : System with generalized human-like cognitive abilities.

Alan Turing

- 인공지능 탄생의 공헌자 : 알란 튜링(Alan Turing)

- 기계지능에 관한 초기의 의미 있는 논문 중 하나인 'Computing machinery and intelligence'을 1950년도에 작성.
- 튜링의 접근 방식은 오래전에 나온 것이긴 하지만, 기나긴 테스트를 통과하여 보편적인 것이 됨.
- 튜링은 기계와 생각의 정의를 내리지 못했고, 튜링 모방 게임(Turing imitation game)을 만듦으로써 의미론적인 논증을 피함.

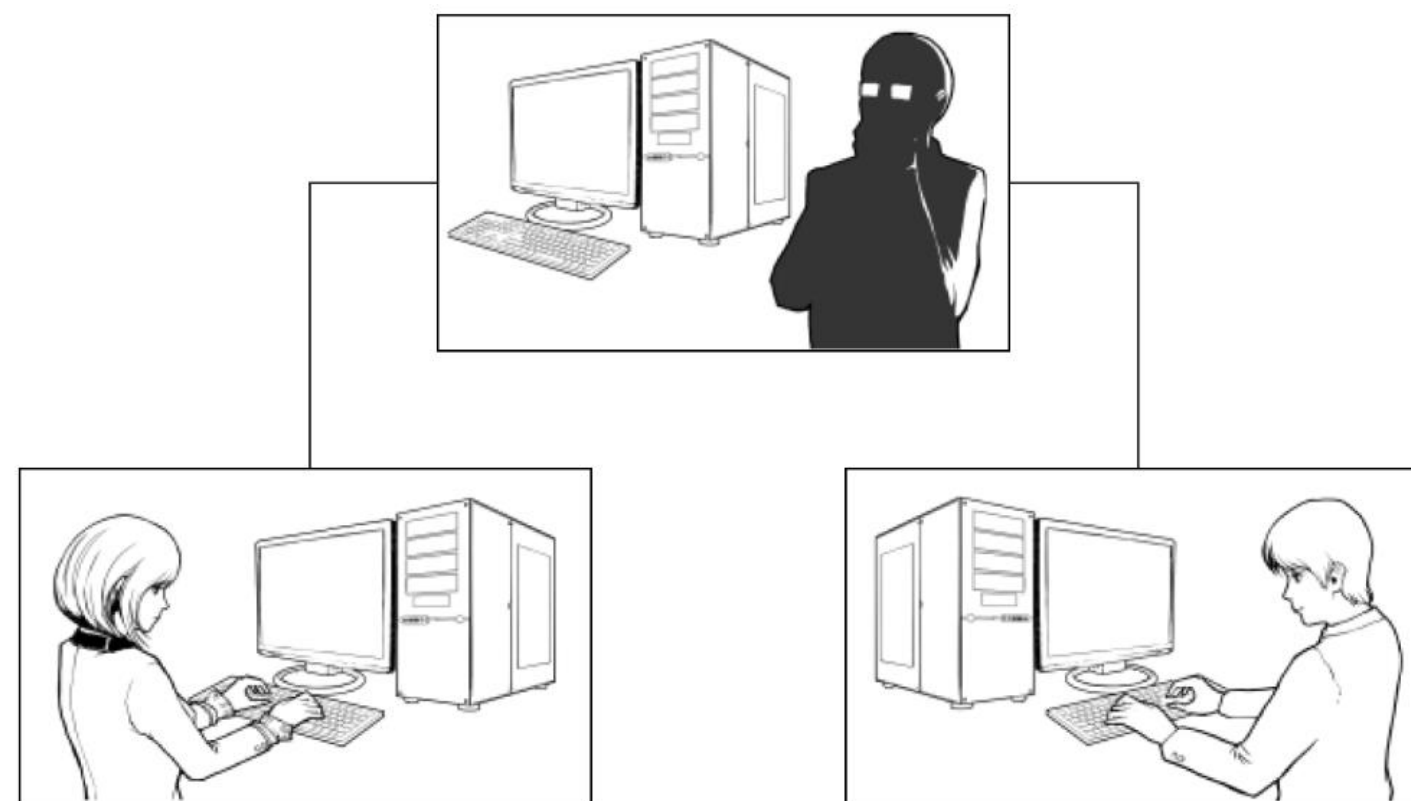
- 튜링 모방 게임

- 튜링은 컴퓨터의 지능적 행동을 인지 작업에서 인간 수준의 성능을 낼 수 있는 능력으로 정의함. 즉, 질문자가 질문에 대한 답변을 보고 답변자가 인간인지 기계인지 구별하지 못한다면 컴퓨터는 테스트를 통과한 것임.
- 튜링이 제안한 튜링 모방 게임은 두 단계로 나뉨.

Turing's Imitation Game 1/2

● 첫 번째 단계

- 질문자, 남자, 여자 한 명이 각각 독립된 방에 있고, 원격 터미널과 같은 중립 매체를 통해서만 통신할 수 있음.
- 질문자의 목적은 상대방에게 질문하면서 누가 남자고 누가 여자인지 알아내야 함.
- 남자는 자신이 여자라고 질문자를 속이고 여자는 질문자에게 자신이 여자라는 확신을 심어주는 것이 게임의 규칙임.



[그림 1-1] 튜링 모방 게임 : 단계 1

Turing's Imitation Game 2/2

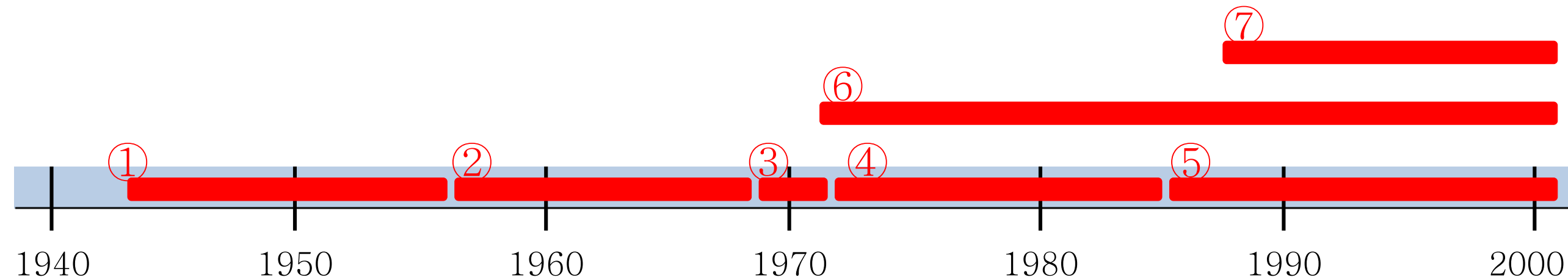
● 두 번째 단계

- 두 번째 단계에서는 남자 대신 질문자를 속이도록 프로그래밍한 컴퓨터를 둠. 사람이 하는 대로 실수도 하고 애매한 답변을 하는 것까지 프로그래밍함.
- 만약 컴퓨터가 남자가 했던 것만큼 질문자를 자주 속일 수 있다면 이 컴퓨터는 지능 행동 테스트를 통과한 것임.



[그림 1-2] 튜링 모방 게임 : 단계 2

AI의 역사



- ① 암흑기: AI의 탄생(1943년 ~ 1956년)
- ② AI의 융성 : 큰 기대의 시기(1956년 ~ 1960)
- ③ 이행되지 않은 약속 : 현실의 직면(1960년대 후반~1970년대 초반)
- ④ 전문가 시스템의 기술 : 성공의 열쇠(1970년대 초반~1980년대 중반)
- ⑤ **기계가 학습하는 법 : 신경망의 재탄생(1980년대 중반~), 딥러닝(2010~)**
- ⑥ 진화 연산 : 탐색하면서 배우기(1970년대 초반~)
- ⑦ 지식 공학의 새로운 시대 : 단어로 계산하기(1980년대 후반~)

■ 인공지능의 재 확산 : 2000년대

- Weak AI (narrow AI, applied AI) : 여러 응용 분야에서 성공적 결과. 산업체 중심
 - 자연어처리, 음성인식, 컴퓨터비전
- Deep Learning : 신경회로망의 부활
 - 빅데이터, 고속연산(GPU)

AI 탄생이전 : 1943~1956

- “Artificial Intelligence” : 1956년 Dartmouth 워크샵에서 만들어 짐
- AI 탄생이전 (1943년 ~ 1956년)
 - 워런 맥클록 (Warren McCulloch)과 월터 피츠(Walter Pitts)
 - AI 분야로 인정받은 최초 연구를 소개. (1943년)
 - AI의 첫 번째 주요 논문 발표 - 뇌의 뉴런 모델(model of neurons of brain)
 - 인공 신경망(artificial neural networks) 모델을 제안. (1943년)
 - 인공 신경망에서 각 뉴런은 이진 상태, 즉 on 또는 off 상태에 있다고 가정함.
 - 계산이 가능한 함수라면 뉴런이 연결된 모든 망에서 계산할 수 있음을 증명.
 - 클라우드 새넌(Claude Shannon)
 - 일반 체스게임이 10^{120} 번의 이동횟수를 포함한다는 점을 지적하는 체스게임 기계에 관한 논문 발표. (1950년)
 - 해를 찾을 때 휴리스틱을 사용해야 한다는 사실을 증명.
 - 존 맥카시(John McCarthy)
 - 기계 지능, 인공신경망, 오토마타 이론에 관심 있는 연구자를 모아 Dartmouth 대학교에서 여름 워크샵 열도록 도움. (1956년)
 - Dartmouth 워크샵에서 인공지능이라는 새로운 과학 분야가 탄생.

AI 탄생, 부흥 : 1956~1960

● Dartmouth Conference : 1956

- Top researcher 10+명 모여 약 8주간 토의
- 인공지능(Artificial Intelligence)라는 분야를 정의
- 인공지능에 관련된 문제들을 나열 많은 학자들을 이 분야에 몰리게 함. 지금도 연구되고 있는 주제들
 - Automatic computers
 - Natural Language Processing
 - Neural Nets
 - Theory of the size of a Calculation
 - Self-Improvement
 - Abstraction (machines forming abstractions from data)
 - Randomness and Creativity

● AI의 융성 : 큰 기대의 시기(1956년 ~ 1960)

- 초기 AI는 '엄청난 열광', '근사한 아이디어', '매우 제한된 성공'으로 특징지을 수 있음.
- 반복적인 수학 계산을 위해 컴퓨터를 도입했으나, 당시 AI 연구자는 컴퓨터가 그보다 훨씬 많은 일을 할 수 있음을 보임.

'AI의 융성'의 공헌자

- 존 맥카시 (John McCarthy)
 - **LISP를 정의 함.**
 - 「Programs with Common Sense」 논문에서 세상의 일반적인 문제에 대한 해를 찾는 전문가 의견 청구자(Advice Taker)라는 프로그램을 제안함.
 - 단순 공리에 기초한 이 프로그램으로 공항에 가는 계획을 생성하는 방법을 보여줌.
 - 다시 작성하지 않고도 다른 분야에서 새로운 공리를 사용할 수 있도록 설계.
 - 전문가 의견 청구자는 지식 표현 및 추론 중심의 원칙을 통합하는 첫 번째 완전한 지식기반 시스템.
- 민스키 (Minsky)
 - 형식 논리에 초점을 두었던 맥카시와 달리 민스키는 지식 표현과 추론에 관한 반논리적인 사고 방식을 발전시킴.
 - 프레임(frame) 이론은 지식 공학에 많은 공헌을 함. (1975)
- 프랭크 로젠블랫(Frank Rosenblatt)
 - 학습 알고리즘이 퍼셉트론(perceptron)의 연결 강도를 조정할 수 있다는 것을 보임.
 - 퍼셉트론 수렴 이론(perceptron convergence theorem)을 증명.
- 앨런 뉴웰(Allen Newell)과 허버트 사이먼(Herbert Simon)
 - 인간의 문제해결 방식을 모방하는 범용 프로그램 GPS(General Problem Solver)을 개발.
 - GPS는 데이터와 문제해결 기법을 구분한 첫 번째 시도임.
 - 방법-결과 분석(means-ends analysis)이라는 기법에 기반을 두고 있음.

'AI의 융성기'의 의미

● GPS프로젝트

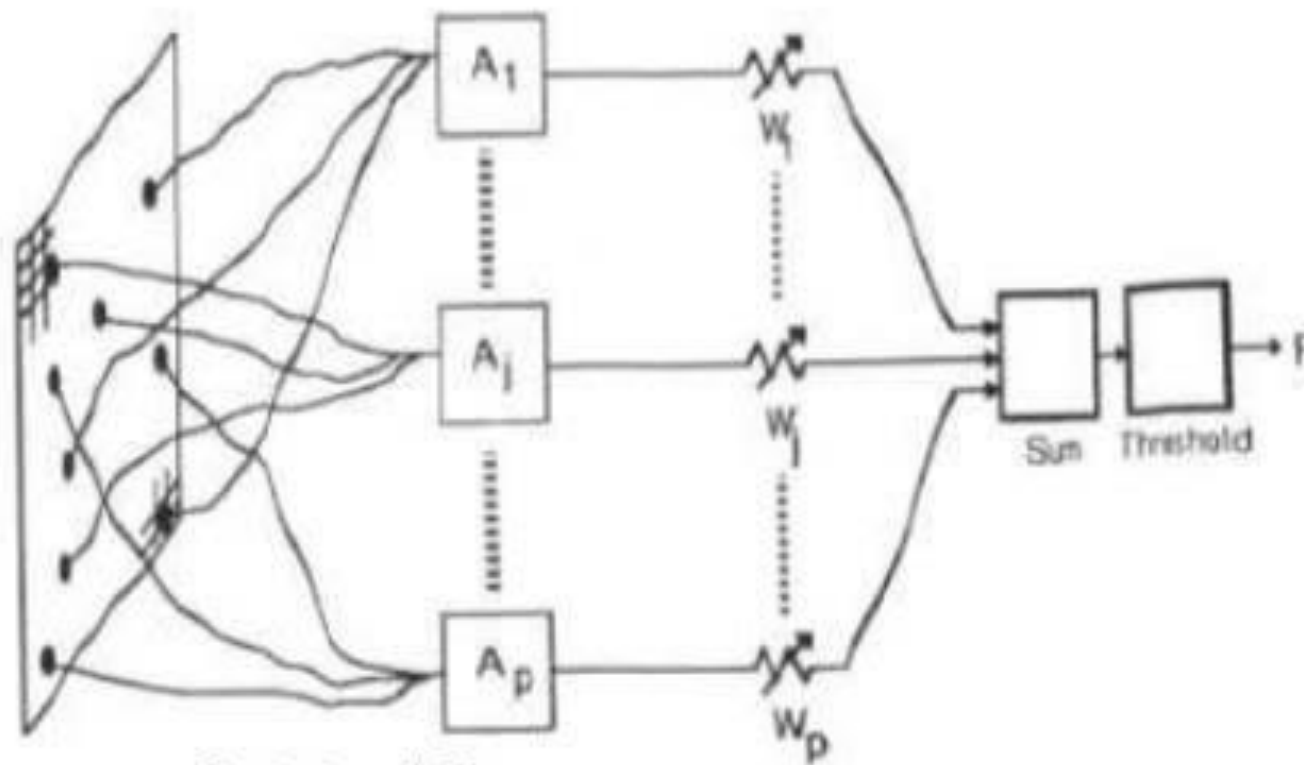
- 뉴웰과 사이먼이 개발(1961, 1972)
- GPS는 데이터와 문제해결 기법을 구분한 첫 번째 시도임.
- 방법-결과 분석(means-ends analysis)이라는 기법에 기반을 둠.
- GPS는 복잡한 문제를 푸는 데 실패함.
 - 형식 논리에 기반을 둔 프로그램이 사용 가능한 연산자를 무한정 생산함.
 - 현실 세계 문제를 푸는 데 필요한 시간과 메모리의 양 때문에 진행이 중단됨.

● 'AI의 융성'의 의의

- AI 연구자들은 광범위한 문제를 풀기 위한 일반 방법(general methods)을 만들어서 복잡한 사고 과정을 묘사하려고 하였음
- 문제 영역에 적합하지 않은 정보를 사용했고 이는 개발한 프로그램의 성능을 저하시켰음.
- AI 분야에 매력을 느낀 위대한 과학자들이 지식 표현, 학습 알고리즘, 신경 컴퓨팅, 단어계산과 같은 분야에 새롭고 중요한 아이디어를 제안함.

Perceptron (1957)

Perceptron (1957)

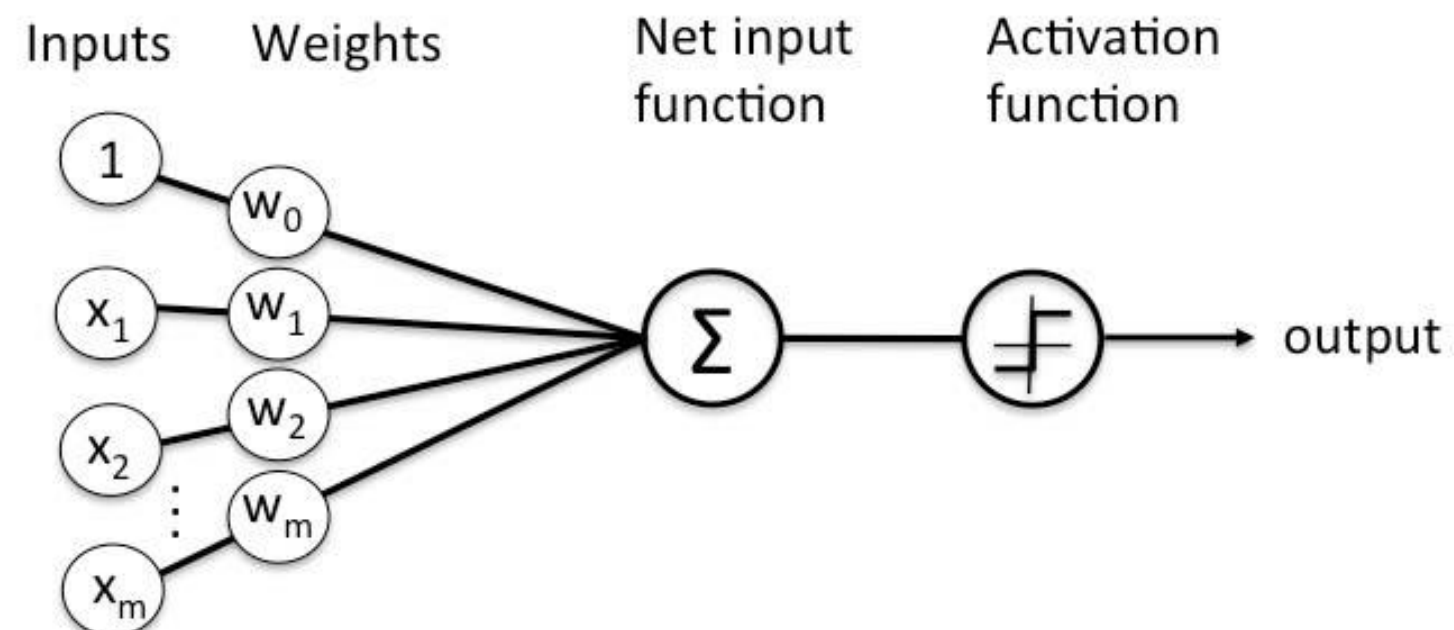
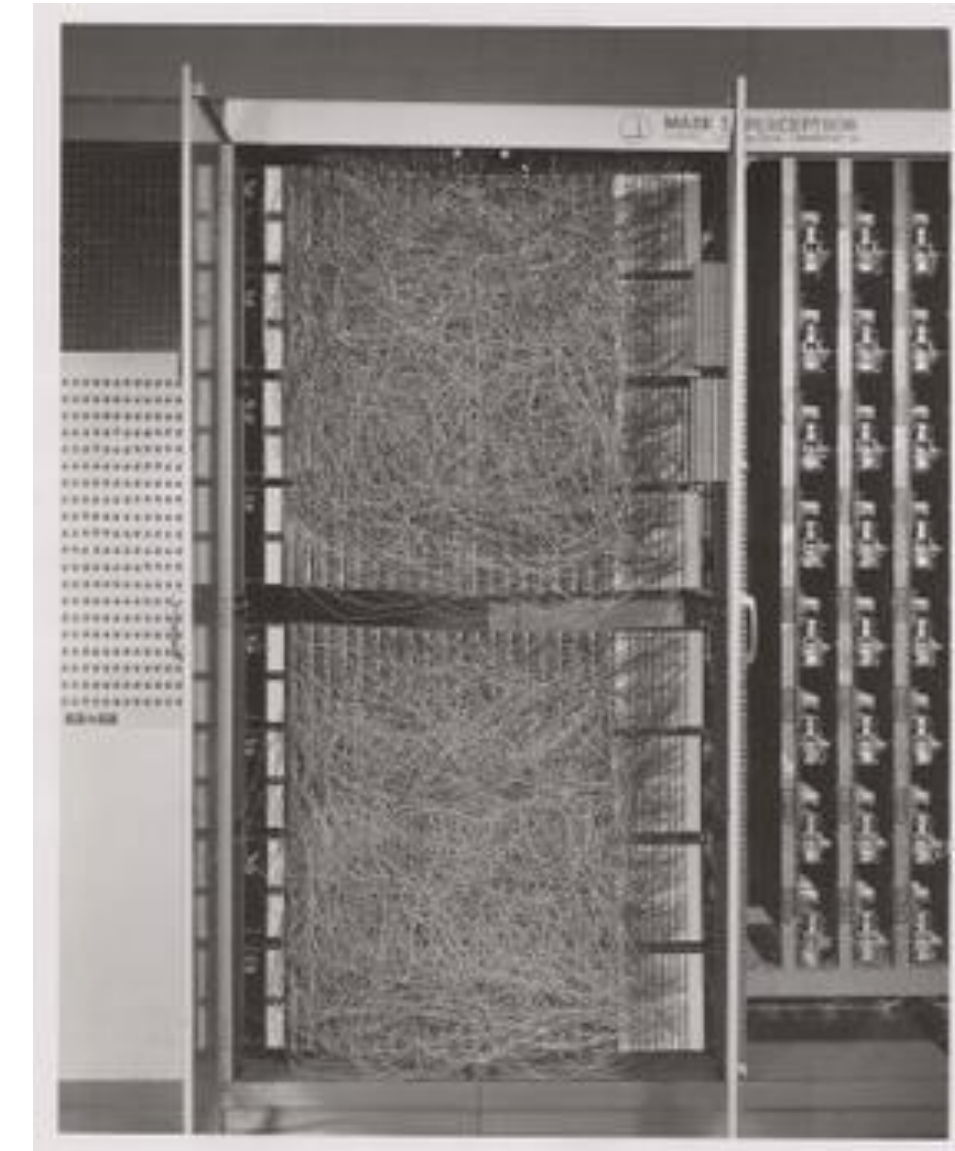
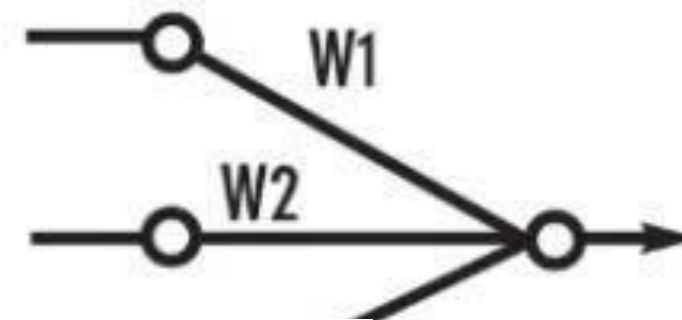


Original Perceptron

(From *Perceptrons* by M. L. Minsky and S. Papert, 1969, Cambridge, MA: MIT Press. Copyright 1969 by MIT Press.)



Frank Rosenblatt
(1928-1971)



- 입력 : 20x20 camera(photocells)
- 출력 : 영상인식
- Neuron-like computational unit
→ Neural Network → Deep Learning
- **학습** 알고리즘 제안
- 한계가 드러나(1969. Perceptron책) 사망
→ 1980년 중반에 부활

암흑기 : 1960년대 후반~1970년대 초반

● 이행되지 않은 약속 : 현실의 직면

■ AI 연구자들의 낙관과 실패

- AI 연구자들은 1950년대 중반부터 1980년대까지는 인간 규모의 기반지식을 가진 범용 지능형 기계를 만들고, 2000년에는 인간의 지능을 넘어서게 하겠다고 약속함.
- 그러나 1970년에 이르러 그런 주장은 너무 낙관적이었다는 사실을 깨달음.

■ AI의 프로그램의 한계

- 몇몇 AI 프로그램이 현실 문제를 간단하게 만든 장난감 문제 한두 개에서 일정 수준의 기계 지능을 보여줌.
- 하지만 대부분의 AI 프로젝트는 작업의 범위를 넓히거나 더 어려운 실세계 문제를 다루지 못함.

● 1960년대 후반에 AI에 관한 주된 어려움

- AI 연구자는 광범위한 문제를 해결할 일반적인 방법을 개발하고 있었기 때문에 초기 프로그램에는 특정 문제 영역에 관한 지식을 거의 포함하지 못함.
- 문제를 해결하기 위해 프로그램은 해를 발견할 때까지 작은 단계들을 여러 조합으로 만들어보는 탐색 전략을 적용.
- '장난감 문제'와 같은 작은 문제에서 잘 동작. 그러나 어렵고 복잡한 큰 문제에서 잘 동작하지 않음.

암흑기 : 계속

- 1960년대 후반에 AI에 관한 주된 어려움
 - AI로 풀려고 한 많은 문제는 범위가 넓고 풀기도 어려움
 - 초기 AI 작업은 대개 기계 번역.
 - 올바른 단어를 선택하려면 주제를 먼저 이해해야 한다는 사실을 알게 됨. 그러나 이는 너무 어려운 과정이었음.
 - AI에 대한 지원 중단
 - 1966년에 미국 정부의 지원을 받았던 AI를 이용한 모든 번역 프로젝트 취소.
 - 1971년에 영국 정부도 AI 연구에 대한 지원 중단.
- "some respectable achievements (but to a disappointingly smaller extent than had been hoped and expected)" by James Lighthill, "Artificial Intelligence: A General Survey," 1973

전문가 시스템 출현 : 1970년대 초반~1980년대 중반

● 전문가 시스템의 기술 : 성공의 열쇠

- 지능형 기계에 대한 문제 영역을 충분히 제한해야 한다는 사실을 깨달음.
 - 이전의 AI 연구자는 일반적이며 인간의 문제 풀이 방법을 흉내 낸 똑똑한 탐색 알고리즘과 추론 기법을 만들 수 있다고 믿음.
 - 이는 실패했고 이를 통해 연구자들은 전문지식이 필요한 전형적인 영역으로 문제를 제한하고 큰 추론단계로 해결해야 실용적인 결과를 얻을 수 있다는 것을 깨달음.

● '전문가 시스템의 기술' 시대의 주요 프로젝트

- DENDRAL 프로젝트
 - 질량 스펙트럼을 결정하는 시스템.
 - 파이겐바움, 뷰캐넌, 레더버그가 만든 AI 시스템.
 - 인간 전문가의 기술과 경험, 의견을 이용하여 문제를 해결.
 - AI에서 핵심적인 '패러다임 전환'을 만듦. 범용으로 쓰이는 (지식이 빈약한) 약한 방법에서 특정 분야의 (지식 집중적인) 방법으로 전환됨.
 - 인간 전문가에게서 이끌어낸 양질의 특수 규칙 형태로 휴리스틱을 사용하여 컴퓨터가 제한된 문제 영역에서 전문가와 같을 수 있음을 보였음.
 - 전문가 시스템(expert systems)이라고 함 : 지식 공학(knowledge engineering) 개념 만듦.
 - 지식공학 : 전문가의 '노하우'를 규칙으로 얻고 분석하여 표현하는 기법.

전문가 시스템 : 계속

■ MYCIN 프로젝트

- 전염성 혈액 질환을 진단하는 규칙기반 전문가 시스템.
- 관련 분야에서 인간 전문가와 동일한 수준으로 수행할 수 있었고, 경험이 부족한 의사보다 오히려 수준이 높았음.
- 규칙의 형태로 통합된 지식은 추론 메커니즘과 분리함. 시스템 개발자는 일부 규칙을 추가하거나 삭제함으로써 시스템 내의 지식을 쉽게 다룰 수 있음.
- MYCIN에 통합된 규칙은 지식과 관련된 불확실성을 반영함.

■ PROSPECTOR 프로젝트

- 광물탐사 전문가 시스템.
- 프로젝트는 1974년~1983년 동안 수행되었으며, 전문가 아홉 명이 자신의 지식과 의견을 제공.
- 지식을 표현하기 위해 규칙과 의미망(semantic network)을 결합한 구조를 사용.

전문가 시스템 : 계속

● 전문가시스템의 특징

- 1970년대 후반에 전문가 시스템을 성공적으로 응용한 사례가 증가하면서 연구소에 머물러있던 AI 기술이 상업적인 환경으로 성공적으로 전이함.
- 이 기간 동안 대부분의 전문가 시스템은 강력한 워크스테이션에 기반을 둔 LISP, PROLOG, OPS와 같은 특별한 AI 언어로 개발됨.
- 비싼 하드웨어와 복잡한 프로그래밍 언어가 있어야 한다는 조건 때문에 전문가 시스템 개발은 스탠포드 대학교, MIT, 스탠포드 연구소, 카네기멜론 대학교의 몇몇 연구 그룹에서 이루어짐.
- 1980년대 개인용 컴퓨터(PC)와 사용하기 쉬운 전문가 시스템 개발 툴(shell)이 등장하고 나서야 모든 분야의 일반 연구자와 공학자가 전문가 시스템을 개발할 기회를 얻음.
- 전문가 시스템의 제약 사항
 - 전문가 시스템의 사용은 매우 한정된 전문적 기술분야로 제한됨.
 - 1세대 전문가 시스템은 자신의 경험을 통해 배울 수 있는 능력이 없음.

● 지식병목(Knowledge Bottleneck) 현상 ➔ 지식을 **학습**할 수 있으면...

신경망의 재탄생 : 1980년대 중반

● 신경망의 부활

- 기존 지식기반(symbolic AI) 한계 부각됨 : **학습**의 문제
- 사람의 뇌(신경세포의 네트워크) 구조를 흉내 내어야 학습 문제가 해결할 수 있다 : 연결주의(connectionism) 부각
- Perceptron 문제점 보완한 학습 알고리즘 개발됨 : Multi-Layer Perceptron

● 신경망 부활'의 공헌자

- 그로스버그가 자기조직의 '적응형 공진' 이론을 세워, 신경망에 대한 기초를 제공.
- 홉필드는 피드백이 있는 신경망인 '홉필드 신경망'을 제안.
- 코호넨이 '자기조직 맵'에 관한 논문을 출판.
- 루메하트와 맥클러랜드가 '역전파 학습 알고리즘'을 재발견함.

The Second AI Winter : 1988~1993

- 신경망은 Layer가 많아지면 학습이 안됨 : Gradient Vanishing Problem
- 전문가 시스템은 학습의 한계 + 계산량이 많아 전용 하드웨어 필요
- 300개이상 AI 회사들 파산

AI 다시 살아남 : 1993~2011

- Deep Blue chess computer : 세계 챔피언 이김 1997. IBM
- Roomba(Home Robot) : 상업적 성공 1992. iRobot
- Autonomous Robotic Car : 모하비사막 132miles 주행. 2005. Stanford Univ.
- Google app with Speech recognition 성공적 : 2008.
- Watson : Jeopardy! Game show 챔피언 이김. 2011. IBM

- LeNet 1998 : Deep Layer에서 신경회로망 학습 성공
→ 지금의 Deep Learning

딥러닝 혁명 : 2012~

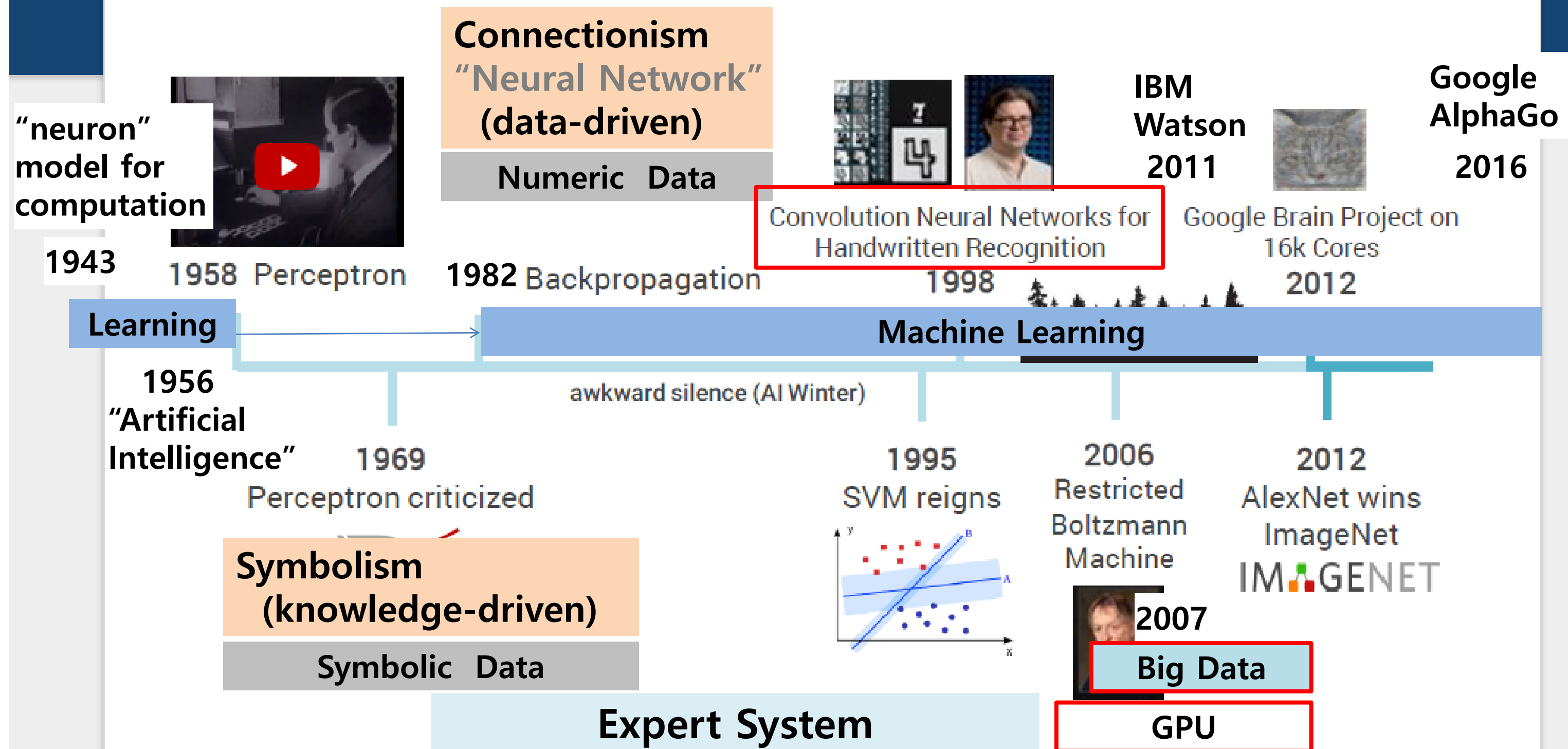
- AlexNet : CNN(딥러닝의 일종). 영상인식분야 대회(ILSVRC 2012)에서 우승
 - 25.8%(2011) ➔ 16.8%(2012) : '혁명적'으로 오인식율 감소
 - 사람은 약 5% 오류 : 2015년부터는 사람 인식율 능가
 - ImageNet database 때문에 가능해짐
 - GTX 580 3GB GPU 2개에서 5~6일 학습
- ImageNet database(2009) : 1,500만 Images, 2,200 Categories. cloud sourcing(Amazon mechanical Tech 등) 이용, 167개국에서 5만명 참가. 2007년부터 구축 시작
 - ILSVRC(Large Scale Visual Recognition Competition) 대회 (2010) : 1,000 Categories, 120만 images 학습 data. random하게 고른 20만장 test

인간을 넘어서?

- Chess : DeepBlue(IBM) 1997
- Quiz Show(Jeopardy!) : Watson(IBM) 2011
 - “이 도시의 가장 큰 공항은 제2차 세계대전 영웅의 이름을 따서 명명되었고, 두 번째로 큰 공항은 제2차 세계대전 중의 전투 이름을 따릅니다” : 시카고
- 영상인식 : ResNet(Kaiming He. Microsoft → Facebook) 2015 (3.75%. 사람은 평균 5%)
- 바둑 : AlphaGO(Google DeepMind) 2016
- Poker : DeepStack(Intl' team. Czech+Canada) 2017
- Stanford Univ reading and comprehension test (SQuAD. Stanford Question Answering Dataset) : Alibaba, Microsoft 2018
- 인간을 넘어서?

A brief History

A long time ago...



한눈에 보는 머신러닝

- 머신러닝이란 무엇일까요?
- 기계가 배운다는 것은 정확히 무엇을 의미하는 걸까요?
- 머신러닝이 왜 필요한가요?

머신러닝의 시스템 종류, 주요 도전 과제, 평가와 튜닝

머신러닝이란

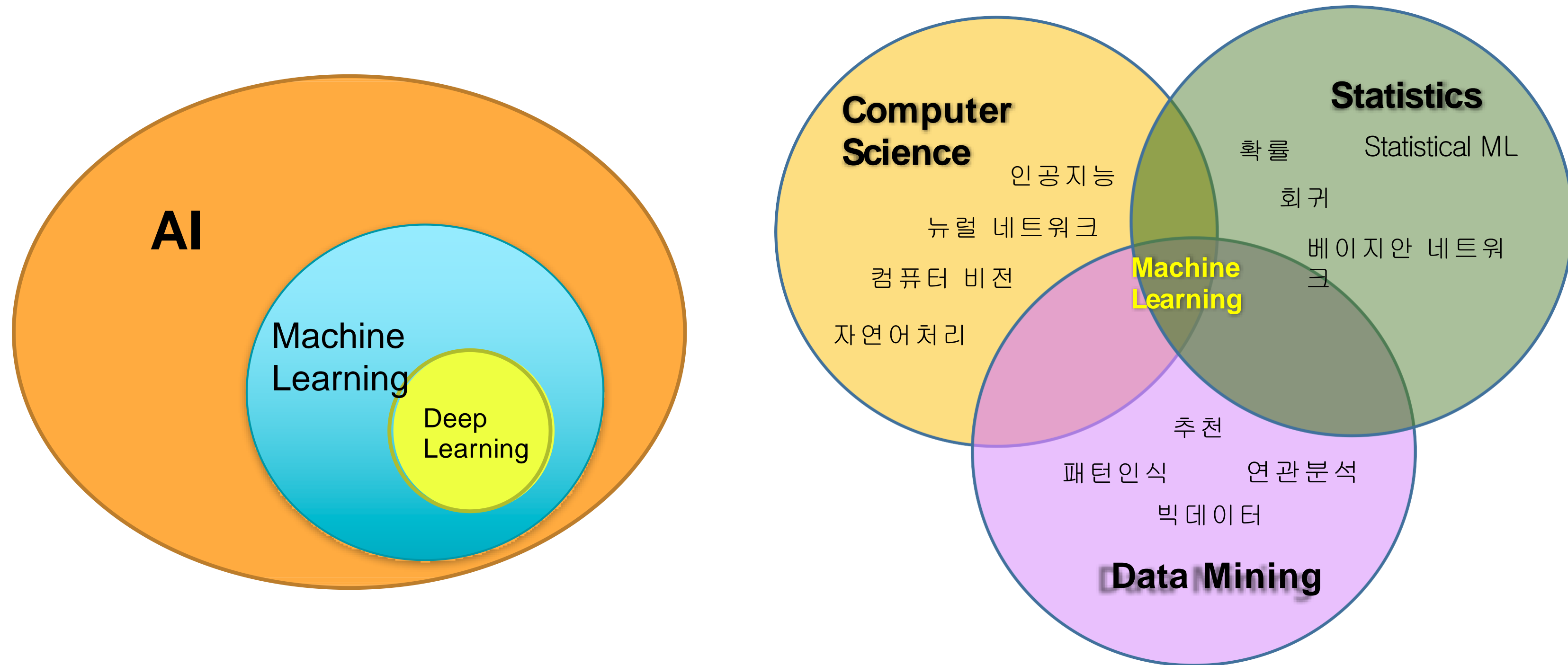
- 현업에서 본 머신러닝 : <https://www.youtube.com/watch?v=aF03asAmQbY>
- “머신러닝”은 명시적인 프로그래밍 없이 컴퓨터가 학습하는 능력을 갖추게 하는 연구 분야다. (아서 사무엘 Arthur Samuel, 1959)
- 어떤 작업 **T**에 대한 컴퓨터 프로그램의 성능을 **P**로 측정했을 때 경험 **E**로 인해 성능이 향상됐다면, 이 컴퓨터 프로그램은 작업 **T**와 성능 측정 **P**에 대해 경험 **E**로 학습한 것이다. (토미 미첼 Tom Mitchell, 1997)
 - 스팸 메일 구분하기—작업 **T**
 - 훈련 데이터 training data—경험 **E**
 - 정확도 accuracy—성능 측정 **P**
- 머신러닝 : 데이터로부터 학습하도록 컴퓨터를 프로그래밍하는 기술

The Machine Learning Tsunami (교재 서문에서 가져옴)

In 2006, Geoffrey Hinton et al. published a paper¹ showing how to train a deep neural network capable of recognizing handwritten digits with state-of-the-art precision (>98%). They branded this technique “Deep Learning.” Training a deep neural net was widely considered impossible at the time, and most researchers had abandoned the idea since the 1990s. This paper revived the interest of the scientific community and before long many new papers demonstrated that Deep Learning was not only possible, but capable of mind-blowing achievements that no other Machine Learning (ML) technique could hope to match (with the help of tremendous computing power and great amounts of data). This enthusiasm soon extended to many other areas of Machine Learning.

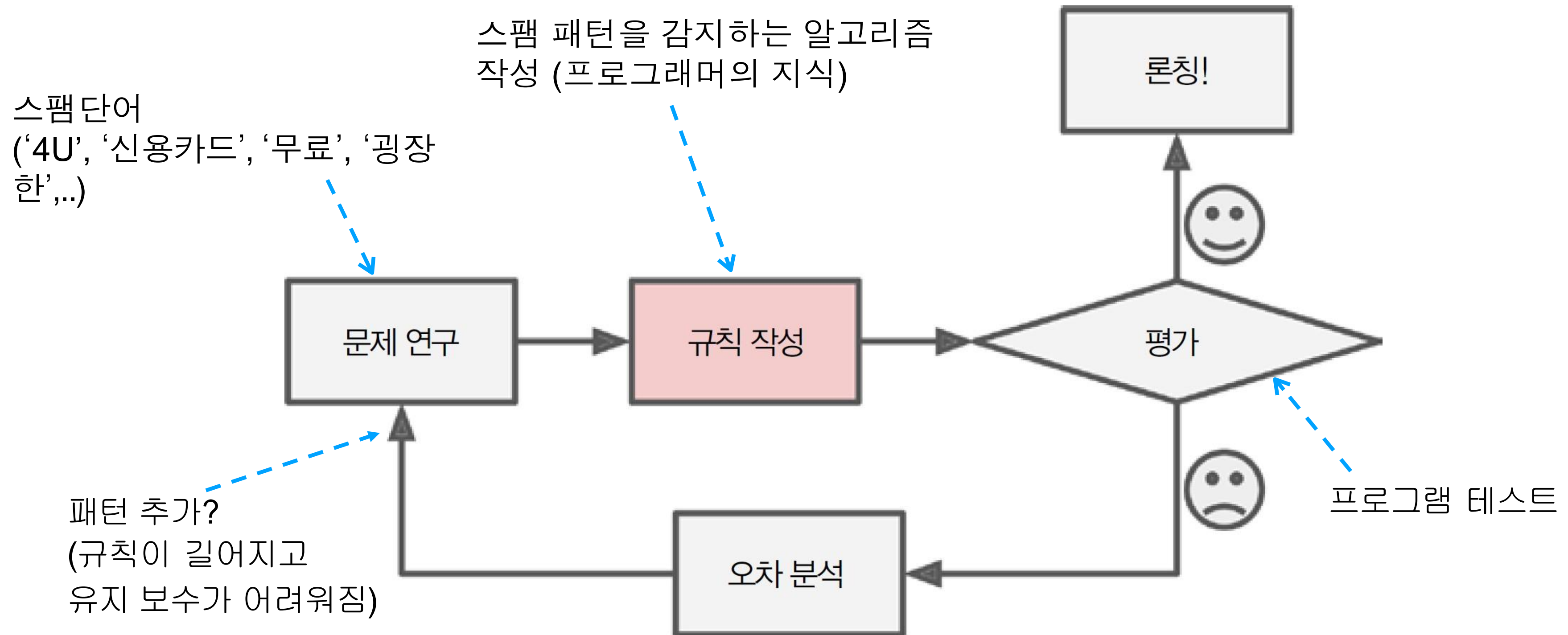
Fast-forward 10 years and Machine Learning has conquered the industry: it is now at the heart of much of the magic in today’s high-tech products, ranking your web search results, powering your smartphone’s speech recognition, and recommending videos, beating the world champion at the game of Go. Before you know it, it will be driving your car.

인공지능 > 머신러닝 > 딥러닝



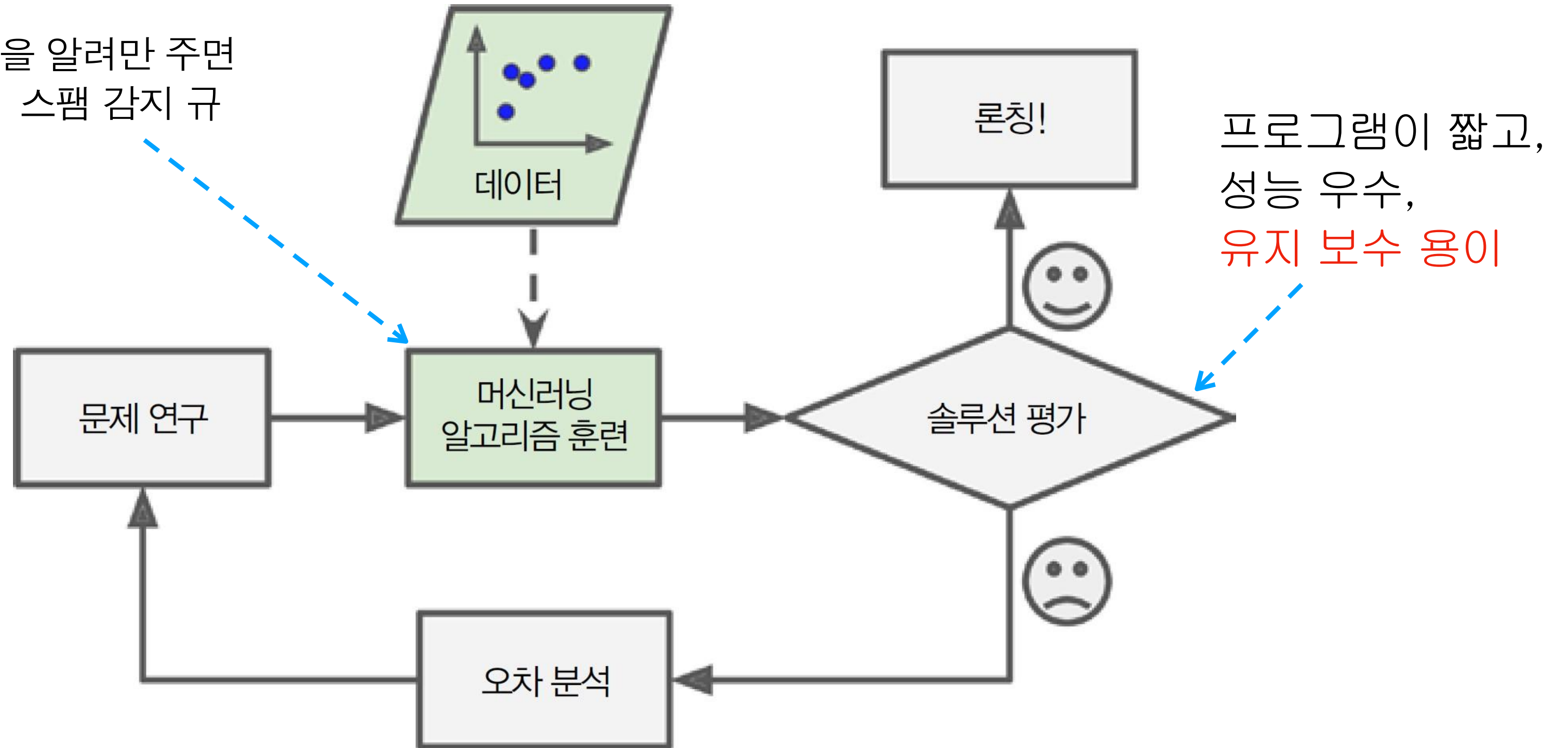
왜 머신러닝을 사용하는가

전통적인 접근 방법: 규칙 기반 시스템(rule based system)으로 스팸 필터 프로그램 예



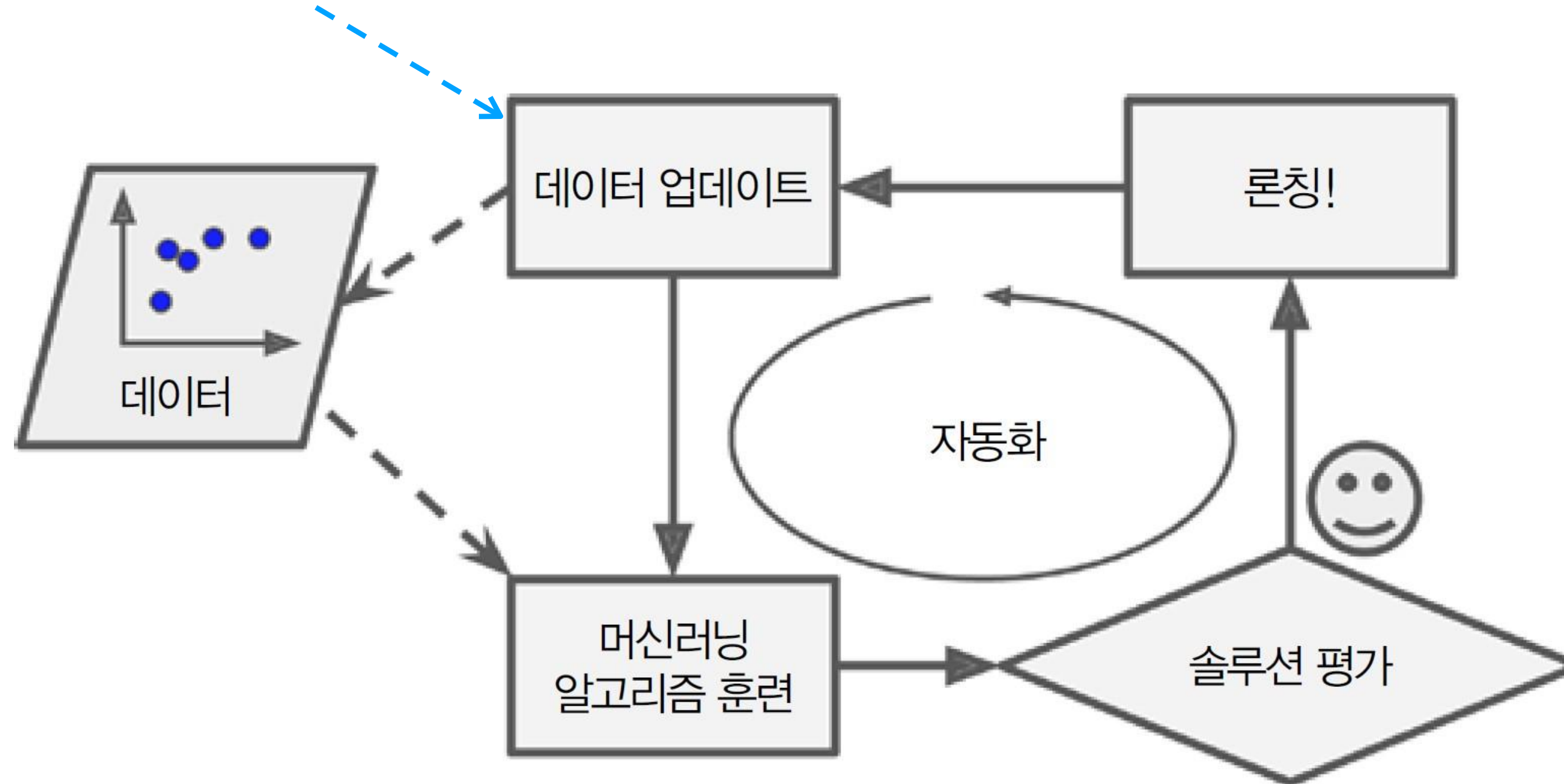
머신러닝 접근 방법

사용자가 스팸 메일이라는 것을 알려만 주면
자주 나타나는 패턴 감지하여 스팸 감지 규
칙을 자동으로 학습



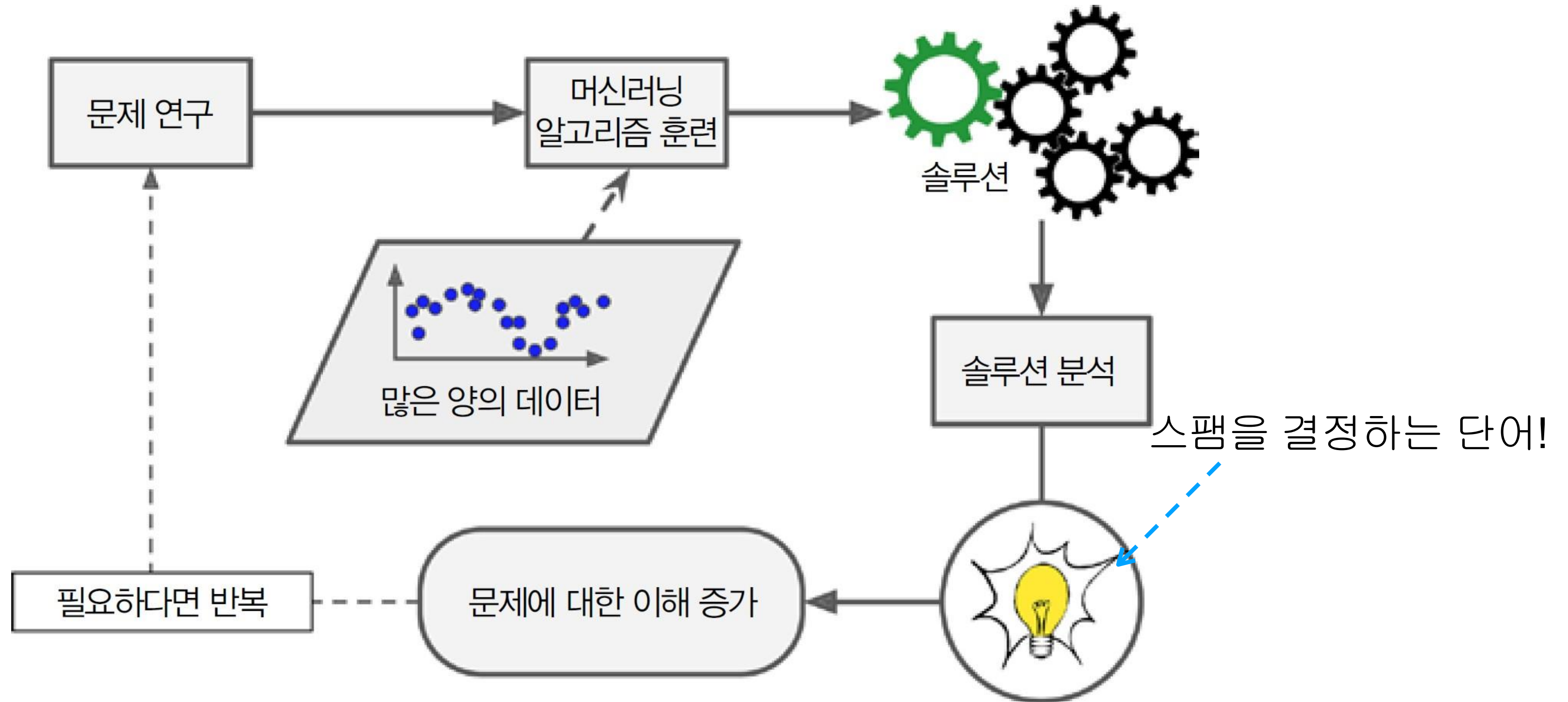
자동으로 변화에 적응

'4U' → 'For U'



머신러닝을 통해 새로운 지식을 얻을 수 있음

대용량 데이터를 분석해서 겉으로 보이지 않던 패턴을 발견 :
데이터마이닝(data mining)



머신러닝이 유용한 분야

- 많은 수동 조정과 규칙이 필요한 문제
- 전통적인 (지식기반) 방법으로 해결하기 너무 어렵거나 알려진 해가 없는 문제
 - 음성인식
- 변화하는 환경에 적응해야 하는 문제 : 새로운 데이터에 적용할 수 있음
- 복잡한 문제와 대량의 데이터에서 통찰 얻기(데이터 마이닝)

머신러닝 시스템의 종류

- 사람의 감독 여부(지도, 비지도, 준지도, 강화학습)
 - Supervised, Unsupervised, Reinforcement
- 실시간으로 점진적 학습인지 여부(온라인 학습, 배치 학습, 점진적 학습)
 - Online, Batch, Incremental
- 훈련 데이터와 단순 비교인지, 패턴을 찾기 위해 예측 모델을 만드는지(사례 기반, 모델 기반 학습)
 - Instance-based, Model-based
- 이 범주들은 배타적이지 않음(e.g. 스팸 필터-온라인 모델 기반 지도 학습)

지도학습(Supervised Learning)

- 학습데이터베이스(훈련세트) = (특징, 레이블) 쌍

- 레이블 [Label. 정답(ground truth) 또는 타겟(target)이라고도 함] :

- 스팸 / 스팸아님. → 이진 분류(binary classification)
- 개 / 고양이 / 새 → 다중 분류(multiclassclassification)

- 클래스 [Class] : 레이블의 범주

- 스팸 / 스팸아님.
- 개 / 고양이 / 새

- 특징 [feature. 속성(attribute)]

교과서에서는 '특성'으로 번역 → '특징'이 일반적인 용어

- 분류(classification) / 회기(regression) 문제

- 문서내 문자들 → 스팸 / 스팸아님 : 분류(Classification)
- 영상의 픽셀들 → 개 / 고양이 / 새 : 분류(Classification)
- (주행거리, 년식, 브랜드) → 중고차 가격 : 회기(Regression)

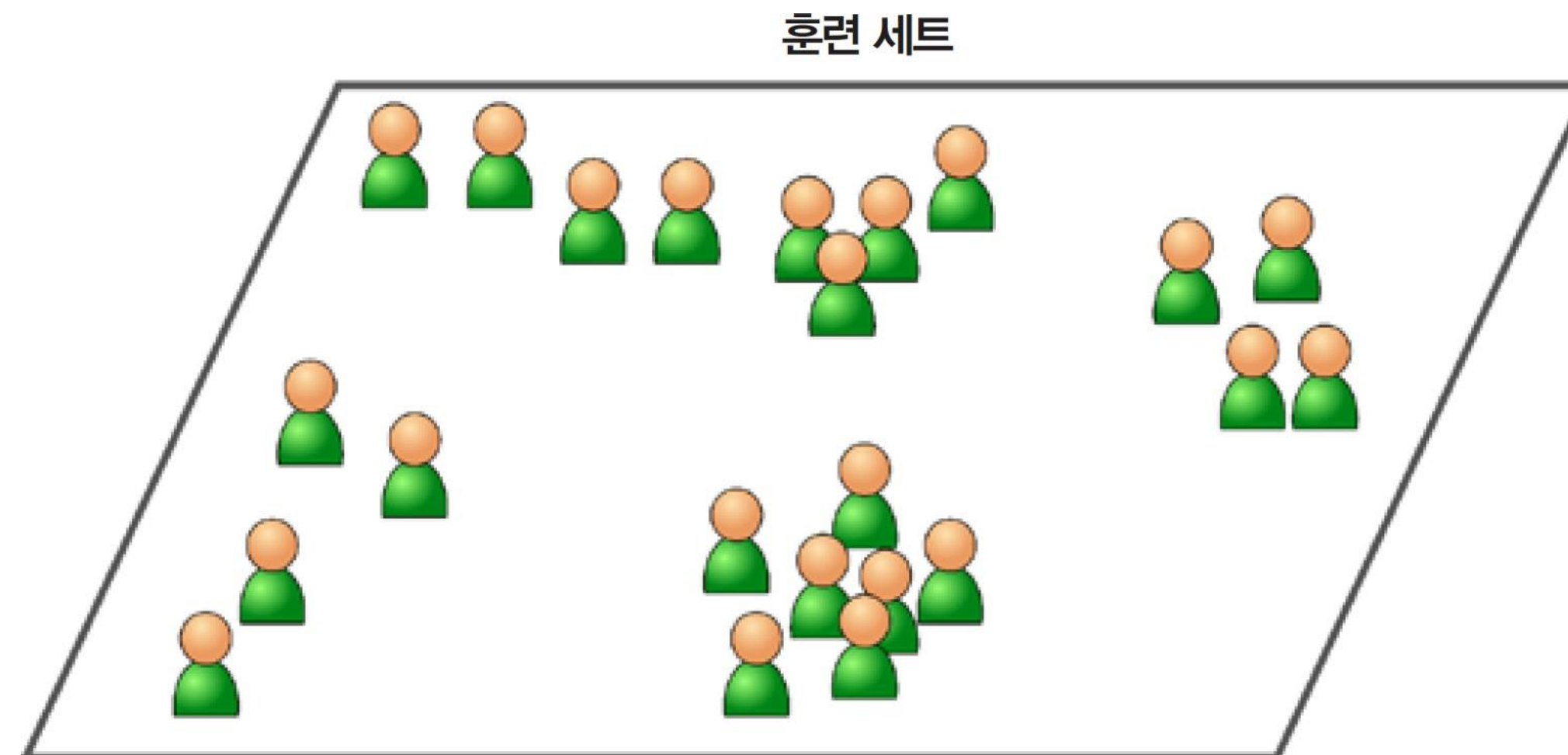


대표적인 지도 학습 알고리즘

- k-최근접 이웃(k-Nearest Neighbors)
- 선형 회귀(Linear Regression) - 회귀 알고리즘
- 로지스틱 회귀(Logistic Regression) - 분류 알고리즘
- 서포트 벡터 머신(Support Vector Machine, SVM)
- 결정 트리(Decision Tree), 앙상블 학습(Ensemble Learning)
- 신경망(Neural Network), 딥러닝(Deep Learning) : 지도/비지도/강화

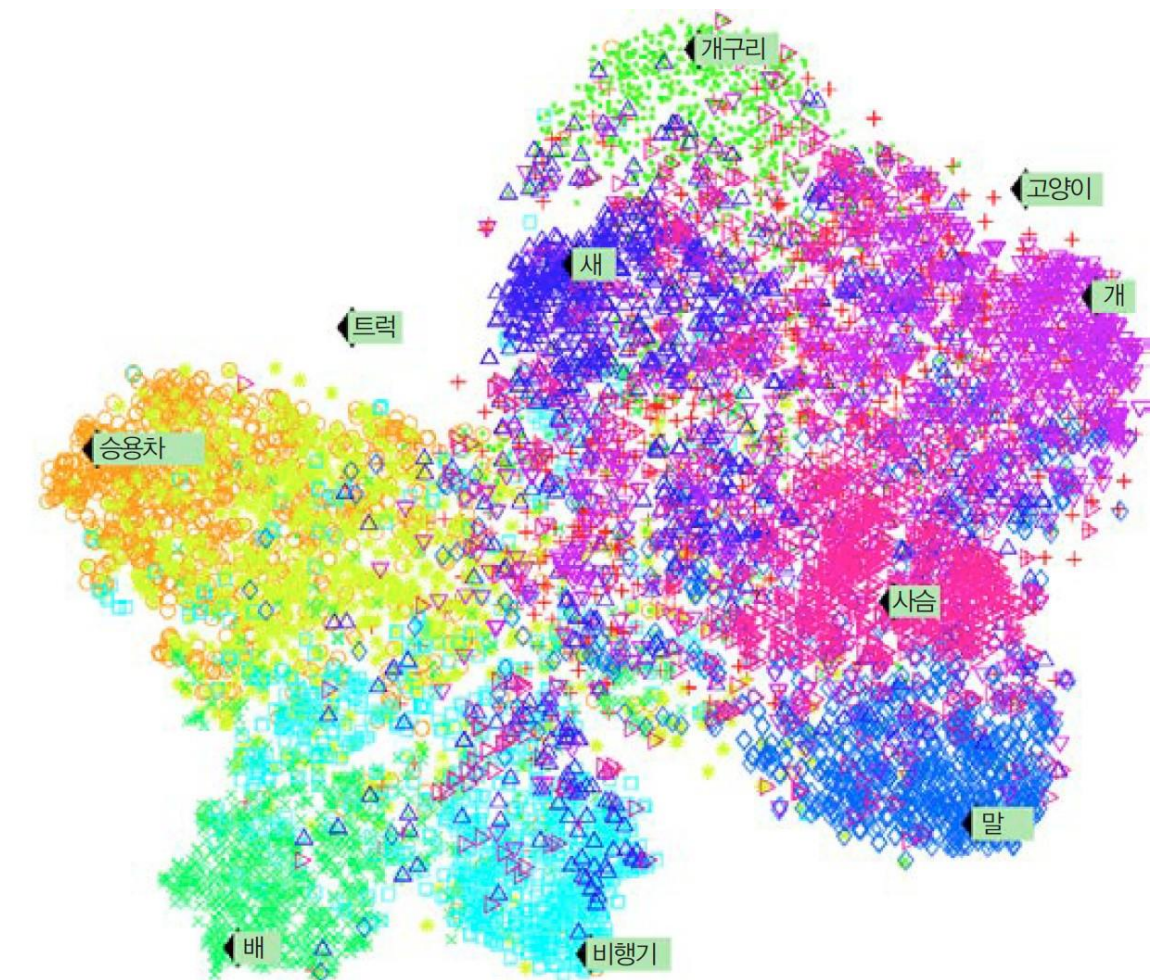
비지도 학습(Unsupervised Learning)

- 훈련 데이터에 레이블이 없음
- 블로그 방문자를 그룹으로 묶기 (어떻게 40%가 만화 애호가, 20%는 공상가인지 알 수 있을까?)
- 고객을 자동으로 그룹으로 나누어 각각 다른 홍보자료를 보내기 (캠핑용품, 식료품, 가전제품, ..)



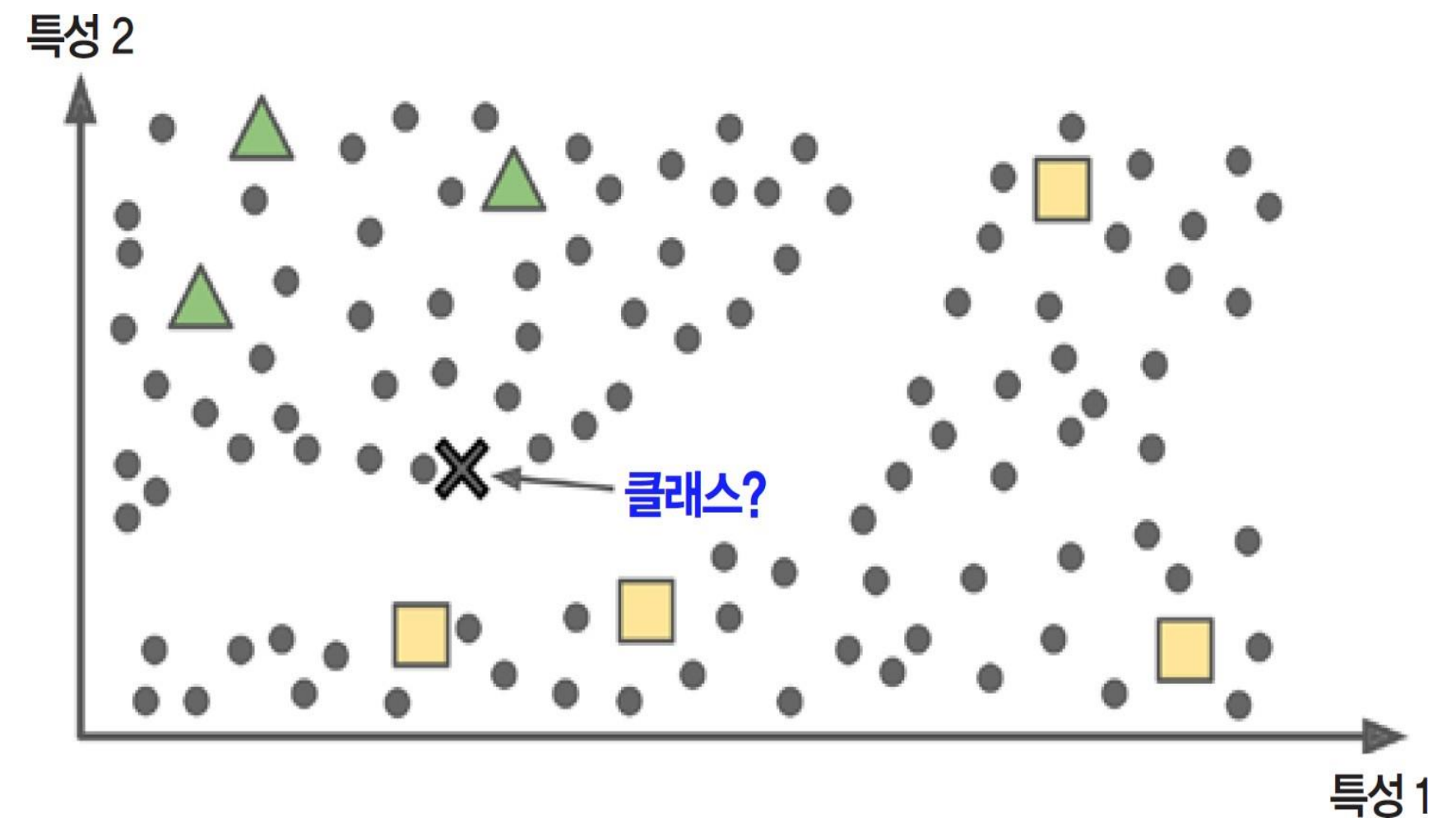
대표적인 비지도 학습 알고리즘

- 군집(Clustering)
 - k-평균(k-means)
 - 계층 군집(Hierarchical Clustering): 병합 군집(Agglomerative Clustering)
 - 기댓값 최대화(Expectation Maximization)
- 시각화, 차원 축소-8장
 - 주성분 분석(PCA), 커널 PCA, NMF
 - 지역 선형 임베딩(LLE)
 - t-SNE ----->
- 연관 규칙(Association Rule): 데이터 마이닝
 - 어프라이어리(Apriori), 이클렛(Eclat)



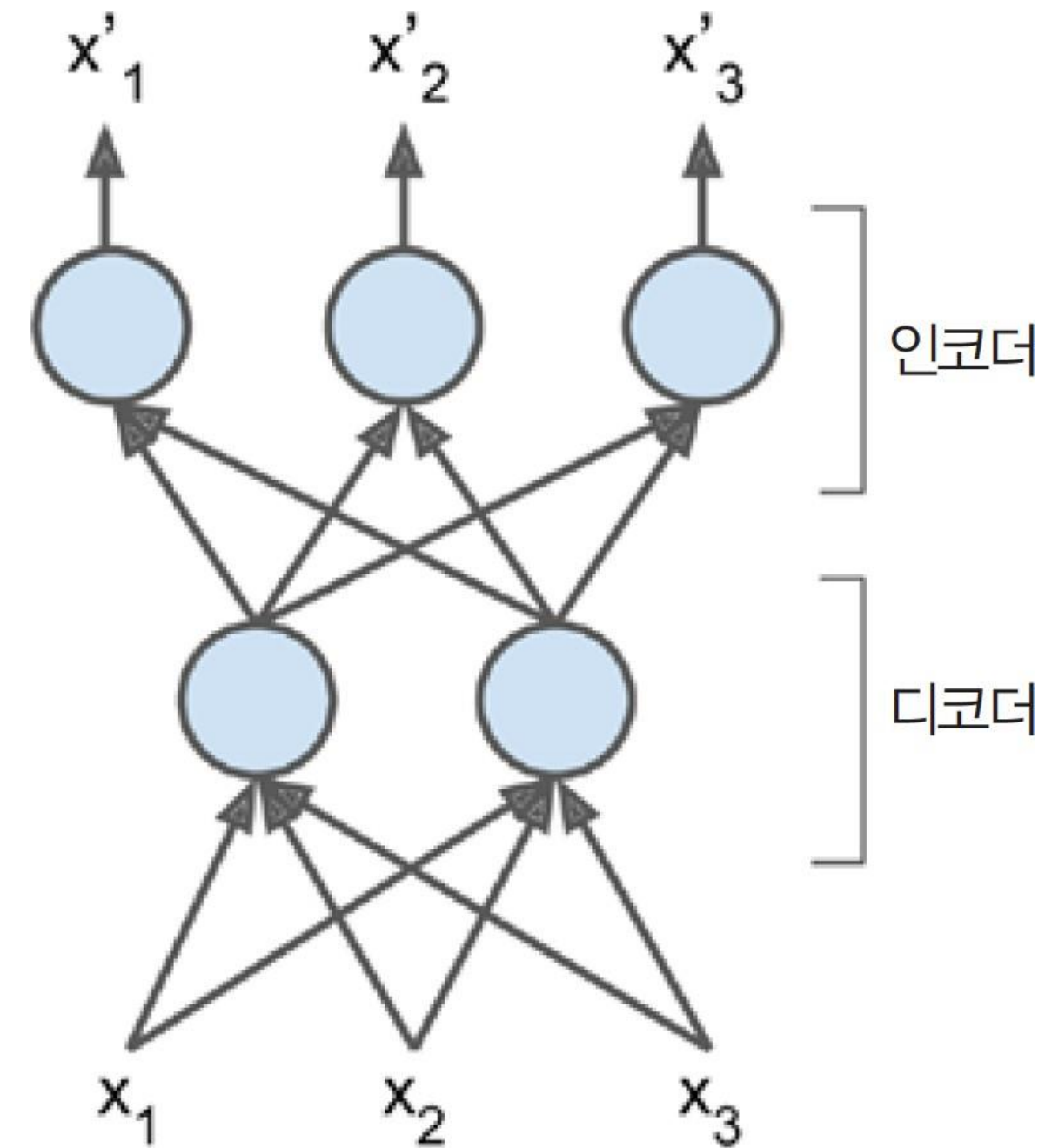
준지도 학습(Semi-supervised Learning)

- 대부분 데이터에는 레이블이 없고 일부 데이터에만 레이블이 있음
- 군집을 통해 데이터를 분할하고 소수 레이블을 이용해 전체 그룹을 인식.
- Google Photos, Facebook 사진 태그
- 제한된 볼츠만 머신(RBM)으로 구성된 심층 신뢰 신경망(DBN)



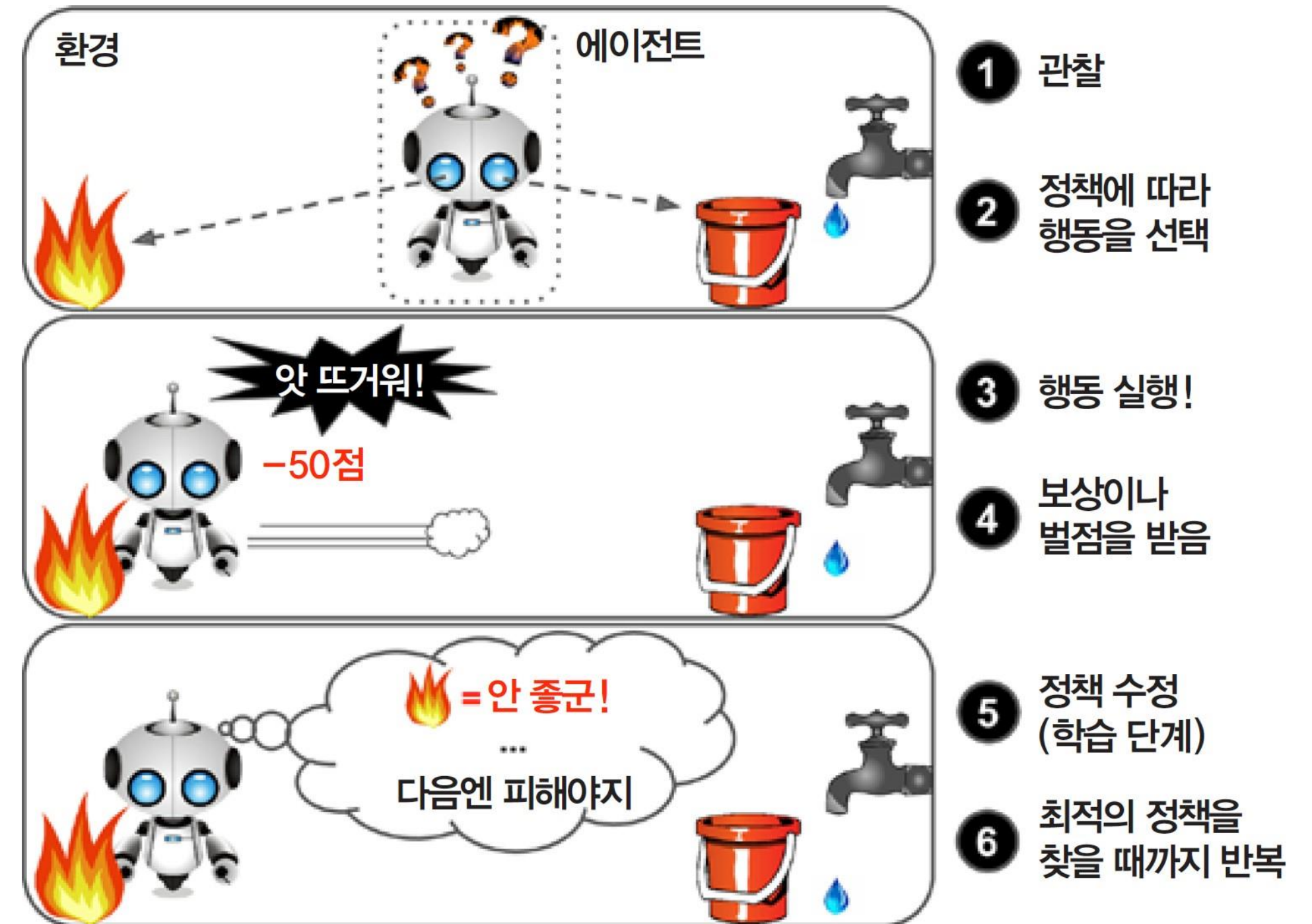
자기지도 학습(Self-supervised Learning)

- 오토인코더(Autoencoder)-15장
- 입력과 레이블이 동일함.
- 일반적인 레이블이 없기 때문에 비지도 학습, 자기자신이 타깃이기 때문에 자기지도 학습, 어쨌든 타깃이 있으므로 지도 학습으로도 부름.



강화 학습(Reinforcement Learning)

- 행동 결과에 따른 보상(혹은 벌점)을 레이블로 줌 : 바둑 경우 이겼다 혹은 졌다
- 주어진 환경(environment)에서 에이전트(agent)가 최대의 보상(reward)을 얻기 위해 최상의 정책(policy)을 학습.
- 딥마인드의 알파고(Alpha Go), 아타리(Atari) 게임

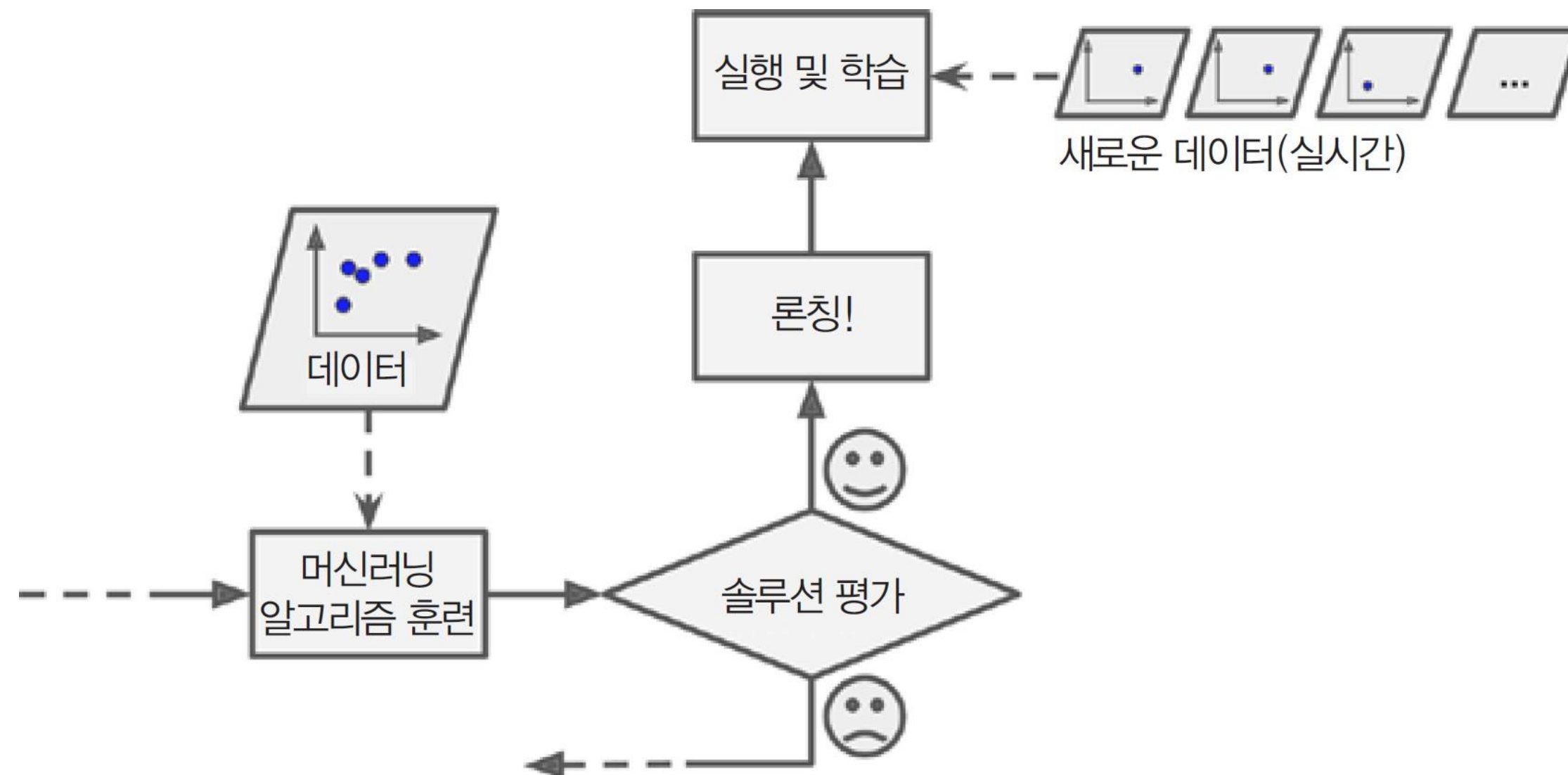


배치(batch) 학습

- 학습 데이터를 모두 사용하여 훈련시키는 오프라인(offline) 학습
- 새 데이터가 추가되면 이전 학습 데이터와 합쳐 새로 학습함.
- 일반적으로 시간과 자원이 많이 소모됨
- 훈련과 모델 론칭을 자동화할 수도 있음
 - 예) 새 데이터를 추가해서 24시간, 혹은 1주일마다 학습

온라인(online) 학습

- 샘플 한 개 또는 미니배치(mini-batch)라 부르는 작은 묶음 단위로 훈련
➔ 추가학습(incremental learning)이 더 올바른 표현
- 학습 단계가 빠르고 데이터가 준비되는 대로 즉시 학습할 수 있음(주식 가격?)
- 사용한 샘플을 버릴 수도 있고 보관할 수도 있음

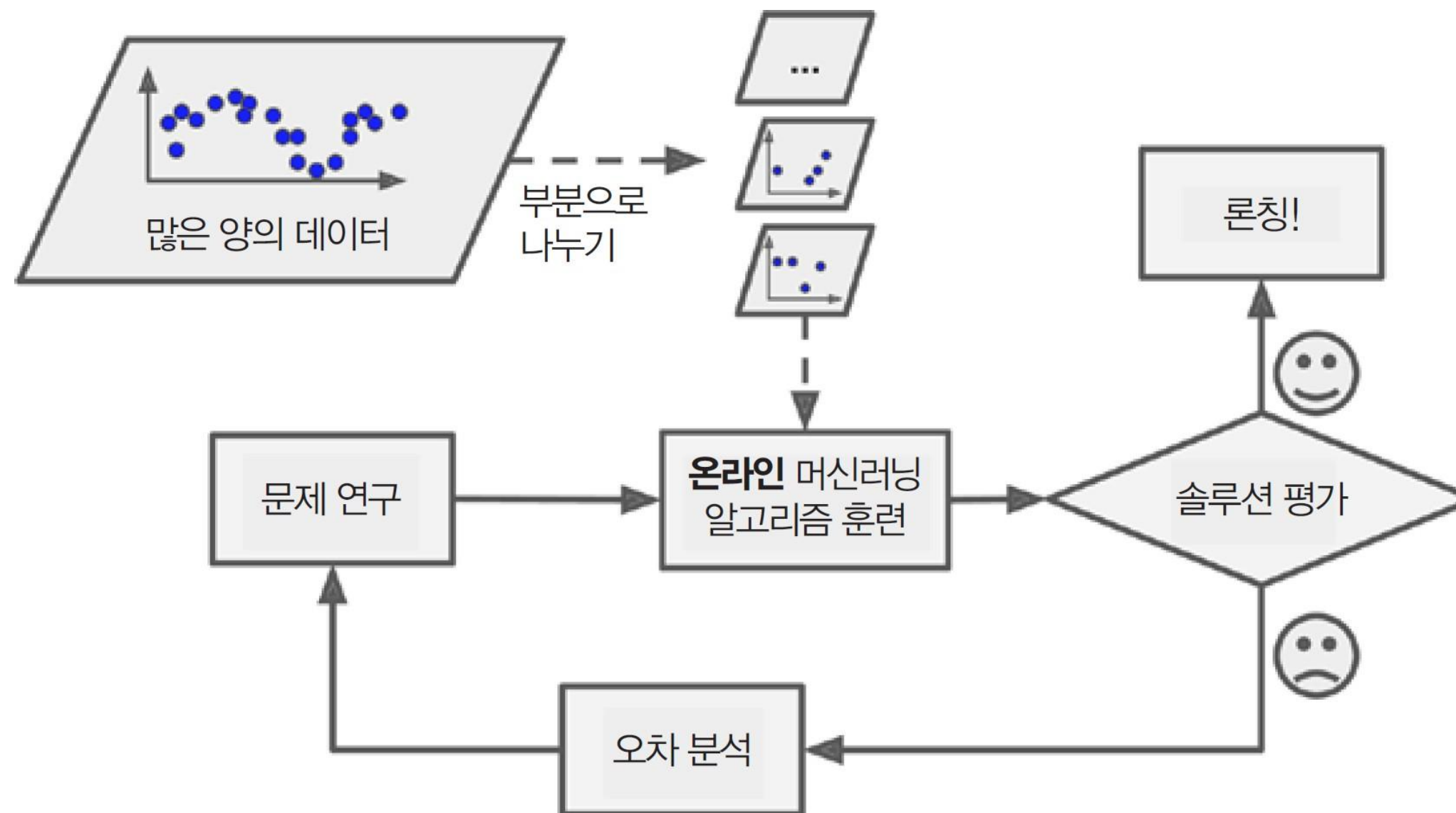


온라인 학습의 주의 사항

- 학습률(learning rate)로 데이터에 얼마나 빠르게 적응할지 제어
- 나쁜 데이터가 모델 학습에 주입되면 성능이 조금씩 감소됨
- 시스템 모니터링이 필요하고 성능 감소가 감지되면 학습을 중지하고 이전 버전으로 되돌리거나 비정상적인 데이터를 찾을

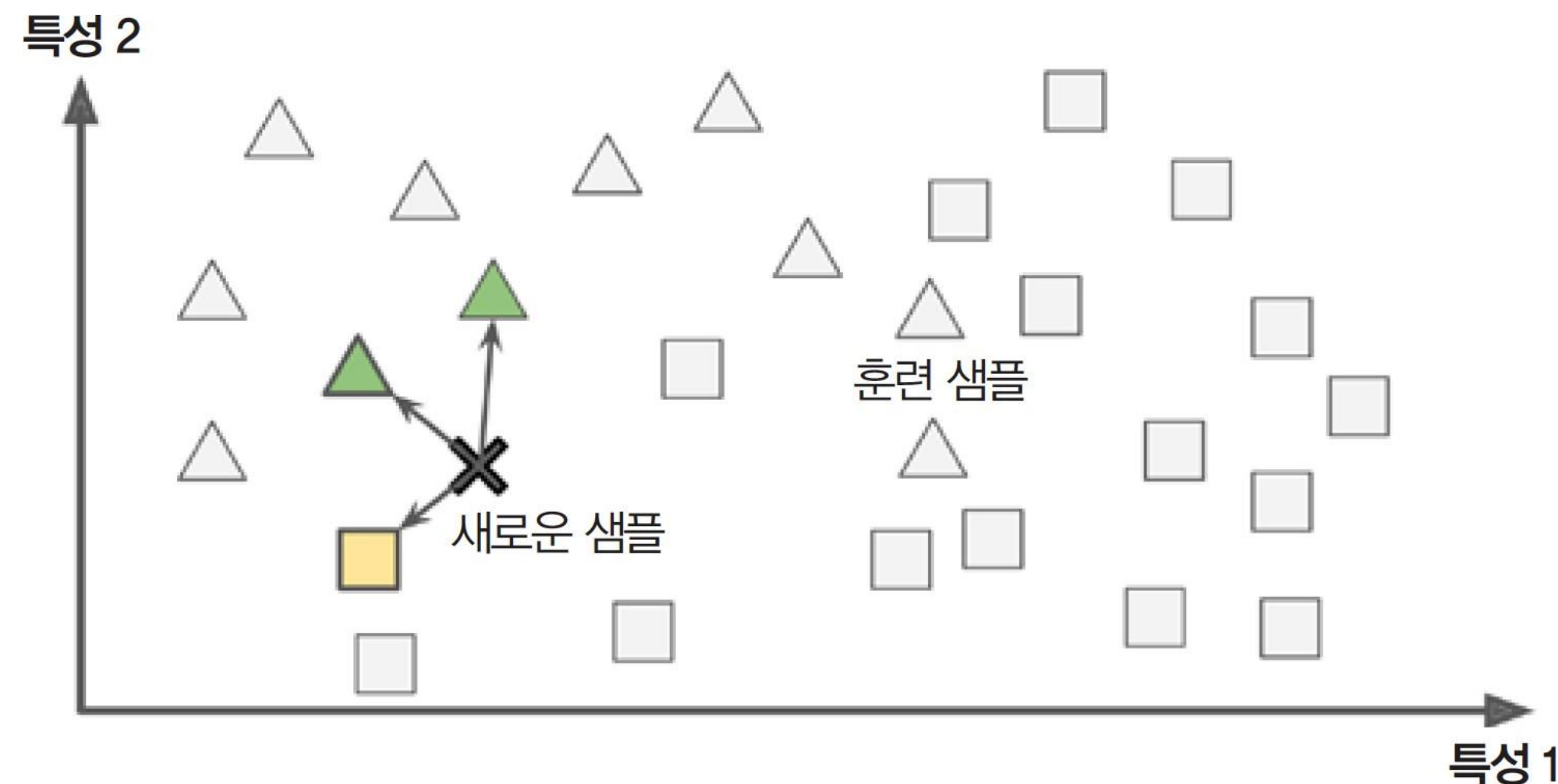
외부 메모리(ooc) 학습

- 데이터를 부분으로 나누어 오프라인에서 학습
- Out-of-core learning
- 딥러닝 처럼 데이터가 많은 경우 사용 : 데이터를 작은 배치(Batch)로 나누어 학습



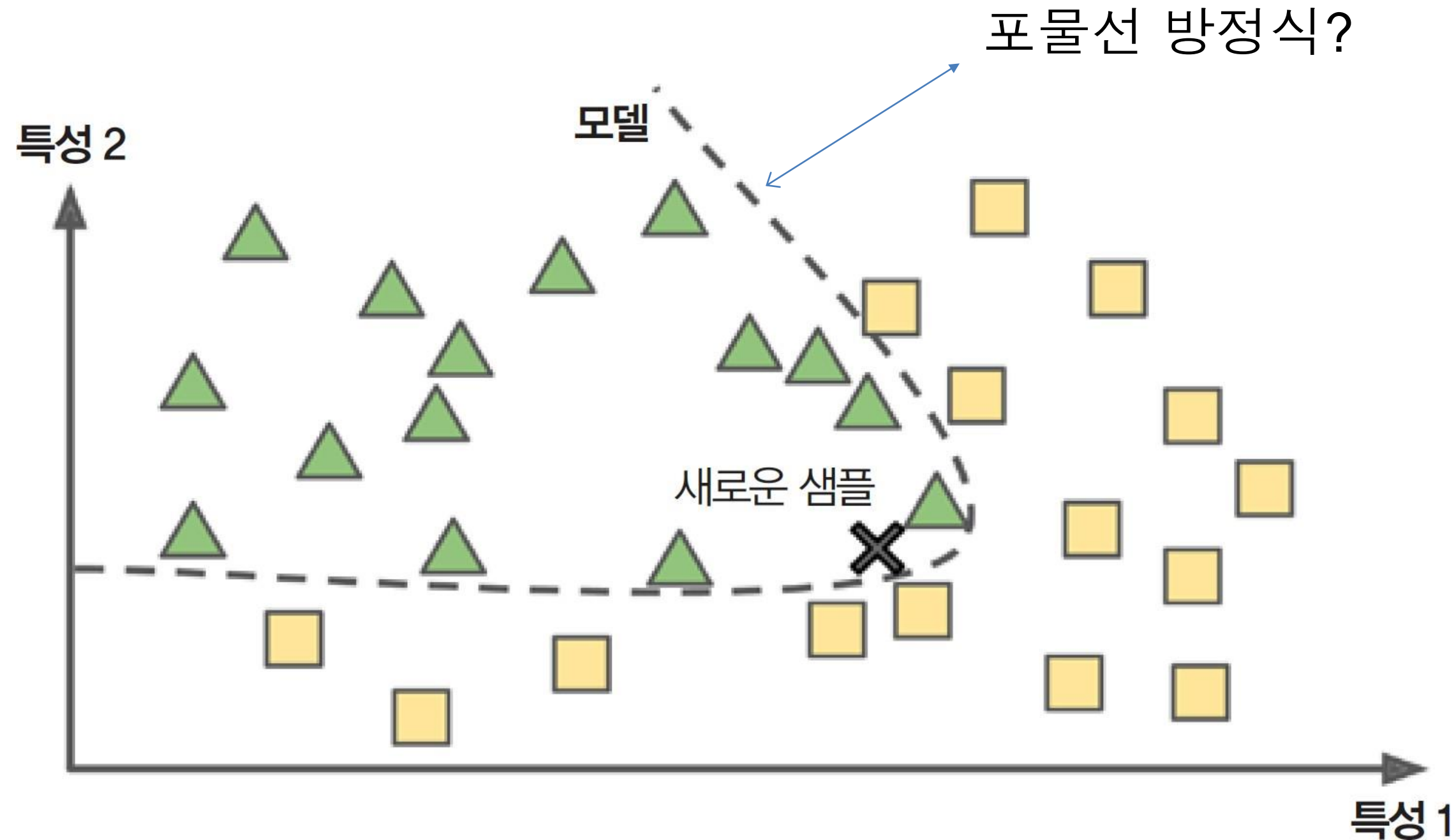
사례 기반 학습 (instance-based learning)

- 사례 기반(instance-based) : 학습데이터를 그대로 사용
모델 기반(model-based) : 학습데이터를 대표하는 모델을 만들어 그 모델을 학습함
- k-최근접 이웃(k-Nearest Neighbors) : 저장된 훈련 데이터에서 가장 가까운 샘플을 찾음. (학습이 필요 없거나 간단함) → 사례기반 학습



모델 기반 학습

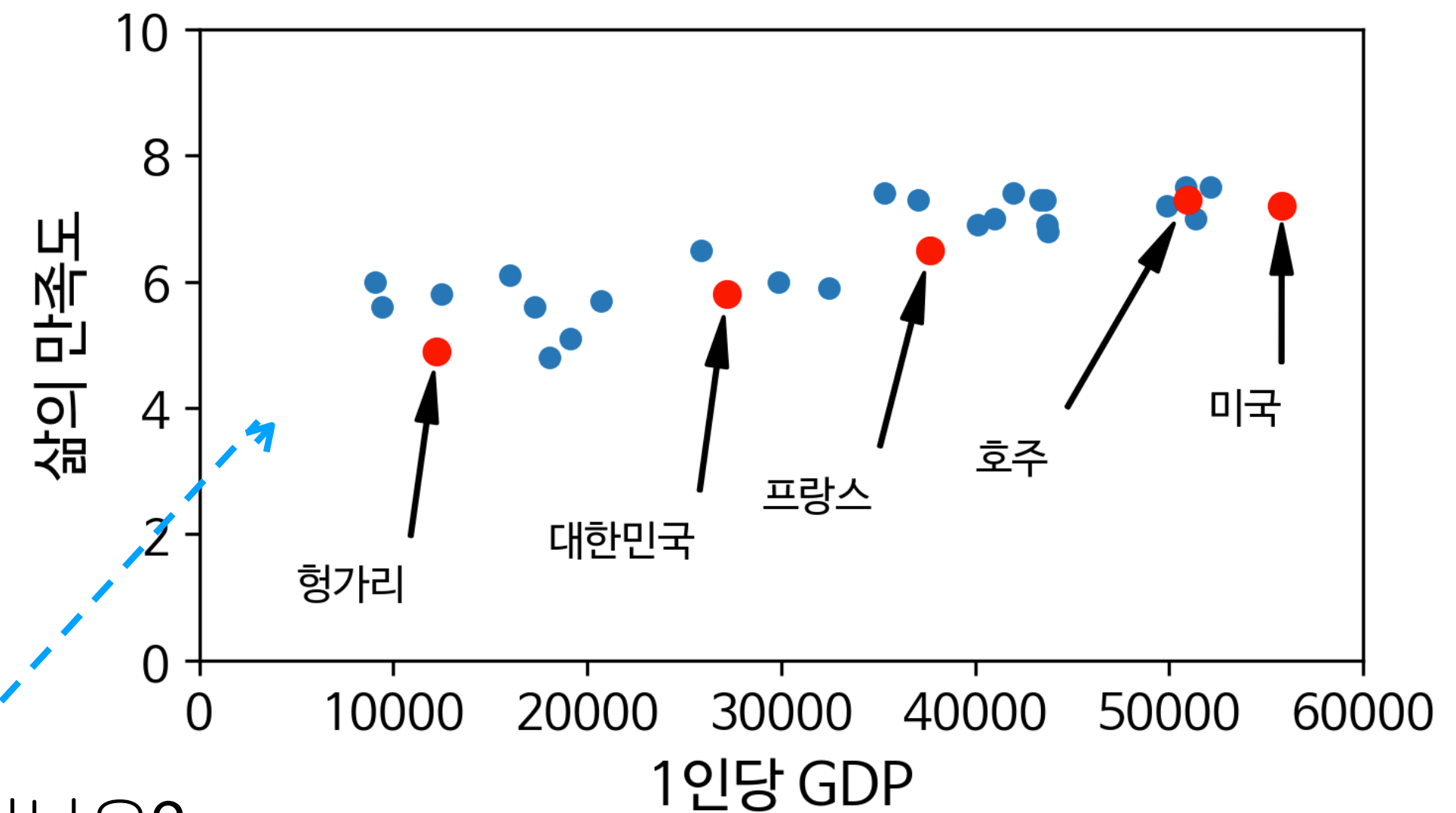
- 모델 기반(model-based) : 학습데이터를 대표하는 모델을 만들어 그 모델을 학습함
- 대부분 모델은 파라미터(parameter)를 가지고 있음 → 학습은 최적의 파라미터값을 결정하는 것
- 거의 대부분의 머신러닝은 모델 기반 학습



1인당 GDP에 대한 삶의 만족도

- OECD 웹사이트에서 '더 나은 삶의 지표'와 IMF 웹사이트에서 '1인당 GDP' 데이터를 다운로드하여 사용 : 교재에서 제공하는 code 참조 <https://github.com/ageron/handson-ml>

국가	1인당 GDP(US달러)	삶의 만족도
헝가리	12,240	4.9
대한민국	27,195	5.8
프랑스	37,675	6.5
호주	50,962	7.3
미국	55,805	7.2

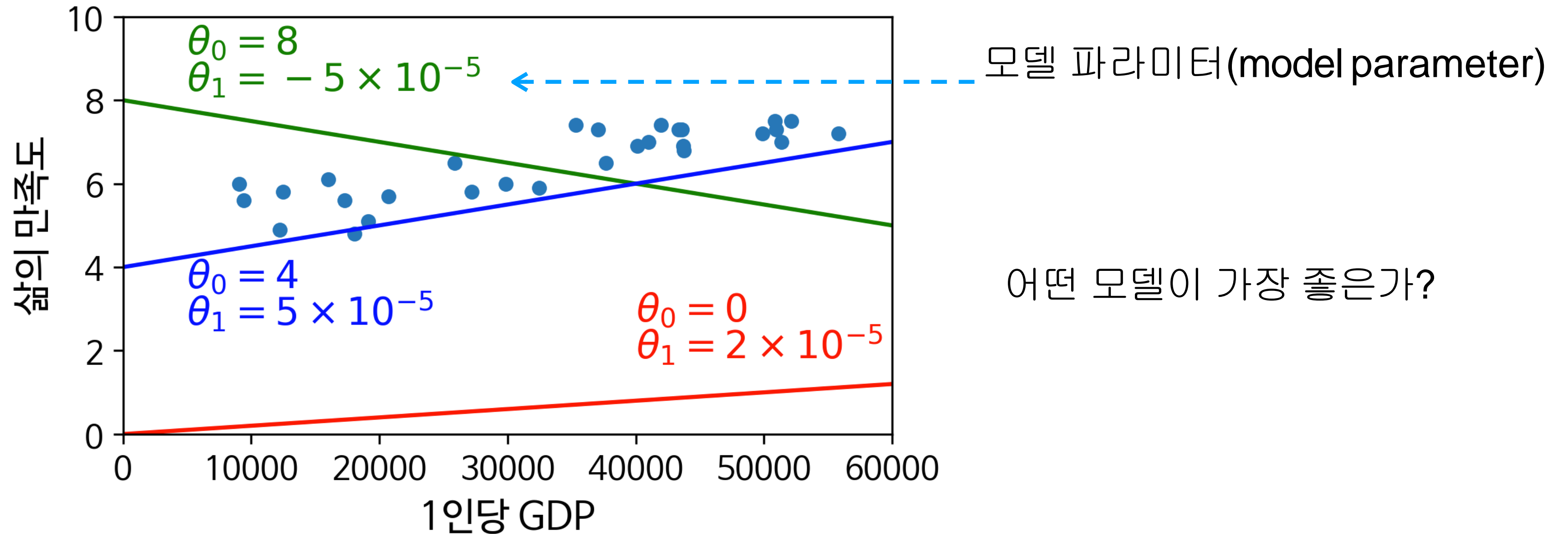


어떤 경향이 보이나요?

모델 선택

- 1인당 GDP의 선형 함수(linear function)로 삶의 만족도를 모델링 : 선형 회귀

$$\text{삶의 만족도} = \theta_0 + \theta_1 \times \text{1인당 GDP}$$



비용 함수(Cost Function)

- 좋은 모델을 고르기 위해 얼마나 나쁜지를 측정.
- 목적 함수(Object function) > 비용 함수 > 손실 함수(Loss function)
- 선형 회귀에서의 비용함수 : 예측과 타깃 사이의 거리의 합
➔ 비용이 클 수록 나쁜 모델
- 주어진 데이터에서 비용 함수의 값이 가장 작아지는 모델 파라미터(θ_0, θ_1)를 찾는 과정을 훈련(training) 또는 학습(learning)이라고 함
- 학습된 모델을 이용해서 입력 샘플에 대한 출력값을 계산하는 과정은 예측(predict) 혹은 시험(test)이라고 함

훈련(학습)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model
```

```
# 데이터 적재
```

```
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.csv", thousands=',', delimiter='\t',
                             encoding='latin1', na_values="n/a")
```

```
# 데이터 준비
```

```
country_stats = prepare_country_stats(oecd_bli, gdp_per_capita)
X = np.c_[country_stats["GDP per capita"]]
y = np.c_[country_stats["Life satisfaction"]]
```

```
# 데이터 시각화
```

```
ax = country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
ax.set(xlabel="삶의 만족도", ylabel="1인당 GDP")
plt.show()
```

```
# 선형 모델 선택
```

```
model = sklearn.linear_model.LinearRegression()
```

```
# 모델 훈련
```

```
model.fit(X, y)
```

oecd_bli, gdp_per_capit 를 나라이름으로 병합한 후, "GDP per capita " 와 'Life satisfaction' 등 두개 컬럼만 남김

```
def prepare_country_stats(oecd_bli, gdp_per_capita):
    oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
    oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
    gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
    gdp_per_capita.set_index("Country", inplace=True)
    full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita,
                                  left_index=True, right_index=True)
    full_country_stats.sort_values(by="GDP per capita", inplace=True)
    remove_indices = [0, 1, 6, 8, 33, 34, 35]
    keep_indices = list(set(range(36)) - set(remove_indices))
    return full_country_stats[["GDP per capita", 'Life satisfaction']].iloc[keep_indices]
```

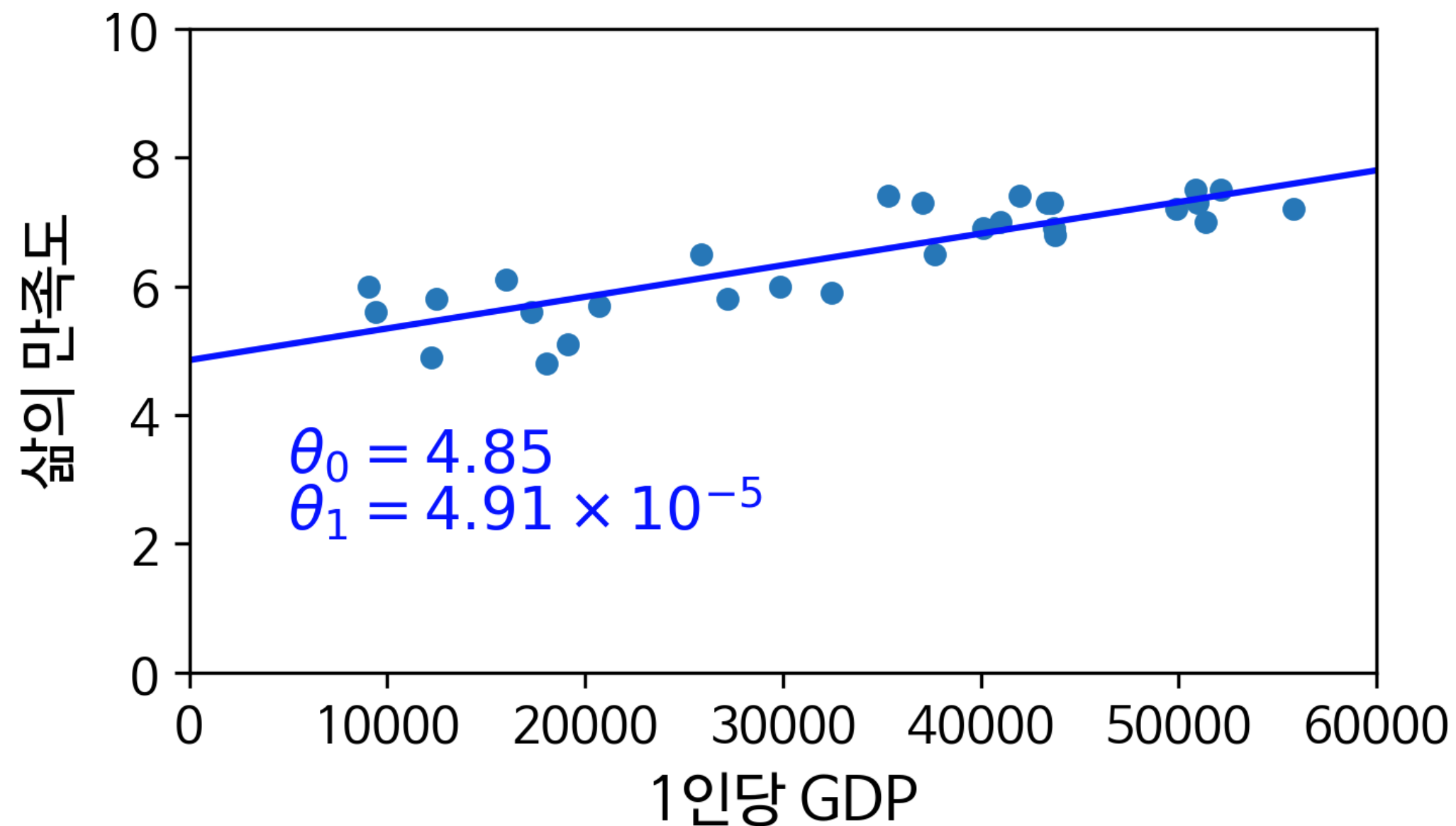
파일 읽어들이기 폴더이름 설정

```
import os
datapath = os.path.join("datasets", "lifesat", "")
```

학습결과 : 모델 파라미터

- model.intercept_, model.coef_

$$\text{삶의 만족도} = 4.85 + 4.91 \times 10^{-5} \times \text{1인당 GDP}$$

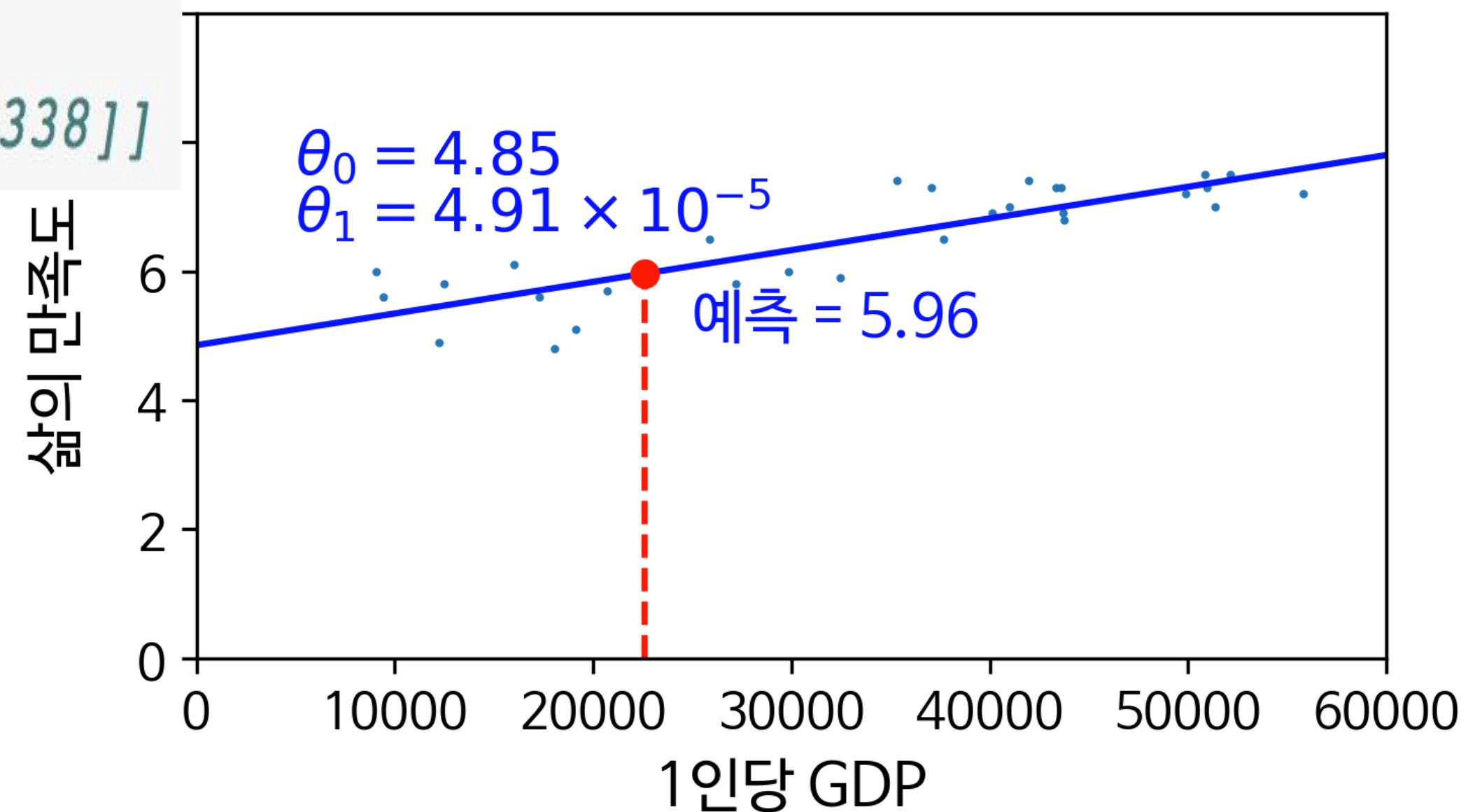
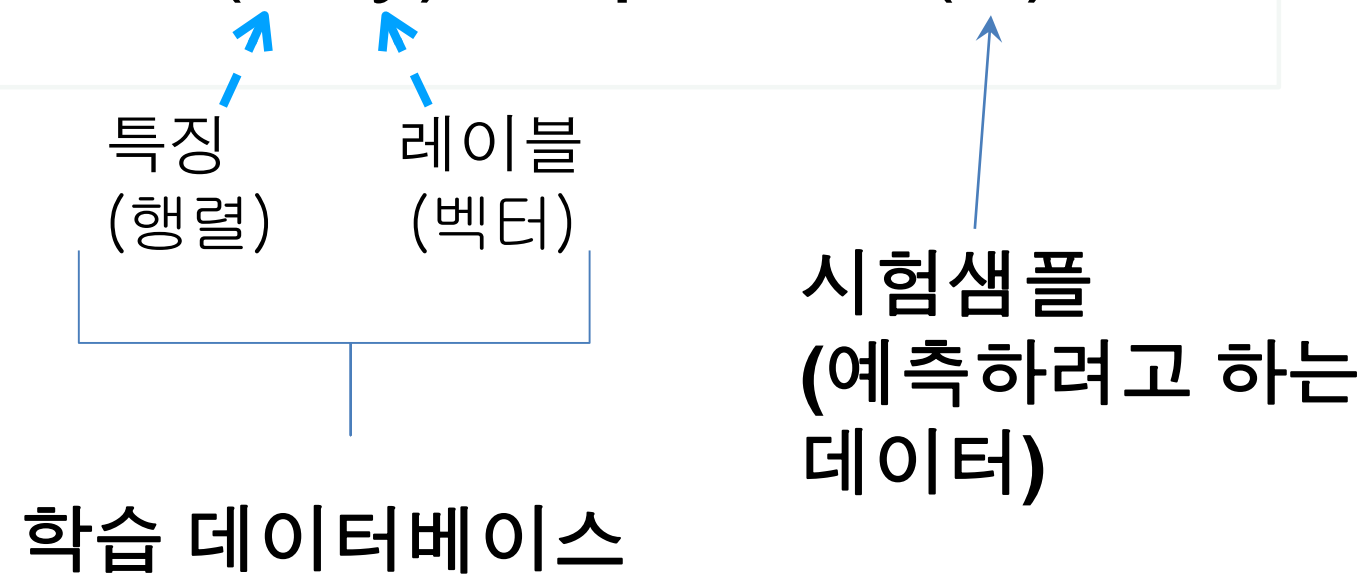


예측 : 키프로스의 삶의 만족도?

- 키프로스의 1인당 GDP는 \$22,587
삶의 만족도 = $4.85 + 4.91 \times 10^{-5} \times 22587 = 5.96$

```
# 키프로스에 대한 예측  
X_new = [[22587]] # 키프로스 1인당 GDP  
print(model.predict(X_new)) # 결과 [[ 5.96242338]]
```

- Scikit-Learn: $\text{fit}(X, y) \rightarrow \text{predict}(X)$



k-최근접 이웃을 사용하면?

- k=3일 때, 키프로스와 가장 가까운 슬로베니아, 포르투갈, 스페인의 삶의 만족도를 평균
- $\text{fit}(X, y) \rightarrow \text{predict}(X)$

```
# 선형 회귀 모델을 k-최근접 이웃 회귀 모델로 교체할 경우
knn = sklearn.neighbors.KNeighborsRegressor(n_neighbors=3)

# 모델 훈련
knn.fit(X, y)

# 키프로스에 대한 예측
print(knn.predict(X_new)) # 결과 [[ 5.76666667]]
```

무언가 잘 못 되었다면?

- 특징 : 더 많은 특징(고용률, 건강, 대기오염 등)을 사용
- 데이터 : 좋은 훈련 데이터를 더 많이 모음
- 모델 : 더 강력한 모델(다항 회귀 모델 등)을 선택

머신러닝 작업 요약

- 데이터를 분석
- 모델을 선택
- 훈련 데이터로 모델을 훈련. (비용 함수를 최소화하는 모델 파라미터를 찾음)
- 새로운 데이터에 대한 예측(추론)을 함. (좋은 일반화를 기대)
- 그리고 반복!

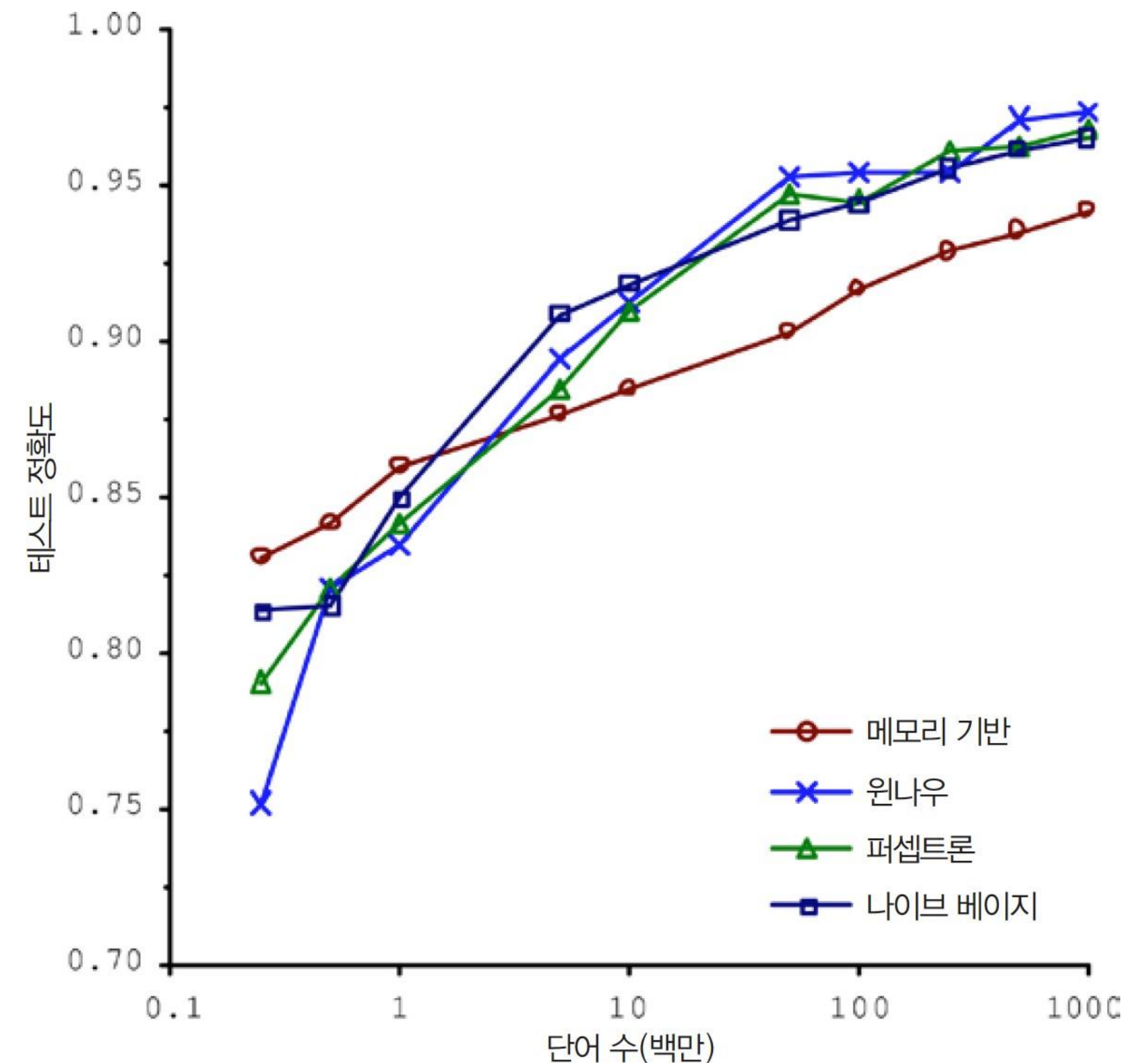
추론(inference) : 특징과 결과와의 관계에 중점,
예측(prediction) : 결과값에 중점 ➔ 이 두 단어는 같은 의미로 사용

머신러닝의 도전 과제

- 나쁜 데이터
 - 충분하지 않은 훈련 데이터
 - 대표성 없는 훈련 데이터
 - 낮은 품질의 데이터
 - 관련 없는 특성
- 나쁜 알고리즘
 - 과대적합
 - 과소적합

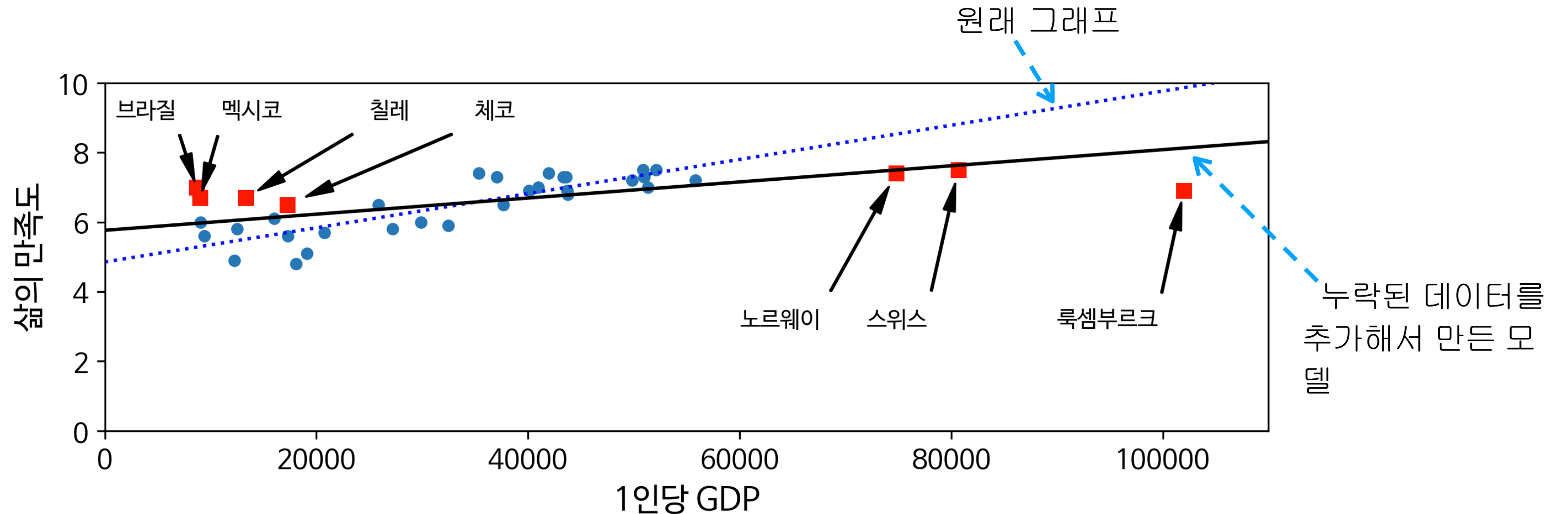
충분하지 않은 훈련 데이터

- 어린 아이는 ‘사과’ 샘플 몇 개를 보고도 모든 종류의 사과를 쉽게 일반화 함
- 머신러닝은 데이터로부터 일반화 규칙 학습
➔ 많은 데이터가 필요함
- 2001년 MS 연구자들은 충분한 데이터가 주어지면 알고리즘들이 복잡한 자연어 처리 문제를 거의 비슷하게 잘 처리한다는 것을 보임 : 데이터 중요 (실제로는 데이터가 제한적이므로 알고리즘도 중요함)



대표성 없는 훈련 데이터

- 삶의 만족도 모델에 누락된 데이터를 추가하면 그래프가 달라짐



- 샘플링 잡음(sampling noise): 샘플이 작거나 이상치가 포함된 경우
- 샘플링 편향(sampling bias): 표본 추출 방법이 잘 못 된 경우

가장 유명한 샘플링 편향 사례

- 1936년 랜던과 루즈벨트의 대통령 선거에서 Literary Digest 사의 여론조사(랜던 57% 예측)
- 실제로는 루즈벨트가 60.8% 득표로 당선됨
→ 샘플링 편향으로 잘못된 결과
 - 1) 전화번호부, 구독자 명부, 클럽 회원 명부를 사용
→ 랜던에 투표할 가능성이 높은 부유층
 - 2) 우편물 수신자 중 25% 미만이 응답
→ 정치에 관심없는 사람, 조사기관(Literary Digest)을 싫어하는 사람이 제외됨 (비응답 편향)



낮은 품질의 데이터

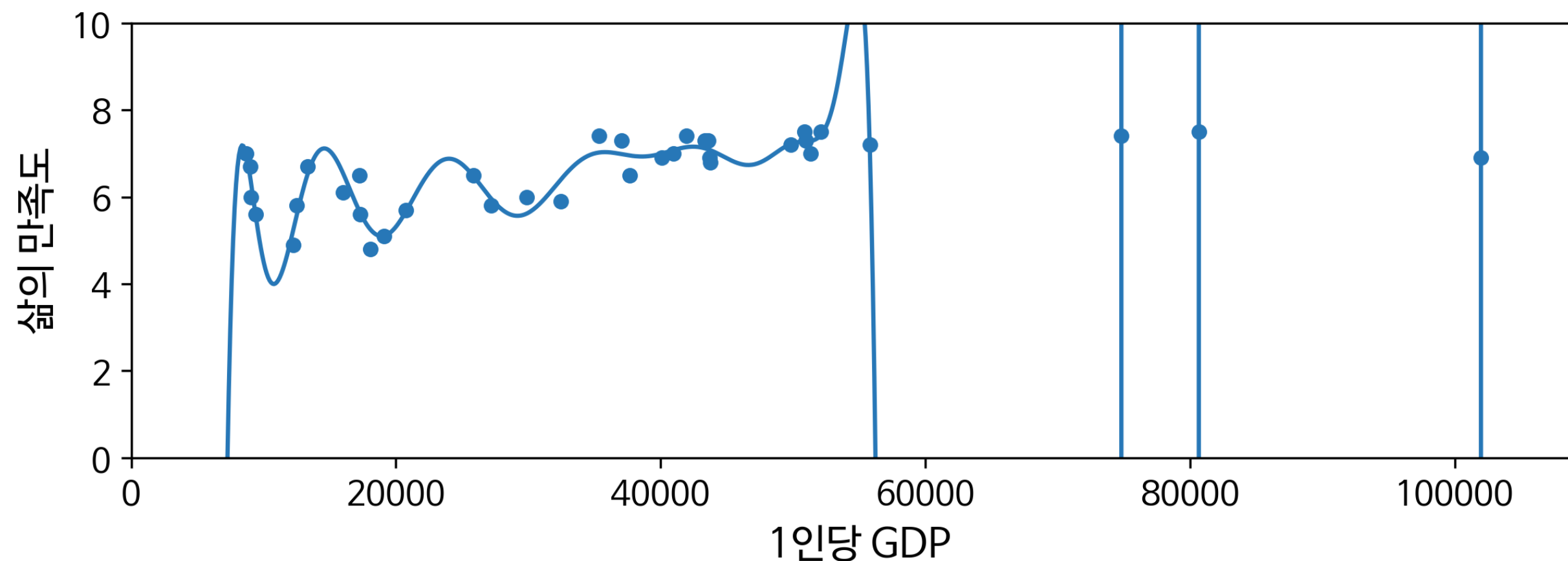
- 일부 샘플이 이상치일 경우 무시하거나 수동으로 고침.
- 일부 샘플에서 특성이 몇 개 빠져 있다면, 특성 전체를 무시할지, 샘플을 무시할지, 빠진 값을 채울지, 이 특성이 넣은 것과 뺀 것을 따로 훈련할지 정해야 함.

특징 공학

- 주어진 문제와 관련이 높은 특징을 찾음 : 특징 공학(feature engineering)
 - ➔ 특징이 많다고 잘 분류하는 것이 아님. “좋은 특징”을 사용하는 것이 중요함
 - 특징 선택: 가지고 있는 특징 중에서 가장 유용한 특징을 선택.
 - 특징 추출: 특징을 결합하여 더 유용한 특징을 만듦(차원 축소, 다항 회귀 등)
 - 새로운 데이터로부터 새로운 특징을 만듦

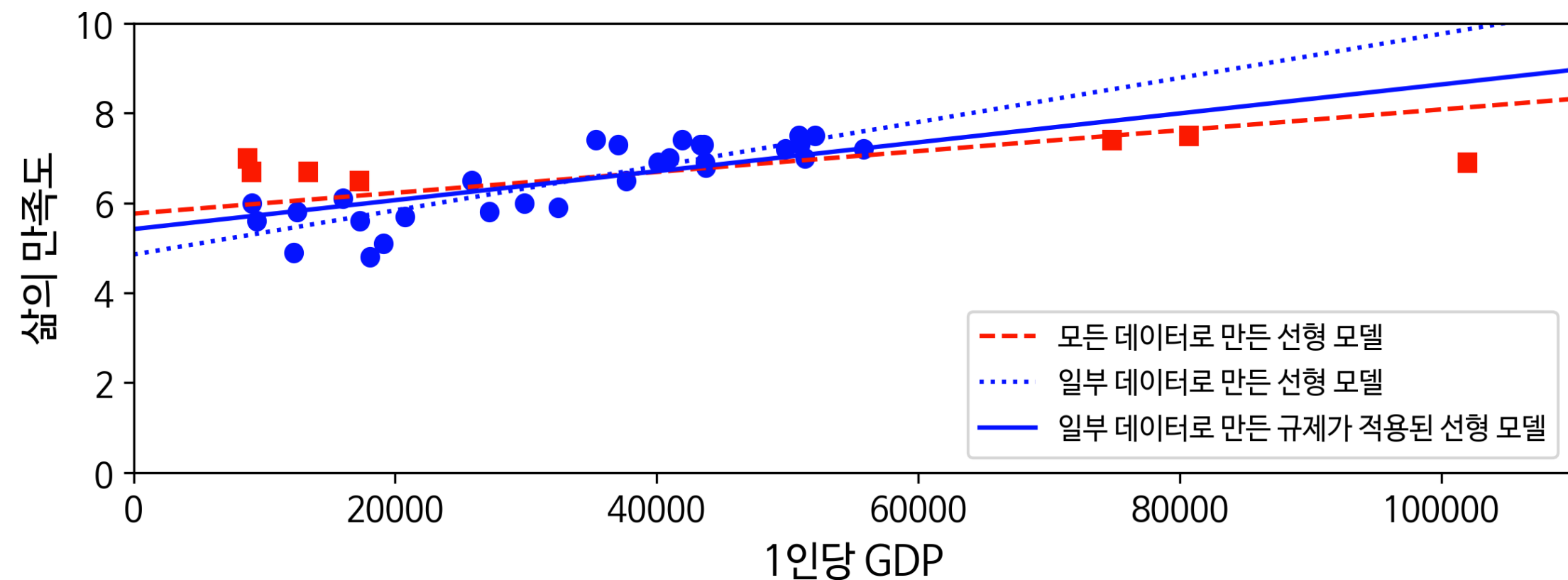
과대적합(overfitting)

- 해외 여행에서 택시 운전사에게 속았다면 그 나라의 모든 택시를 의심하게 됨
➔ 일반화의 오류 (=과대적합)
- 학습데이터(==훈련데이터)에만 잘 들어 맞음. 시험데이터에는 잘 안 맞음
- 이유 :
 - 너무 복잡한 모델을 사용
 - 잡음이 많거나 너무 적은 수의 샘플(=학습샘플=학습데이터)



과대적합을 피하려면

- 데이터 측면 :
 - 학습데이터 수 늘림
 - 학습데이터의 잡음 감소 : 오류 데이터 수정, 이상치 제거
- 모델 측면 :
 - 모델 파라미터 개수가 적은 모델(복잡도가 낮은 모델)을 선택
 - 특징 수를 줄임 : 특징공학 방법 사용
 - 모델에 regularization(정규화, 혹은 '규제'로 번역)를 추가
- Regularization : 모델 파라미터의 값에 제한 조건을 줌.
 - 크기의 절대값의 합을 제한 : L1
 - 크기의 제곱합을 제한 : L2



과소적합(underfitting)

- 과대적합의 반대. 모델이 너무 단순해서 적절한 패턴을 학습하지 못함
- 해결 방법
 - 모델 파라미터가 더 많은 (복잡도가 높은) 모델을 선택
 - 더 좋은 특징 사용 : 특징 공학
 - 모델의 Regularization 축소

Regulariazation을 얼마나 할 것인가는 사용자가 결정하는 파라미터임.
이는 학습에 의해 결정되는 모델 파라미터와 다름.
학습되지 않고 결정되는 파라미터를 하이퍼파라미터(Hyperparameter)라고 함

한걸음 물러서서

- 머신러닝은 명시적인 규칙을 코딩하지 않고 데이터에서 학습함
- 지도 학습 / 비지도 학습,
배치 학습 / 온라인 학습,
사례 기반 학습 / 모델 기반 학습
- 학습 : 학습 데이터를 잘 표현하는 모델 파라미터를 찾음
- 데이터 : 개수가 너무 작거나, 대표성이 없거나, 잡음이 많거나, 관련 없는 특징이 많으면 올바른 모델을 학습하지 못함
- 모델이 너무 복잡하거나(과대적합), 단순하지(과소적합) 않아야 함.

테스트 세트와 검증 세트

- 훈련된 모델을 검증하기 위해 따로 떼어 놓은(훈련에 사용하지 않은) 테스트 세트를 사용해야 함
- 훈련 세트(training set): 80%, 테스트 세트(test set): 20%
- 하이퍼파라미터를 조정하기 위해 검증세트(validation set)가 사용되기도 함
(테스트 세트를 사용하여 하이퍼파라미터를 조정하면 모델이 테스트 세트에 과대적합됨)
예) 훈련세트 60%, 검증세트 20%, 테스트세트 20%
 - 훈련 세트와 검증 세트를 교번하여 여러번 검증 점수를 계산(교차 검증(cross-validation)).

공짜 점심 없음(no free lunch)

- 어떤 가정도 없다면 특정 모델이 뛰어나다고 판단할 근거가 없다는 이론 : 데이비드 월퍼트(1996)
- 주어진 데이터셋에 선형 모델이 잘 맞을지 신경망이 잘 맞을지 경험하기 전에 알 수 없음 : 결국 모두 시도해 보아야 함
- 일반적인 모델 선정 가이드 :
 - 정형화된 데이터 : 트리 기반 앙상블
 - 시각에 관련된 데이터(이미지, 텍스트, 사운드) : 신경망