

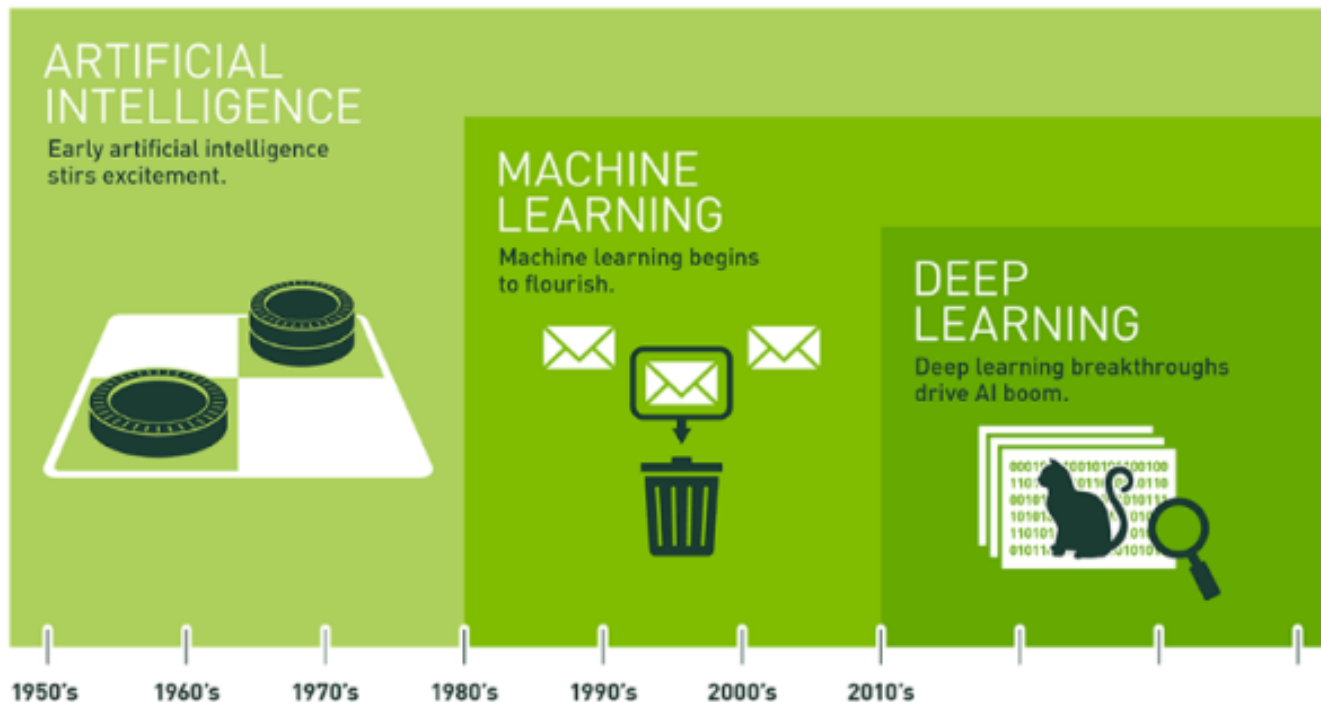
머신러닝 프로젝트

Seolyoung Jeong, Ph.D.

경북대학교 IT 대학

인공지능의 개념

- ◆ **인공지능** : 인간의 감각, 사고력을 지닌 채 인간처럼 생각하는 컴퓨터
- ◆ **머신 러닝** : 인공 지능을 구현하는 구체적 접근 방식.
알고리즘을 이용해 데이터를 분석하고, 분석을 통해 학습하고, 학습한 내용을 기반으로 판단이나 예측 수행
- ◆ **딥 러닝** : 완전한 머신 러닝을 실현하는 기술.
인공신경망에서 발전한 형태의 인공 지능.
뇌의 뉴런과 유사한 정보 입출력 계층을 활용



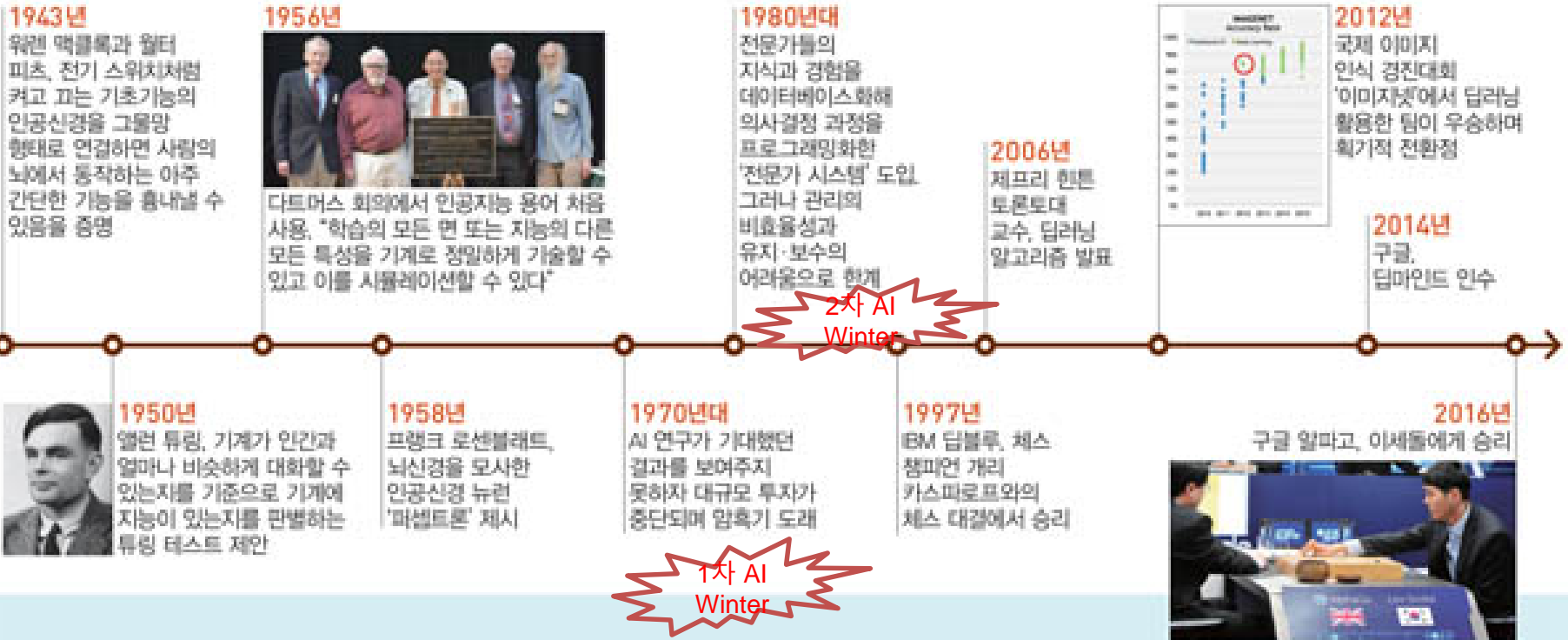
인공지능의 역사

◆ 두 번의 AI 겨울

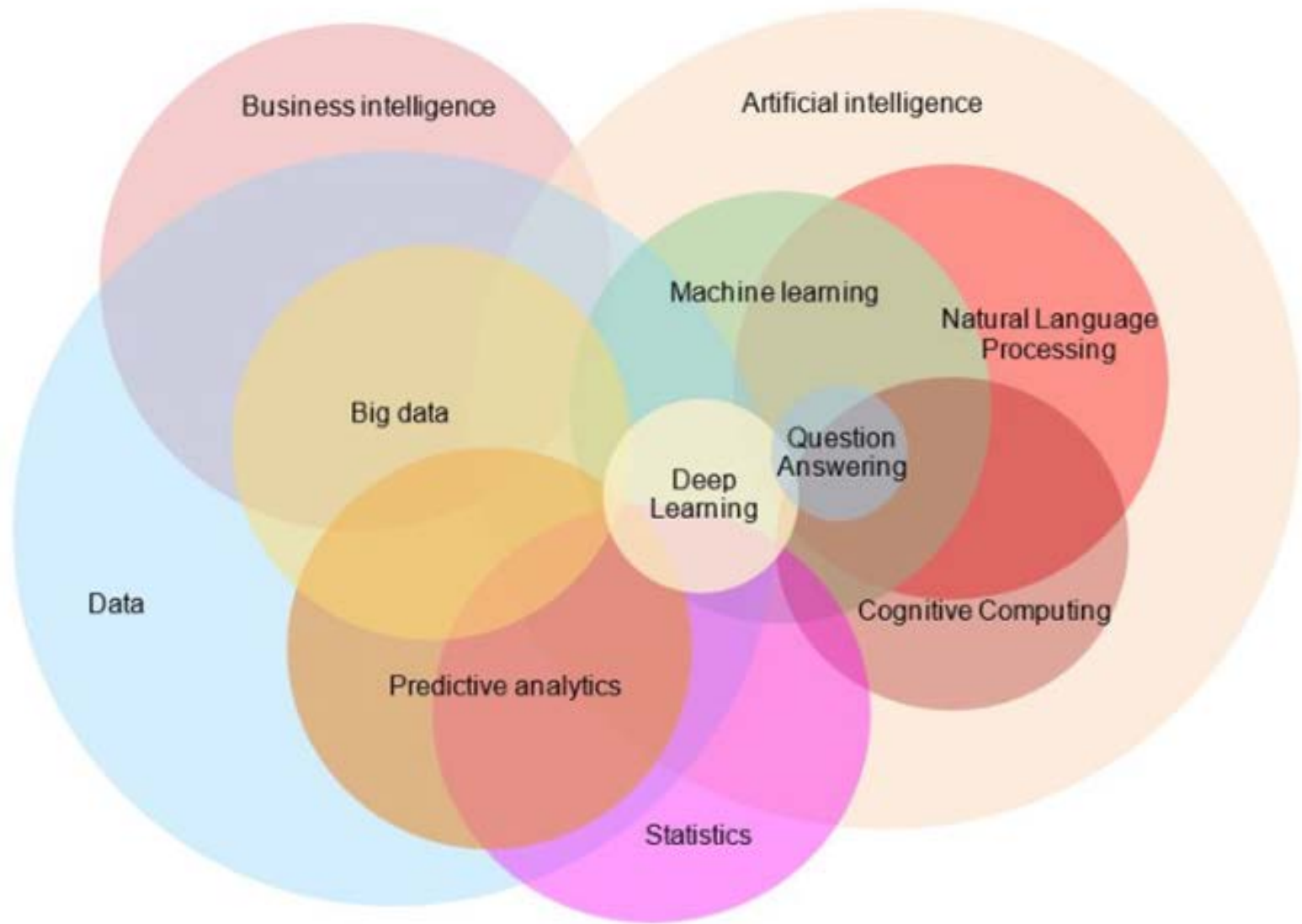
- 1960년대 말 ~ 1970년대 : 부족한 컴퓨터 기술, XOR 문제
- 1980년대 후반 ~ 1990년대 초 : 자연어 처리의 한계 (컴퓨터는 의미를 이해하지 못함)

◆ 세 번의 AI 봄 (현재는 3차. 발전된 하드웨어 성능 및 이미지넷 등 딥러닝 개발)

인공지능(AI)의 역사

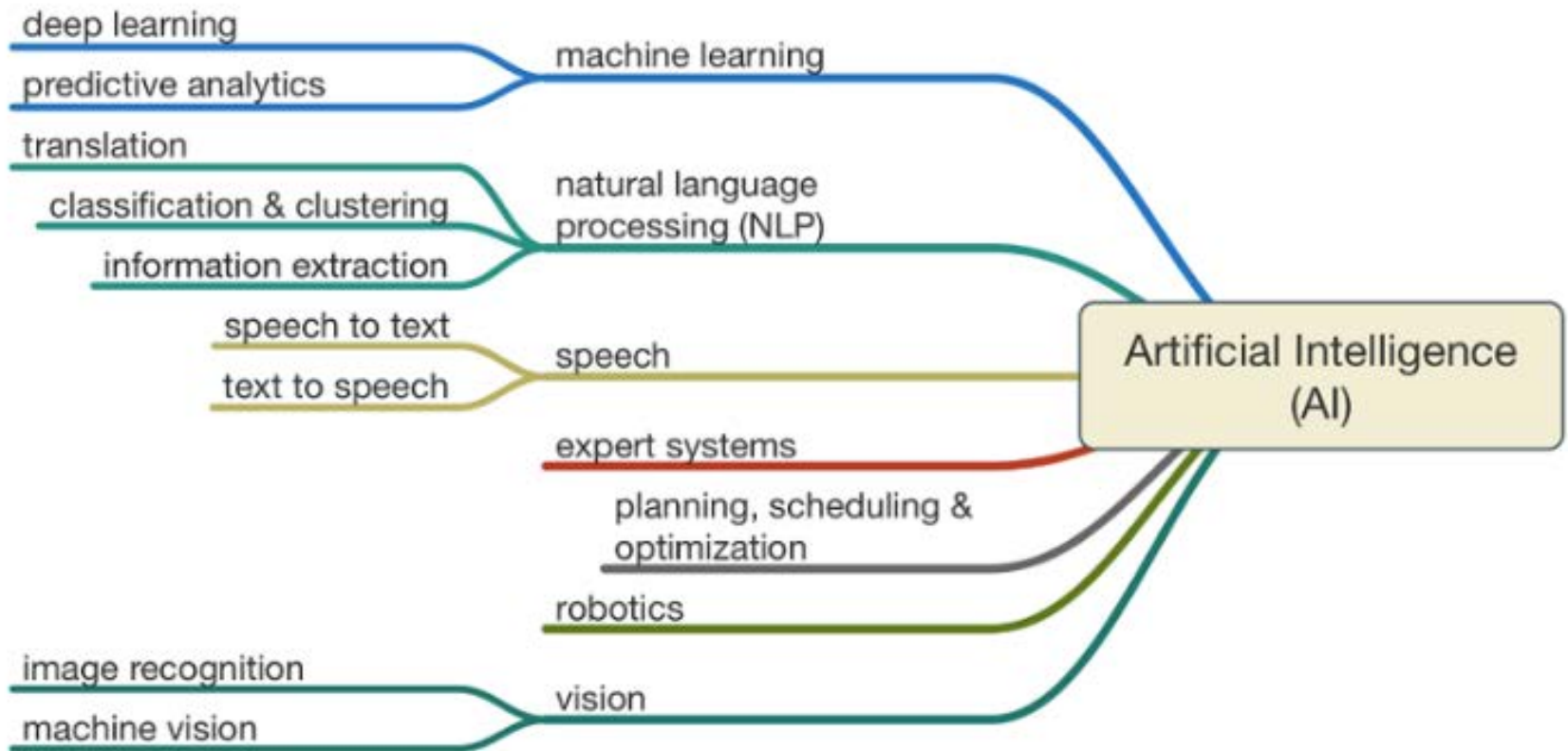


인공지능 연구 기술



인공지능 연구분야

- ◆ 자연어 처리, 음성인식, 전문가 시스템, 로봇, 컴퓨터 비전, 의료 등 다양
- ◆ 기계가 지능적으로 움직이기 위해 필요한 기술 연구



Contents

1.1 머신러닝이란?

1.2 왜 머신러닝을 사용하는가?

1.3 머신러닝 시스템의 종류

1.3.1 지도 학습과 비지도 학습

1.3.2 배치 학습과 온라인 학습

1.3.3 사례 기반 학습과 모델 기반 학습

1.4 머신러닝의 주요 도전 과제

1.4.1 충분하지 않은 양의 훈련 데이터

1.4.2 대표성 없는 훈련 데이터

1.4.3 낮은 품질의 데이터

1.4.4 관련 없는 특성

1.4.5 훈련 데이터 과대적합

1.4.6 훈련 데이터 과소적합

1.5 테스트와 검증

1.1 머신러닝이란?

- ◆ 데이터로부터 학습하도록 컴퓨터를 프로그래밍하는 과학
- ◆ 기계 학습(機械學習) 또는 머신 러닝(영어: machine learning)
 - 인공지능의 한 분야로, 컴퓨터가 학습할 수 있도록 하는 알고리즘과 기술을 개발하는 분야
 - 예) 기계 학습을 통해서 수신한 이메일이 스팸인지 아닌지를 구분할 수 있도록 훈련 [출처: 위키백과 기계학습]
- ◆ 많은 양의 데이터를 분석하기 위해서는 먼저 시각화가 필요
- ◆ 데이터 시각화를 위한 라이브러리 :
 - R(ggplot), SQL, Zepplin(라이브러리 아님)
 - Python – Matplotlib (그래프/차트 도식화), Numpy (배열, 벡터 계산), Pandas (Numpy를 기반으로 개발. 데이터 정렬 가능한 자료구조 Dataframe)

1.1 머신러닝이란?

- ◆ 1959년, 아서 사무엘은 기계 학습을 "기계가 일일이 코드로 명시하지 않은 동작을 데이터로부터 학습하여 실행할 수 있도록 하는 알고리즘을 개발하는 연구 분야"라고 정의하였다. [위키백과:기계학습 정의]
- ◆ 어떤 **작업 T**에 대한 컴퓨터 프로그램의 **성능을 P**로 측정했을 때 **경험 E**로 인해 성능이 향상됐다면, 이 컴퓨터 프로그램은 작업 T와 성능 측정 P에 대해 E로 학습한 것이다. [토미첼, 1997]

구분	작업 T	경험 E	성능측정 P
스팸 메일 시스템	새로운 메일이 스팸인지 구분	훈련데이터	정확도

시스템이 학습하는데 사용하는 샘플 : 훈련세트 (training set)
각 훈련데이터 : 훈련사례 (training instance, 샘플)

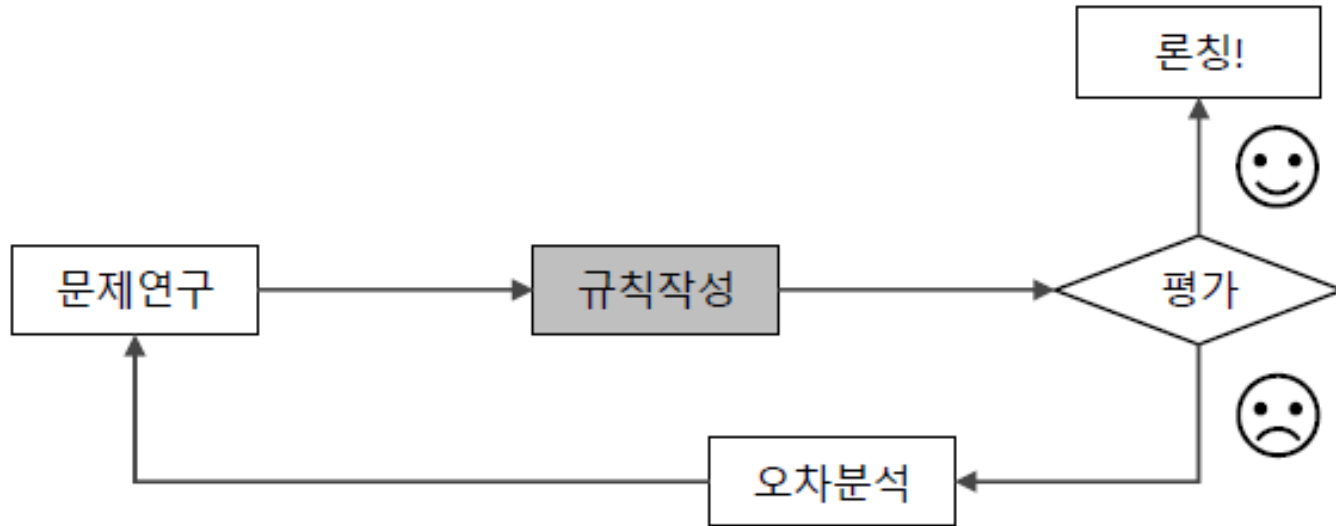
1.2 왜 머신러닝을 사용하는가?

◆ 머신러닝 기반 해결 필요한 문제들

- 기존 솔루션으로는 많은 수동 조정과 규칙이 필요한 문제
- 전통적인 방법으로는 전혀 해결 방법이 없는 복잡한 문제
- 유동적인 환경에 적응하기 어려운 문제
- 대량의 데이터와 복잡한 문제들로 해결하기 어려운 문제

전통적인 접근 방법

- ◆ 문제가 단순하지 않아 규칙이 점점 길고 복잡해지므로 유지보수가 매우 힘들

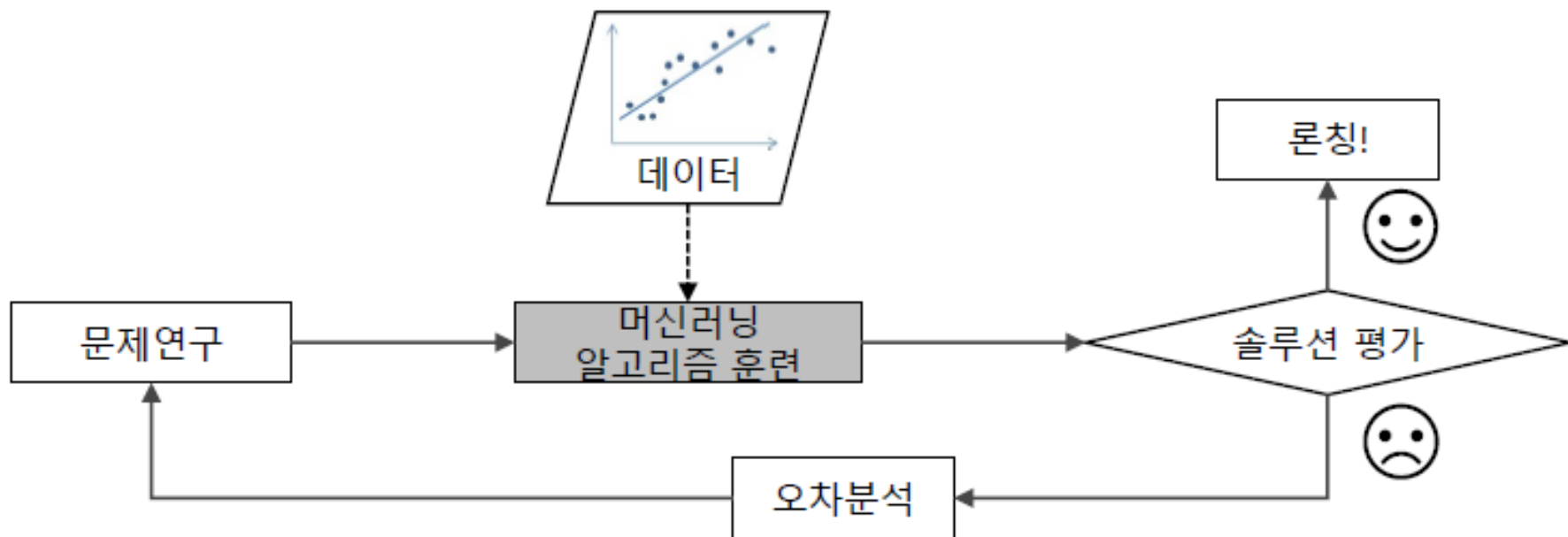


- 예) 스팸필터 : 규칙(4U, 신용카드, 무료, 대출, 광고, 대행 등)을 분석, 패턴을 감지하는 알고리즘 작성 후 테스트/적용
- 작성된 규칙 예)
if “광고” in “이메일제목” :
스팸처리

머신러닝 기반 문제 해결 장점

◆ 머신러닝 접근 방법

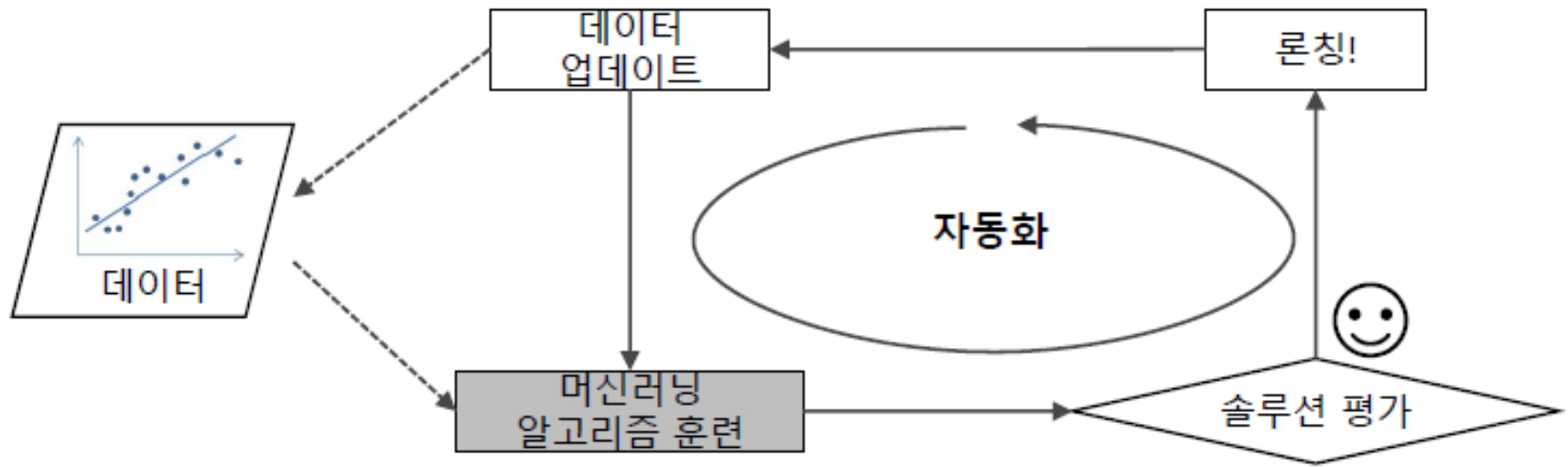
- 문제에 대한 패턴을 인지하고 학습
- 프로그램이 짧아지고 유지보수가 쉬우며 정확도를 높임



머신러닝 기반 문제 해결 장점

◆ 자동으로 변화에 적응

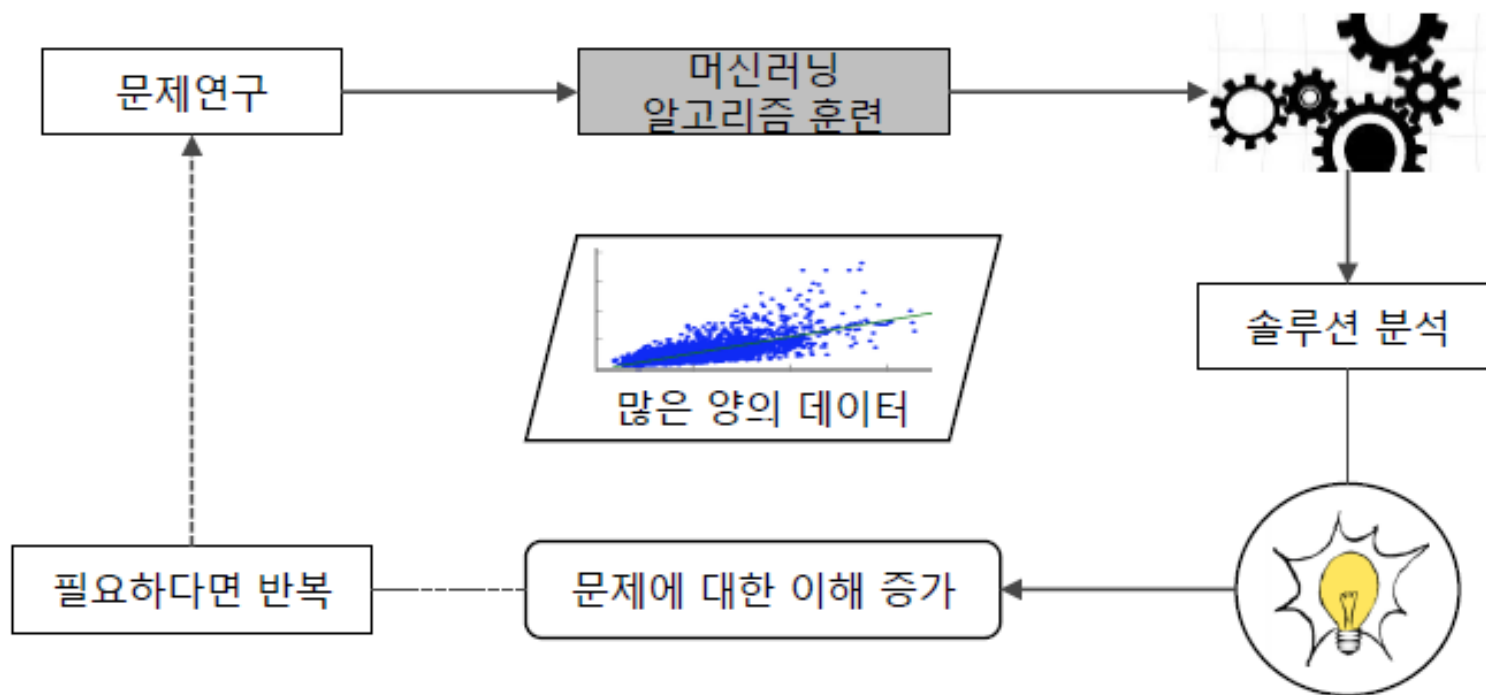
- 사용자가 지정한 데이터에서 특정 패턴을 자동으로 인식하고 별도의 작업이 없어도 분류



머신러닝 기반 문제 해결 장점

◆ 머신러닝을 통해 배우기

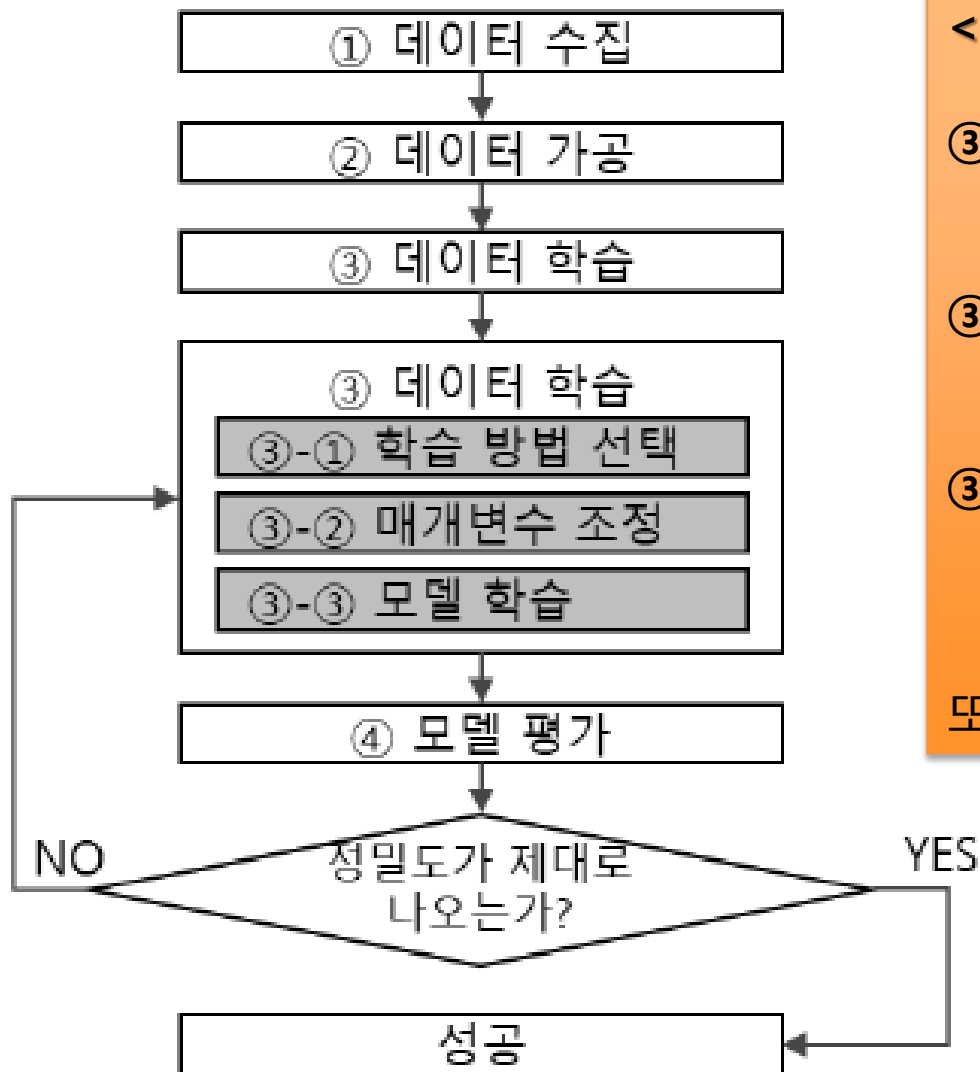
- 대용량의 데이터를 분석하면 겉으로는 보이지 않던 패턴을 발견
→ 데이터 마이닝 (Data Mining)



1.3 머신러닝 시스템의 종류

- ◆ 머신러닝 시스템의 종류는 굉장히 많으며, 아래와 같이 크게 3가지로 분류할 수 있음
 - (지도학습 / 비지도학습) 사람의 감독 하에 훈련하는 것인지, 그렇지 않은 것인지
 - (온라인 학습 / 배치 학습) 실시간으로 점진적인 학습을 하는지 아닌지
 - (사례 기반 학습 / 모델 기반 학습)
단순히 알고 있는 데이터 포인트와 새 데이터 포인트를 비교하는지,
아니면 훈련 데이터셋에서 과학자들처럼 패턴을 예측하여 예측 모델을 만드는지

머신러닝의 과정



<데이터학습>

③-① 학습방법선택 :

알고리즘 선택 (K-means, SVM 등)

③-② 매개변수 조정 :

데이터와 알고리즘에 맞게 변수 조정

③-③ 모델 학습 :

테스트 데이터를 활용해 정밀도 측정

정밀도가 나오지 않으면 매개변수 수정
또는 알고리즘 변경 이후 반복 수행

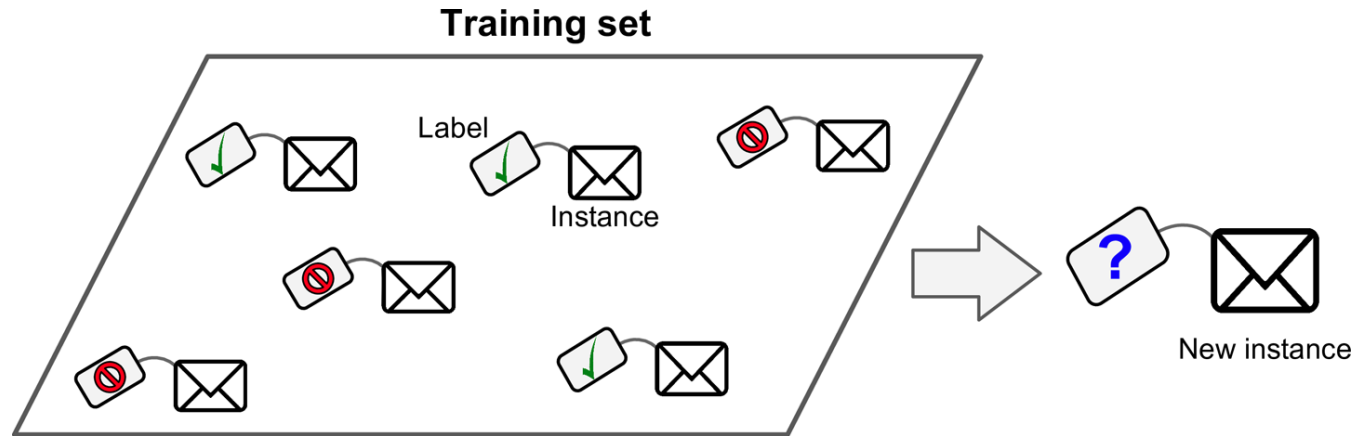
1.3.1 지도 학습과 비지도 학습

- ◆ 학습하는 동안의 감독 형태와 정보량에 따라
지도학습 / 비지도학습 / 준지도학습 / 강화학습으로 구분

종류	내용	알고리즘
지도 학습	데이터와 정답을 함께 입력	<ul style="list-style-type: none">• K-최근접 이웃• 선형회귀• 로지스틱 회귀• 서포트 벡터 머신• 결정트리와 랜덤 포레스트• 신경망
	다른 데이터의 정답을 예측	
비지도 학습	데이터는 입력하지만 정답은 미입력	<ul style="list-style-type: none">• 군집• 시각화와 차원 축소• 연관 규칙 학습
	다른 데이터의 규칙성 찾음	
강화 학습	부분적으로 정답을 입력	
	데이터를 기반으로 최적의 정답을 찾음	

지도 학습

- ◆ 알고리즘에 주입하는 훈련데이터에 레이블(LABEL)이라는 정답이 포함



- ◆ 가능한 작업1: 분류(Classfication)

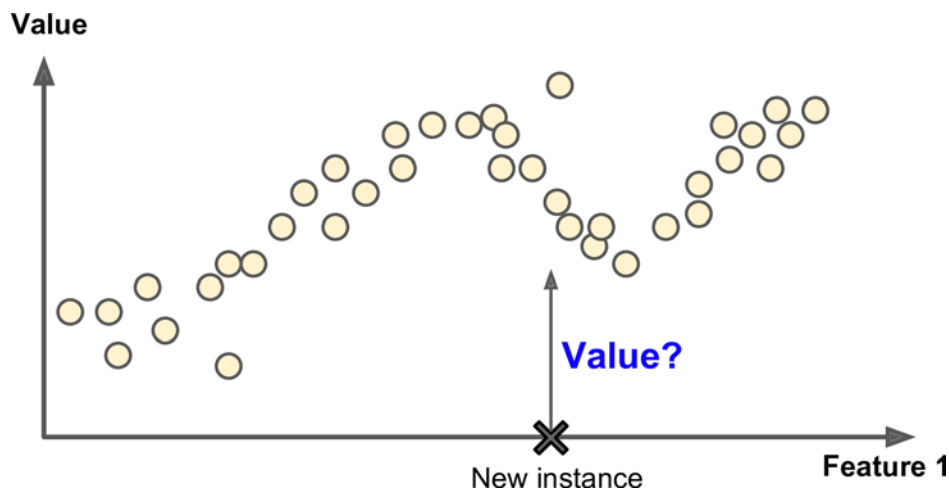
- 예) 스팸필터 작업 (많은 양의 메일, 스팸 유무)

- ◆ 가능한 작업2: 타겟 수치 예측

- 예) 중고차 가격
- 예측 변수(predictor variable)라 부르는 특성(feature)를 사용해 타겟 예측
- **Feature** : 주행거리, 연식, 브랜드, 사고유무 등
- Feature와 Label(중고차 가격)이 포함된 많은 데이터가 필요
- 이러한 작업을 회귀(regression)이라 함

지도 학습 알고리즘

◆ 회귀 (Regression)



◆ 예) 로지스틱 회귀 : 분류에 많이 사용됨. 클래스에 속할 확률 출력 (스팸일 가능성 20%)

◆ 대표적 지도 학습 알고리즘

- K-Nearest Neighbors (k-최근접 이웃)
- Linear Regression (선형 회귀)
- Logistic Regression (로지스틱 회귀)
- Support Vector Machines (SVM, 서포트 벡터 머신)
- Decision Tree(결정 트리)와 Random Forests (랜덤 포레스트)
- Neural networks (신경망) : 일부 신경망 구조는 비지도학습

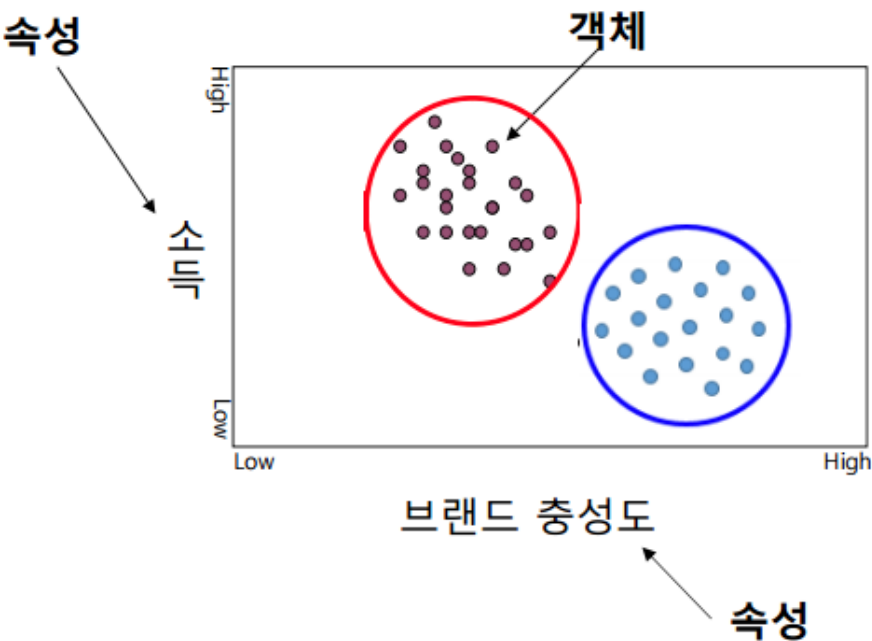
비지도 학습

- ◆ 훈련 데이터에 레이블이 없는 것
- ◆ 최종적으로 내야하는 답이 정해져 있지 않는 것이 특징
- ◆ 대표적 비지도 학습 알고리즘
 - **Clustering (군집)**
 - k-Means (k-평균)
 - Hierarchical Cluster Analysis (HCA, 계층 군집 분석)
 - Expectation Maximization (기댓값 최대화)
 - **Visualization(시각화)와 Dimensionality Reduction (차원 축소)**
 - Principal Component Analysis (PCA, 주성분(최적치) 분석)
 - Kernel PCA
 - Locally-Linear Embedding (LLE, 지역적 선형 임베딩)
 - t-distributed Stochastic Neighbor Embedding (t-SNE, t개의 분산 기반 확률적 근접 위상 배치법)
 - **Association Rule Learning (연관 규칙 학습)**
 - Apriori (어프라이어리) : breadth-first search strategy 사용
 - Eclat (이클렛) : depth-first search algorithm

비지도 학습

◆ 군집 (Clustering) :

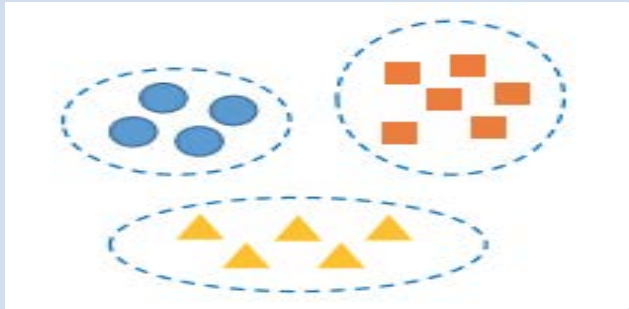
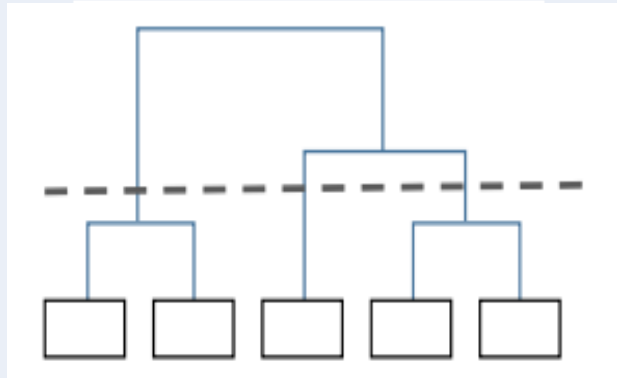
- 유사한 속성의 객체들을 군집(cluster)으로 나누거나 묶어주는 데이터마이닝 기법
- 예) 고객들의 구매 패턴을 반영하는 속성들에 관한 데이터 수집



군집 분석을 통해 유사한 구매패턴을 보이는
고객들을 군집화하고 판매전략을 도출

비지도 학습

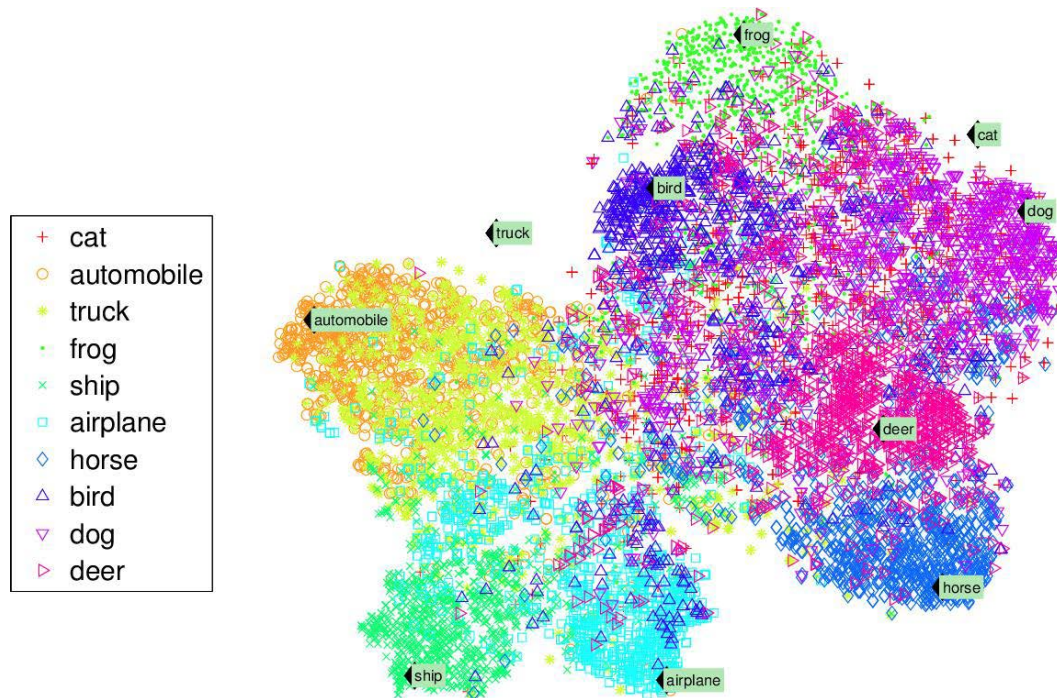
- ◆ 군집 분석의 방법은 ‘계층적 방법’ 과 ‘비계층적 방법’ 으로 구분

종류	내용	도식화
비계층적 군집 (Non-hierarchical Clustering)	사전에 군집 수 k 를 정한 후 입력 데이터를 k 개 중 하나의 군집에 배정	
계층적 군집 (Hierarchical Clustering)	사전에 군집 수 k 를 정하지 않고 단계적으로 군집 트리를 제공	

비지도 학습

◆ 시각화와 차원 축소 (Visualization and Dimensionality Reduction)

- 레이블이 없는 대규모의 고차원 데이터를 2D나 3D로 표현함
- 데이터가 어떻게 조직되어 있는지 이해할 수 있고 예상치 못한 패턴 발견 가능



- 차원 축소 (Dimensionality Reduction)는 너무 많은 정보를 잃지 않으면서 데이터를 간소화하며, 하나의 특성으로 합침
→ 특성 추출 (feature extraction)

비지도 학습

◆ 이상치 탐지 (Anomaly detection)

- 시스템은 정상 샘플로 훈련, 새로운 샘플이 정상 데이터인지 혹은 이상치인지 판단
- 예) 제조 결함 잡아내기, 학습 알고리즘 전 데이터셋에서 이상한 값 자동 제거



◆ 연관 규칙 학습 (Association rule learning)

- 동시 발생 규칙을 이용해 특성 간 관계 탐구로 데이터 패턴 분석
- 연관 규칙 분석 : 군집 분석 이후에 각 그룹의 특성을 분석하기 위해 사용
- 예) 바비큐 소스와 감자를 구매한 사람 → 스테이크 구매 경향 있음

준지도 학습

◆ 레이블이 일부만 있는 데이터

- 대부분 레이블이 없는 데이터가 많고, 레이블이 있는 데이터는 아주 조금

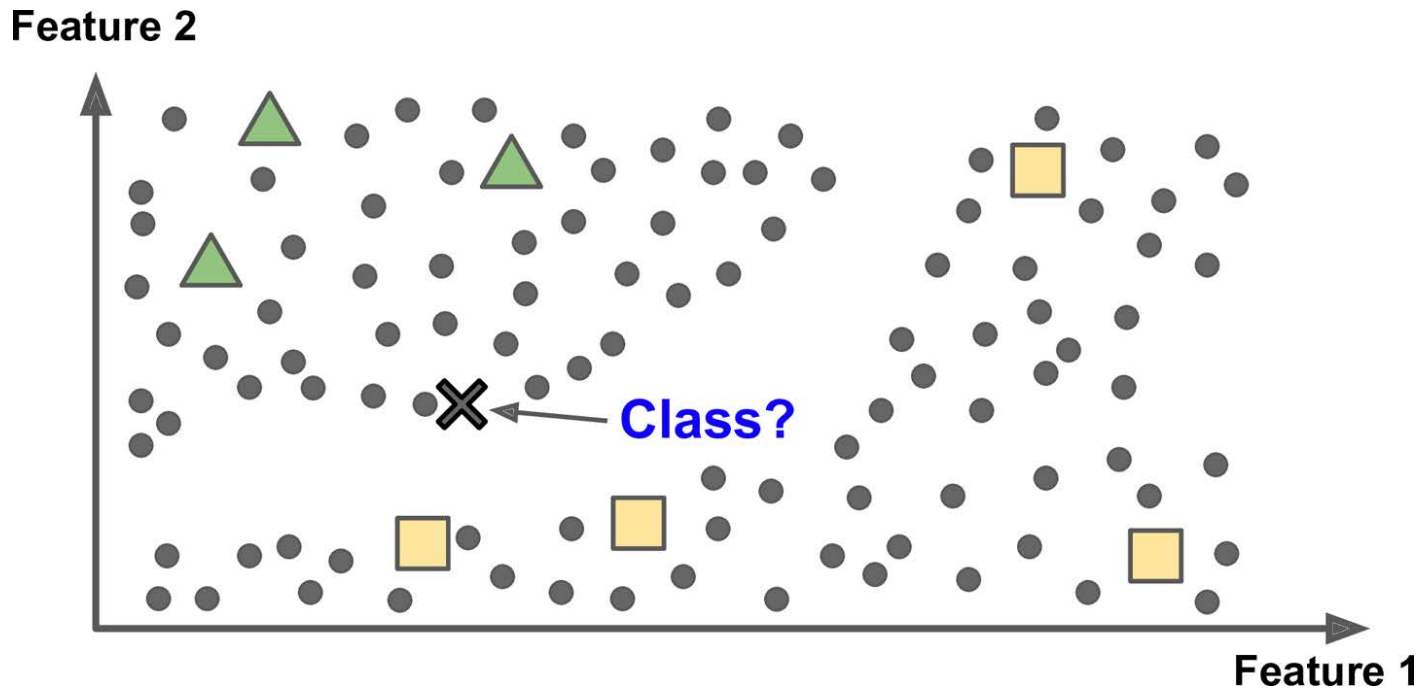
◆ 예) 구글 포토 호스팅 서비스

- 가족사진 업로드 서비스
- 사람 A : 사진 1,5,11 / 사람 B : 사진 2,5,7에 있음 자동 인식 : 비지도학습(군집)
- 문제: 이 사람들이 누구인가?
- 사람마다 레이블이 하나씩만 주어지면 사진에 있는 모든 사람의 이름을 알 수 있음

준지도 학습

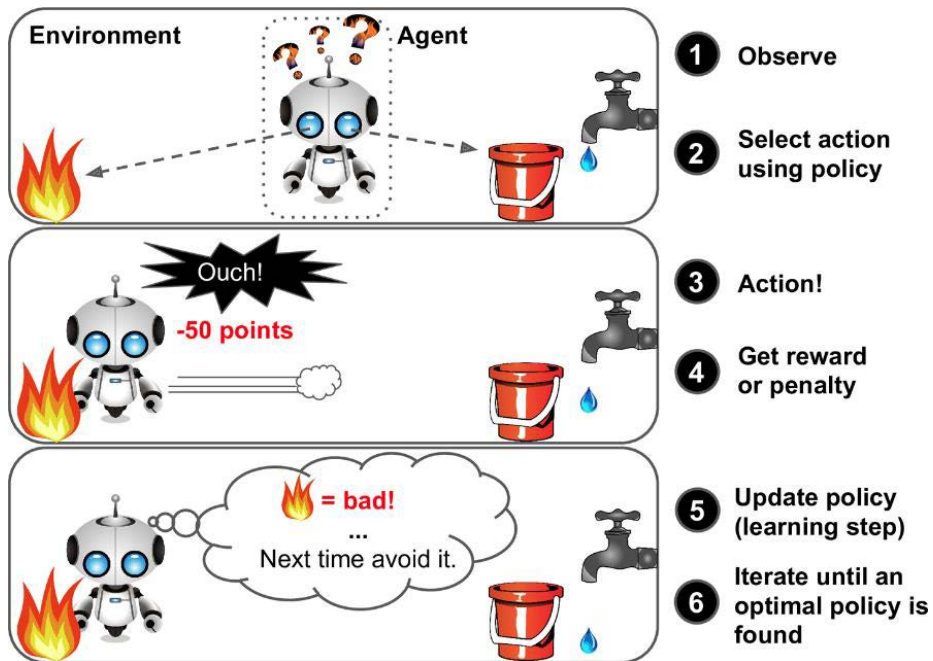
◆ 대부분의 준지도 학습 : 지도 학습 + 비지도 학습의 조합

- 비지도 학습 방식으로 순차적으로 훈련된 다음, 전체 시스템이 지도 학습 방식으로 세밀하게 조정됨



강화 학습

- ◆ 현재 상태를 관찰해서 어떻게 대응해야 할지와 관련된 문제를 다룬다
- ◆ 행동의 주체, 환경(상황 또는 상태), 보상/벌점 등으로 구성
 - 학습하는 시스템: 에이전트
 - 환경을 관찰해서, 행동을 실행하고, 그 결과로 **보상(reward)** 또는 **벌점(penalty)**을 받음
 - 큰 보상을 얻기 위한 최상의 전략(policy) : 시간이 지나면서 스스로 학습

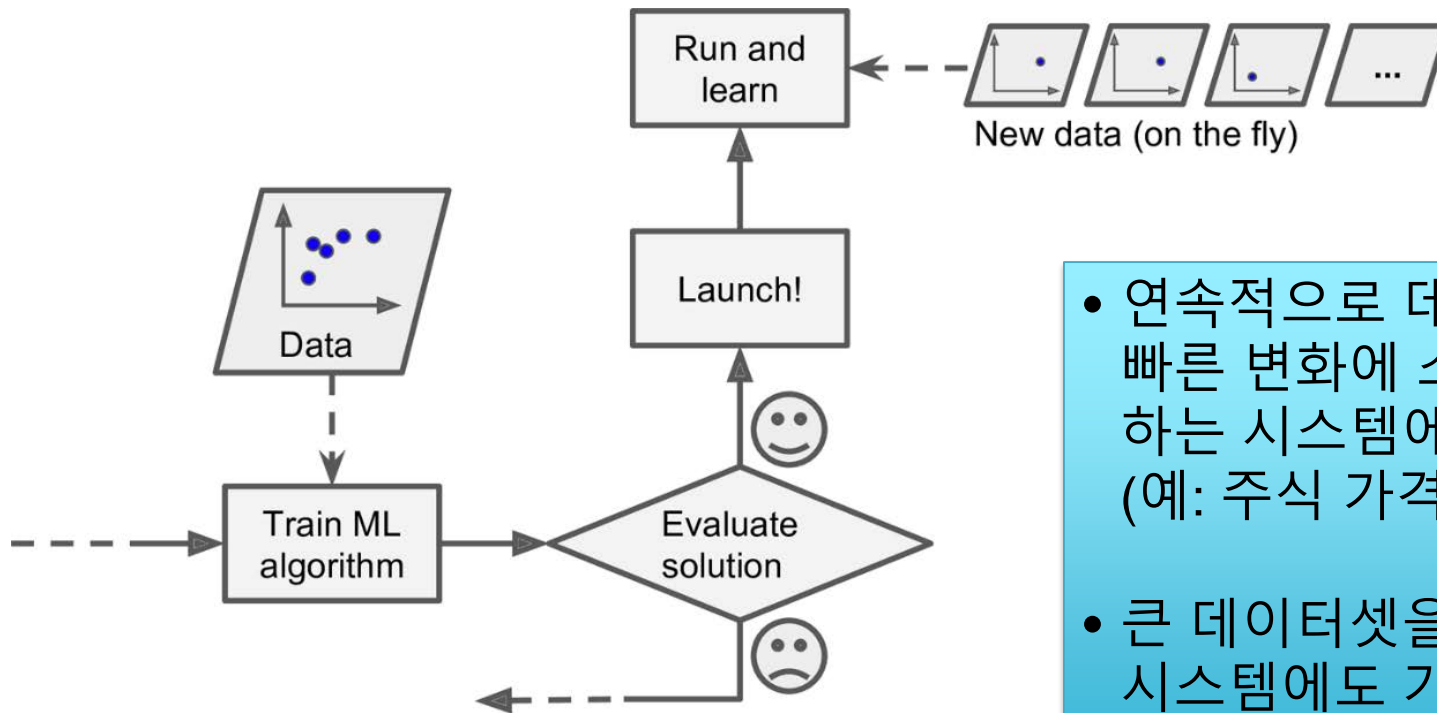


1.3.2 배치 학습과 온라인 학습

- ◆ 머신러닝 시스템을 분류하는데 사용하는 또 다른 기준
- ◆ 입력 데이터의 스트림으로부터 점진적으로 학습 가능 여부
- ◆ 배치학습 (batch learning)
 - 가용한 데이터를 모두 사용하여 훈련 → 시간과 자원 많이 소모
 - 먼저 시스템을 훈련시키고, 제품에 적용하여 더 이상의 학습 없이 실행됨
 - 즉, 학습한 것을 단지 적용만 함 : 오프라인 학습 (offline learning)
- 머신러닝 시스템 훈련, 평가, 론칭 과정 자동화
- 주기적으로 데이터 업데이트 후 시스템 새버전 훈련
- 문제점
 - 새로운 데이터를 학습하려면 전체 데이터를 사용하여 처음부터 다시 훈련해야 함
 - 전체 데이터셋 사용 훈련에 많은 컴퓨팅 자원 필요 (CPU, 메모리 공간 등)
 - 자원이 제한된 시스템(스마트폰, 로봇 등)에서 많은 양의 훈련 데이터를 이동, 학습을 위해 몇 시간씩 많은 자원 사용은 심각한 문제 일으킴

온라인 학습

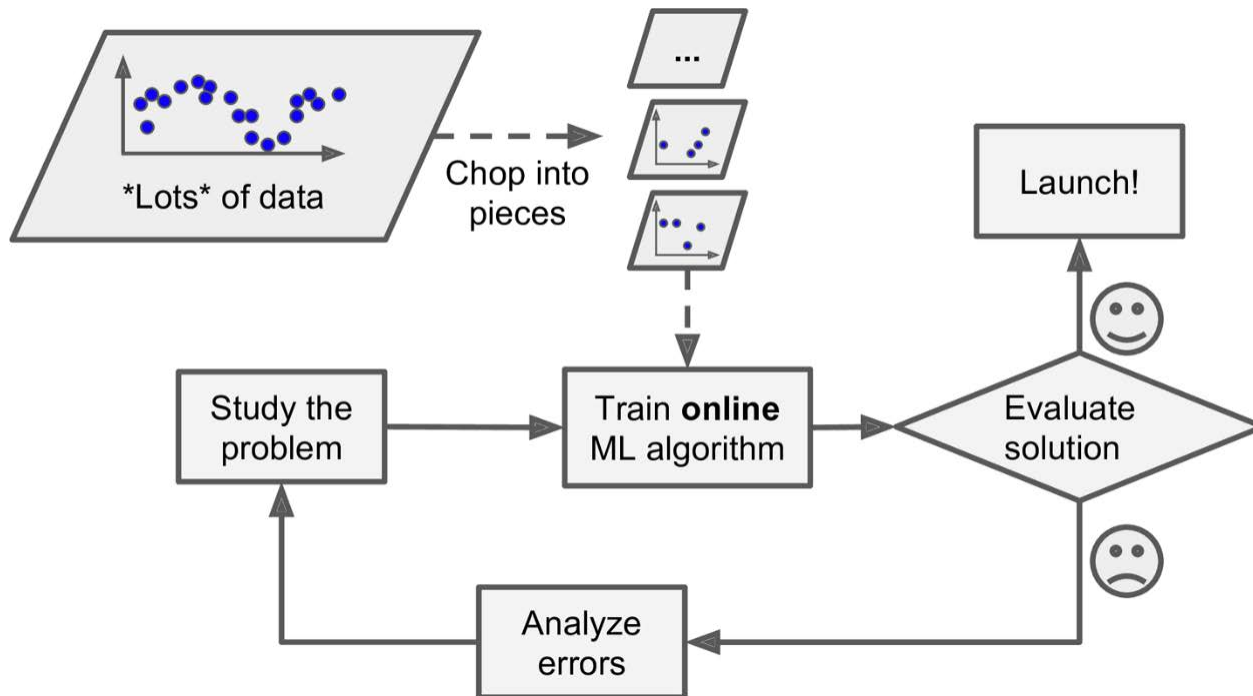
- ◆ 데이터를 순차적으로 한개씩 또는 **미니배치(mini-batch)**라 부르는 작은 묶음 단위로 주입하여 시스템 훈련
- ◆ 비용이 적게 들어 시스템은 데이터가 도착하는 대로 즉시 학습 가능



- 연속적으로 데이터를 받고 빠른 변화에 스스로 적응해야 하는 시스템에 적합 (예: 주식 가격)
- 큰 데이터셋을 학습하는 시스템에도 가능 : 외부 메모리 (out-of-core) 학습

온라인 학습

- ◆ **학습률 (learning rate)** : 변화하는 데이터에 얼마나 빠르게 적응할 것인가
- ◆ 학습률을 높게 하면 데이터에 빠르게 적응하지만, 예전 데이터를 금방 잊음
- ◆ 학습률이 낮으면 시스템의 관성으로 더 느리게 학습, 데이터 잡음에 덜 민감



문제점: 나쁜
데이터가 주입되면
시스템 성능이
점진적으로 감소

1.3.3 사례 기반 학습과 모델 기반 학습

- ◆ 머신러닝 시스템이 어떻게 일반화 되는가에 따라 분류
- ◆ 대부분의 머신러닝 작업: 예측을 만드는 것
- ◆ 주어진 훈련 데이터로 학습하지만, 훈련데이터에서는 본 적 없는 새로운 데이터로 일반화되어야 함
- ◆ 훈련 데이터에서 높은 성능을 내는 것이 좋지만 그게 최종 목표는 아님
- ◆ 진짜 목표는 새로운 샘플에 잘 작동하는 모델
- ◆ 일반화를 위한 두 가지 접근법 :
 - 사례 기반 학습 / 모델 기반 학습

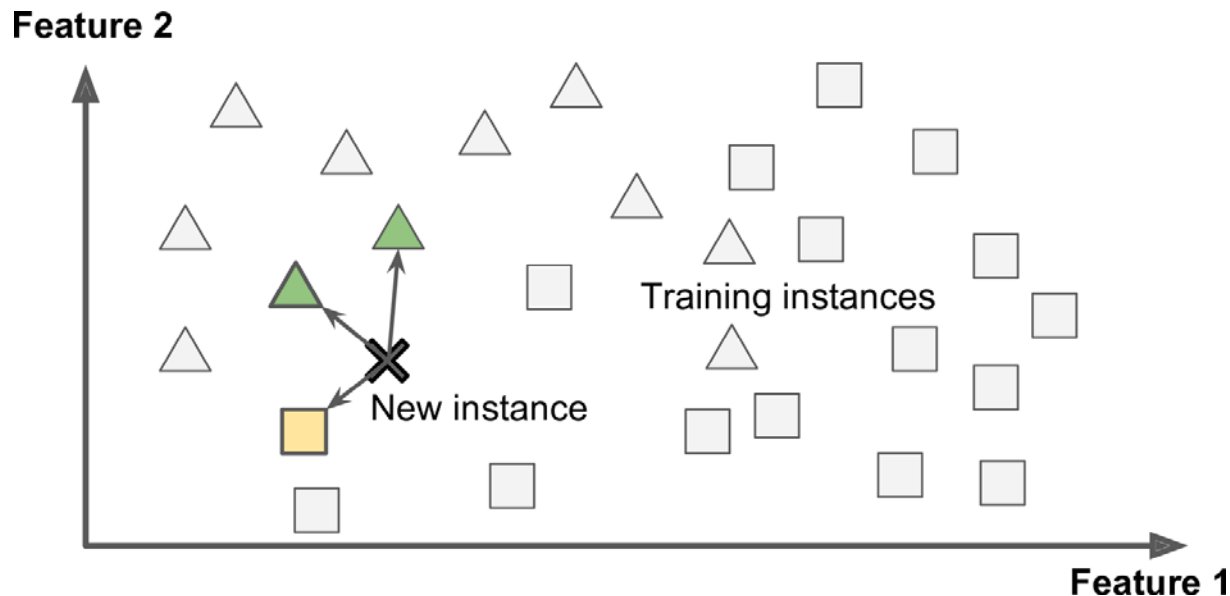
사례 기반 학습

◆ 유사도(similarity) 측정

- 스팸 메일과 동일한 메일을 스팸이라고 지정하는 대신 스팸 메일과 매우 유사한 메일을 구분하도록 스팸 필터 프로그램 개발
- 예) 공통으로 포함한 단어의 수를 세는 것
스팸 메일과 공통으로 가지고 있는 단어가 많으면 스팸으로 분류

◆ 사례 기반 학습 (instance-based learning)

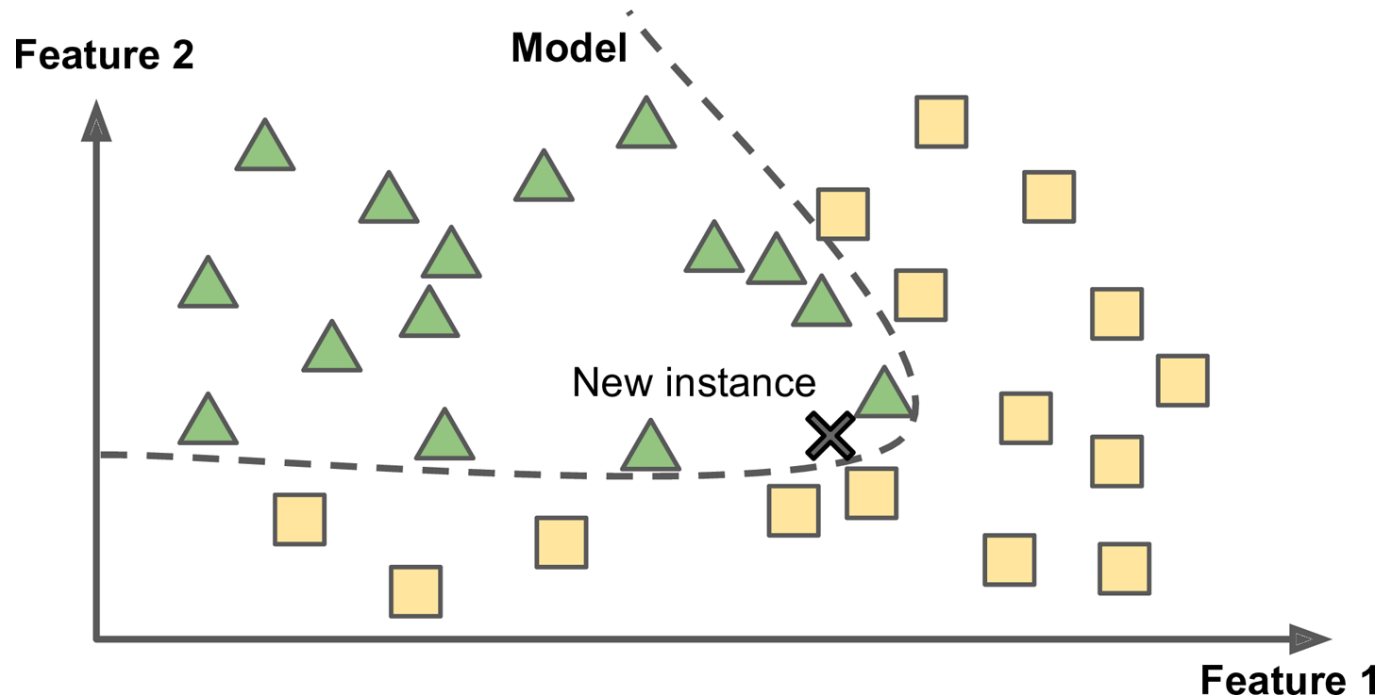
- 시스템이 사례를 기억함으로써 학습
유사도 측정을 통해 새로운 샘플을 일반화



모델 기반 학습

◆ 모델 기반 학습 (model-based learning)

- 샘플들의 모델을 만들어 예측에 사용

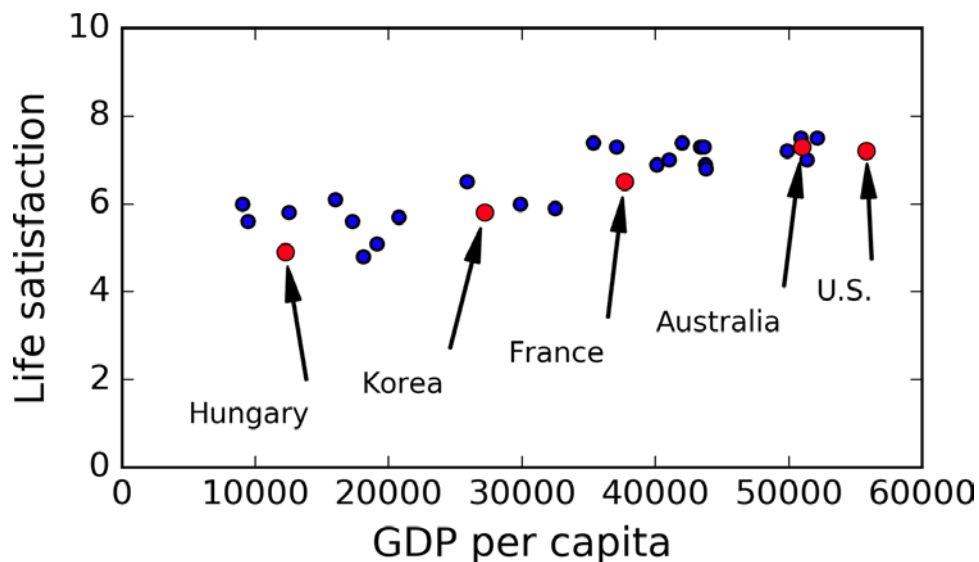


모델 기반 학습

◆ 예) 1인당 GDP에 대한 삶의 만족도

- 국가별 '1인당 GDP' 와 '삶의 만족도' (표 / 그래프)

Country	GDP per capita (USD)	Life satisfaction
Hungary	12,240	4.9
Korea	27,195	5.8
France	37,675	6.5
Australia	50,962	7.3
United States	55,805	7.2

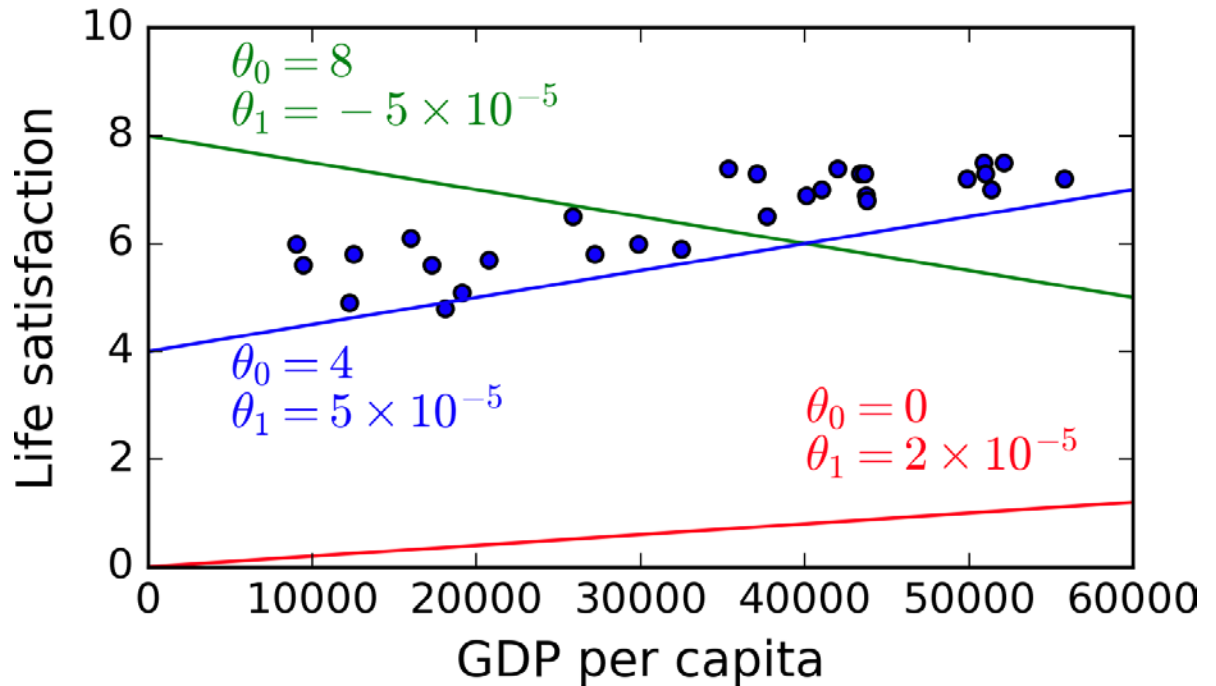


- 1인당 GDP가 증가할수록 선형으로 같이 올라감
- 1인당 GDP의 선형 함수로 **삶의 만족도 선형 모델**을 얻음

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

모델 기반 학습

- ◆ 모델 파라미터 θ_0, θ_1 조정
- ◆ 가능한 몇 개의 선형 모델

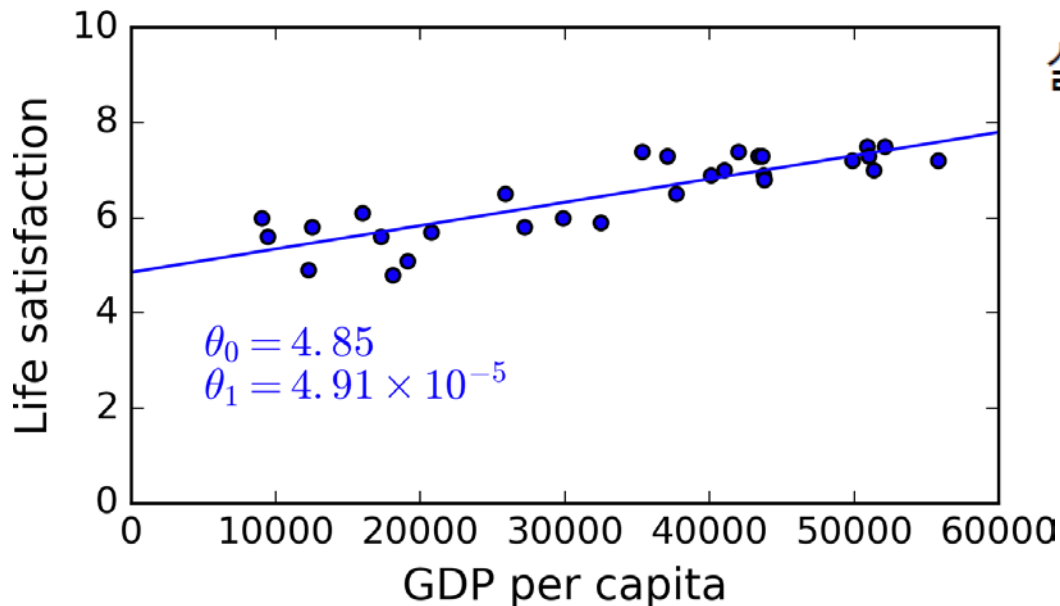


◆ 모델의 최상 성능 측정지표

- 모델이 얼마나 좋은지 측정 : 효용 함수, 적합도 함수
- 모델이 얼마나 나쁜지 측정 : 비용 함수

모델 기반 학습

- ◆ 예) 선형 회귀 → 훈련과 예측 데이터 사이의 거리를 재는 비용함수 최소화가 목표
- ◆ 선형 회귀 알고리즘
 - 알고리즘에 훈련 데이터를 공급하면 데이터에 가장 잘 맞는 선형 모델 파라미터 찾음 : **모델을 훈련(training) 시킨다.**
- ◆ 훈련 데이터에 최적인 선형 모델 찾음



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

예) 국가별 '1인당 GDP'와 '삶의 만족도' 표에 없는 키프로스 국가의 1인당 GDP는 22,587달러
→ 삶의 만족도 계산 : 5.96

머신 러닝 시스템 작업

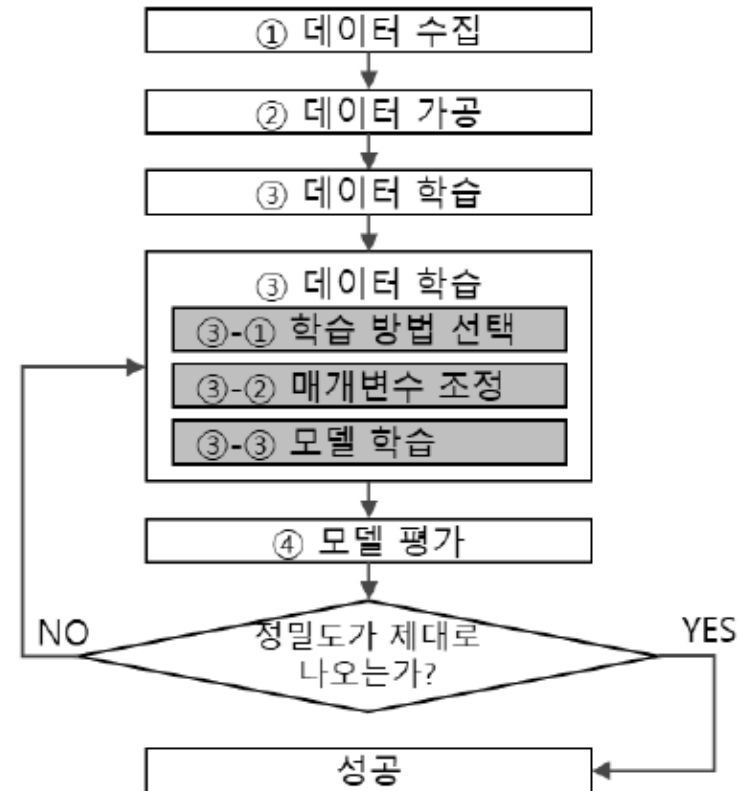
◆ 모든게 다 잘 되면 모델은 좋은 예측 내놓음

◆ 그렇지 않은 경우 추가 처리 필요

- 더 많은 특성 (고용률, 건강, 대기 오염) 을 사용하거나,
- 좋은 훈련 데이터를 더 많이 모으거나,
- 더 강력한 모델(예: 다항 회귀 모델)을 선택해야 할지도...

◆ 머신러닝 시스템 작업 요약

1. 데이터를 분석
2. 모델을 선택
3. 훈련데이터로 모델을 훈련시킴
(비용함수가 최소인 모델 파라미터를 찾음)
4. 새로운 데이터에 모델 적용하여 예측 수행
모델의 일반화 기대...



1.4 머신러닝의 주요 도전 과제

◆ 우리의 주요 작업

- 학습 알고리즘을 선택해서, 어떤 데이터를 훈련시키는 것
- 문제될 수 있는 두 가지 : 나쁜 알고리즘, 나쁜 데이터

◆ 나쁜 데이터 사례

- 충분하지 않은 양의 훈련 데이터
- 대표성 없는 훈련 데이터
- 낮은 품질의 데이터
- 관련 없는 특성

◆ 나쁜 알고리즘 사례

- 훈련 데이터 과대적합
- 훈련 데이터 과소적합

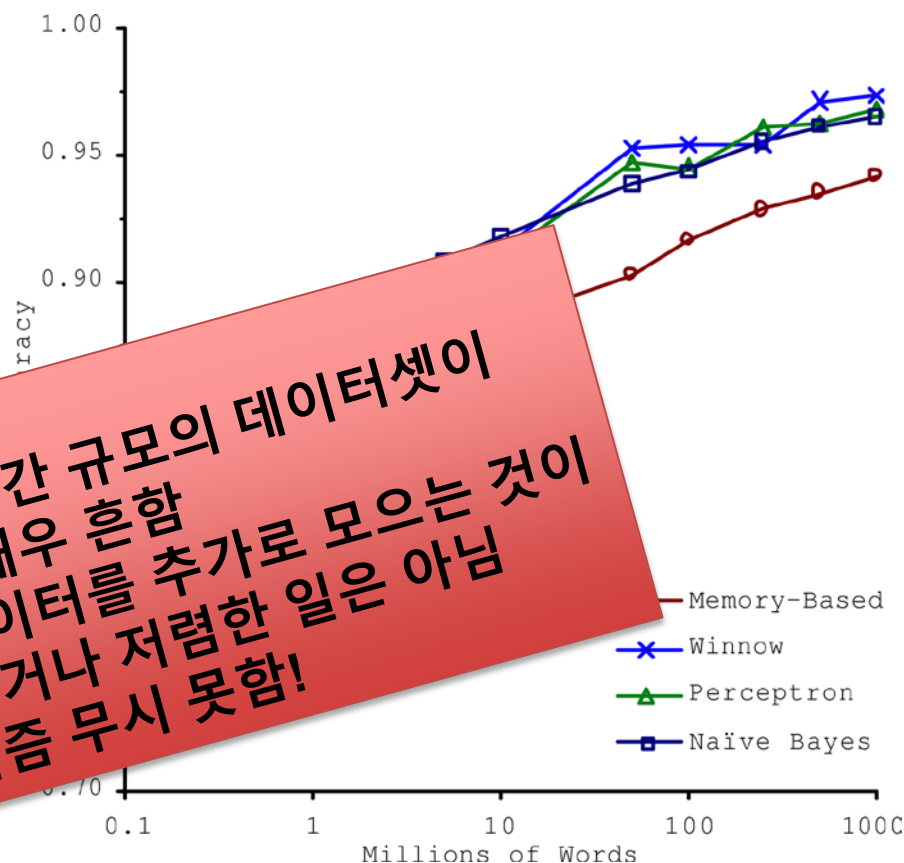
1.4.1 충분하지 않은 양의 훈련 데이터

- ◆ 어린 아이는 ‘사과’에 대해 알려주면 ‘모든 종류의 사과’를 쉽게 일반화 함
- ◆ 예) 충분한 데이터가 주어지면 여러 다른 머신러닝 알고리즘과 관계없이 거의 비슷하게 잘 처리함

시간과 돈을
알고리즘 개발에 쓰는 것
vs.
말뭉치 개발에 쓰는 것
trade-off 관계

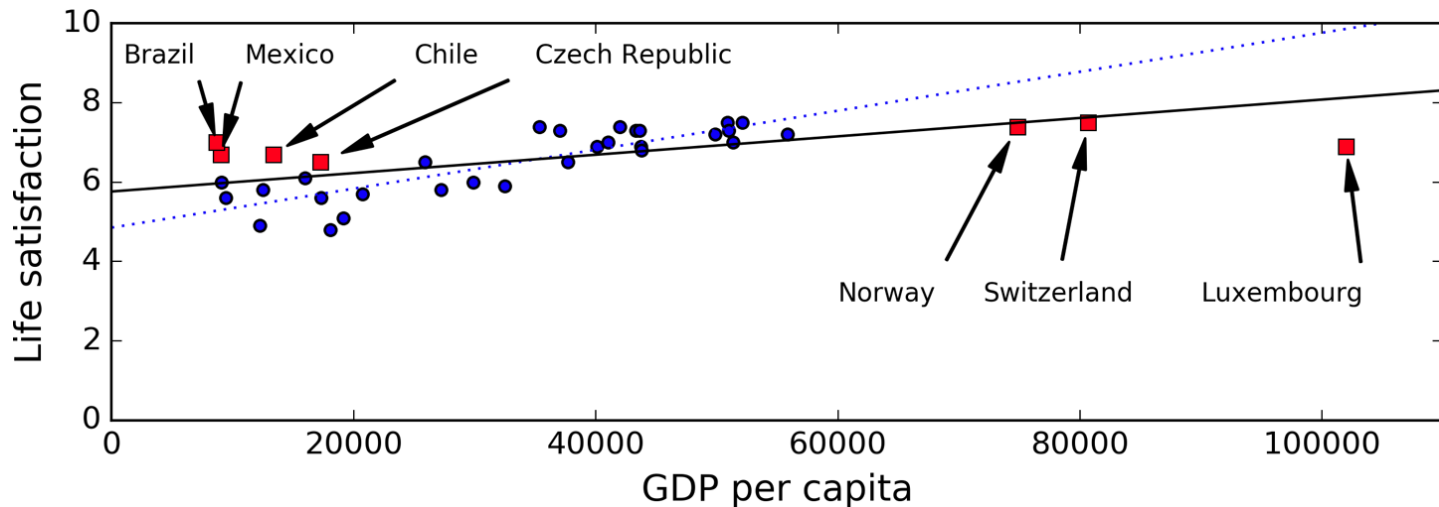
[Michele Banko and Eric
Brill, “Scaling to very very
large corpora for natural language
disambiguation”, 2001]

하지만,
✓ 작거나 중간 규모의 데이터셋이
여전히 매우 흔함
✓ 훈련 데이터를 추가로 모으는 것이
항상 쉽거나 저렴한 일은 아님
✓ 알고리즘 무시 못함!



1.4.2 대표성 없는 훈련 데이터

- ◆ 일반화가 잘되려면, 원하는 새로운 사례를 훈련 데이터가 잘 대표하는 것이 중요
- ◆ 선형 모델 훈련 예) 사용한 나라의 집합에 일부 나라 빠져 있어 대표성이 완벽하지 못함
 - 누락된 나라 추가했을 때 (대표성이 더 큰 훈련 샘플)



- ◆ 누락된 나라 추가 전 모델 : 점선 → 추가 후 모델 : 실선
 - 간단한 선형 모델은 잘 동작하지 않음!
- ◆ 샘플링 잡음 (noise) : 샘플이 작거나 대표성 없는 데이터
- ◆ 샘플링 편향 (bias) : 표본 추출 방법이 잘못된 경우 → 대표성 없음

1.4.3 낮은 품질의 데이터

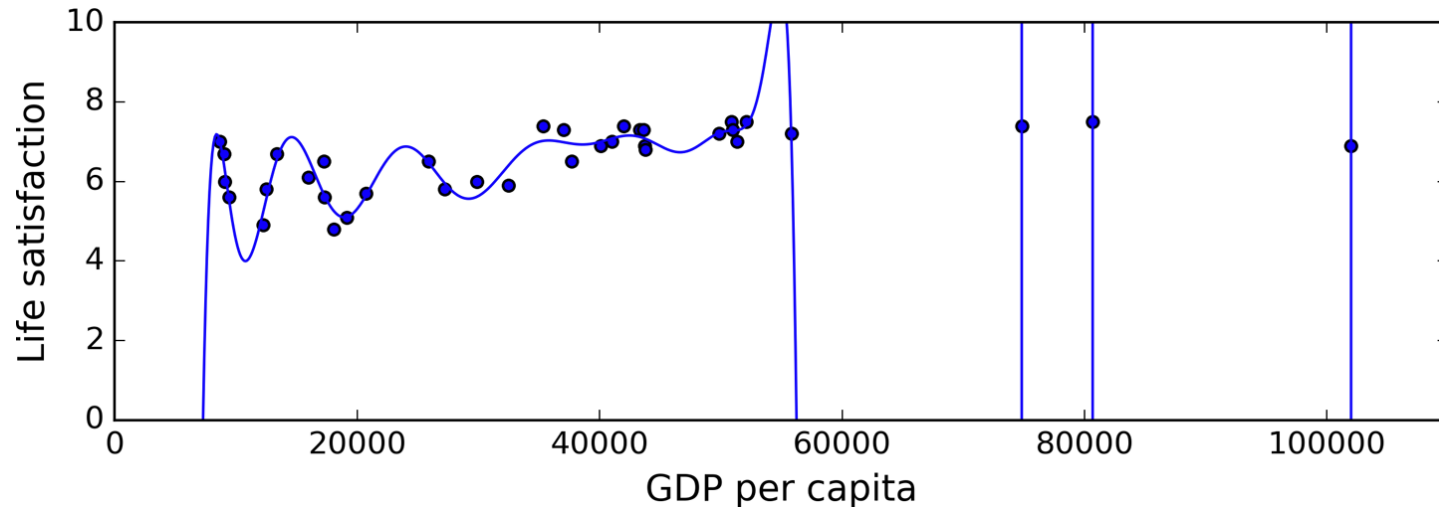
- ◆ 훈련 데이터가 에러, 이상치, 잡음으로 가득한 경우
 - ◆ 패턴을 찾기 어려워 잘 동작 안함
 - ◆ → 훈련 데이터 정제 필요
-
- ◆ 사실, 대부분의 데이터 과학자가 데이터 정제에 많은 시간 할애
 - ◆ 이상치 샘플 무시하거나 수동으로 고침
 - ◆ 일부 샘플에만 특성 몇개가 누락될 시,
특성 모두 무시 / 샘플 무시 / 빠진 값 채움

1.4.4 관련 없는 특성

- ◆ 훈련에 사용할 좋은 특성을 찾는 것 : 특성 공학
- ◆ 성공적인 머신러닝 프로젝트의 핵심 요소
- ◆ **특성 선택 (feature selection)** : 가지고 있는 특성 중에서 훈련에 가장 유용한 특성 선택
- ◆ **특성 추출 (feature extraction)** : 특성을 결합하여 더 유용한 특성을 만듦 (예: 차원 축소 알고리즘 사용)
- ◆ 새로운 데이터를 수집해 새 특성 만듦

1.4.5 훈련 데이터 과대적합

- ◆ 해외여행 중 택시 운전사가 내 물건을 훔쳤다고 가정.
그 나라 모든 택시 운전사는 도둑이라고 생각
- ◆ 일반화의 오류. 머신러닝에서는 과대적합(overfitting)이라고 함
- ◆ 모델이 훈련데이터에만 너무 잘 맞는 경우



- ◆ 훈련 세트에 잡음이 많거나 데이터셋이 너무 작으면 (샘플링 잡음이 발생하므로) 잡음이 섞인 패턴을 감지하게 됨.
- ◆ 이런 패턴은 새로운 샘플에 일반화되지 못함

훈련 데이터 과대적합

◆ 고차원의 다항회귀모델의 경우

◆ 예) 삶의 만족도 모델 : ‘나라 이름’ 특성 추가

- ‘w’가 들어간 나라들의 삶의 만족도는 ‘7’보다 크다는 패턴 감지
- 신뢰할 수 없음 (우연히 훈련 데이터에서 찾은 것)
- 이 패턴이 진짜인지, 잡음 데이터로 인한 것인지 모델이 구분해낼 방법 없음

◆ 과대적합 ← 훈련 데이터에 있는 잡음의 양에 비해 모델이 너무 복잡할 때 자주 발생

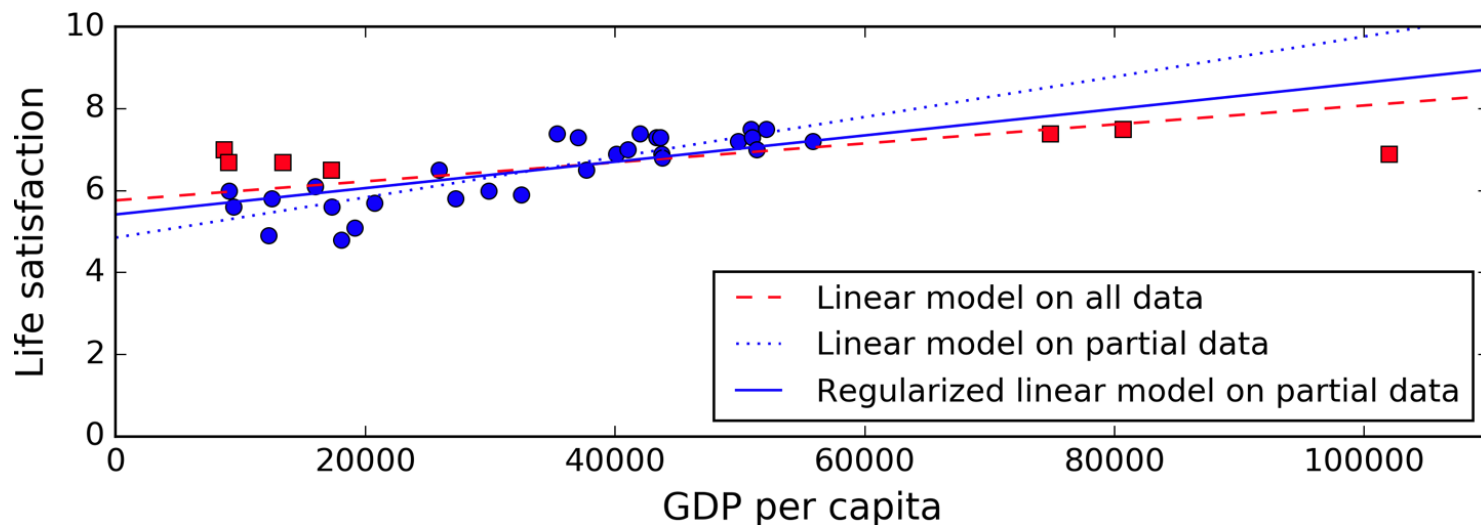
◆ 해결방법

- 파라미터 수가 적은 모델 선택 (고차원 다항 모델보다는 선형 모델)
- 훈련 데이터 특성 수를 줄이거나, 모델에 제약을 가해 단순화시킴
- 훈련 데이터를 더 많이 모음
- 훈련 데이터의 잡음을 줄임 (예: 오류 데이터 수정과 이상치 제거)

훈련 데이터 과대적합

◆ 규제 (regularization)

- 모델을 단순하게 하고, 과대적합의 위험을 감소시키기 위해 모델에 제약을 가하는 것
- 규제로 과대적합될 위험 감소



- 규제가 모델의 기울기를 더 작게 만들어 훈련데이터에는 덜 맞지만, 새로운 샘플에는 더 잘 일반화됨

훈련 데이터 과대적합

◆ 하이퍼파라미터

- (모델이 아니라) 학습 알고리즘의 파라미터
- 훈련 전에 미리 저장되고, 훈련하는 동안에는 상수로 남아 있음
- 매우 큰 값의 규제 하이퍼파라미터 지정 (기울기=0에 가까운)
→ 평편한 모델
- 과대 적합될 가능성 없지만 좋은 모델 찾지 못함
- 머신러닝 시스템 구축 시 하이퍼파라미터 튜닝은 매우 중요한 과정임

1.4.6 훈련 데이터 과소적합

◆ 과대적합의 반대

- ◆ 모델이 너무 단순해서 데이터의 내재된 구조를 학습하지 못할 때 발생
- ◆ 예) 삶의 만족도 선형 모델 : 현실은 이 모델보다 더 복잡. 훈련 샘플에서조차도 부정확한 예측

◆ 해결 방법

- 파라미터가 더 많은 강력한 모델 선택
- 학습 알고리즘에 더 좋은 특성 제공 (특성 공학)
- 모델의 제약을 줄임 (예: 규제 하이퍼파라미터 감소)

1.5 테스트와 검증

- ◆ **모델이 새로운 샘플에 얼마나 잘 일반화될까?**
 - 새로운 샘플에 실제로 적용해 봄
 - 실제 서비스에 모델을 넣고 잘 동작하는지 모니터링
 - 모델이 아주 나쁠 때 고객 불만 토로
- ◆ **훈련 데이터 : 훈련 세트 + 테스트 세트로 나눔**
 - 훈련 세트를 사용해 모델을 훈련시킴 (데이터의 80%)
 - 테스트 세트를 사용해 모델을 테스트함 (나머지 20%)
- ◆ **새로운 샘플에 대한 오류 비율 : 일반화 오차 (외부 샘플 오차)**
 - 테스트 세트에서 모델을 평가하여 오차에 대한 추정값 획득
 - 이전에 본 적이 없는 새로운 샘플에 모델이 얼마나 잘 작동하는가?
 - 훈련 오차가 낮지만 (훈련 세트에서 모델의 오차 적음), 일반화 오차가 높다면 : 훈련 데이터에 과대 적합됨

모델 평가

◆ 모델 평가 : 두 모델 중 선택 갈등 (선형 모델 vs. 다항 모델)

- 두 모델 모두 훈련 세트로 훈련
- 테스트 세트를 사용해 얼마나 잘 일반화되는지 비교
- 선형 모델이 더 잘 일반화되었다고 가정
- 과대적합을 피하기 위해 규제 적용
- 하이퍼파라미터 값 선택 (100개의 하이퍼파라미터 값으로 100개의 다른 모델 훈련)
- 모델을 실제 서비스에 투입
- 성능이 예상보다 좋지 않음 (오차 15% 발생)
- 테스트 세트에 최적화된 모델
- **검증 세트** : 두번째 홀드아웃(holdout) 세트 생성
 - 최상의 성능을 내는 모델과 하이퍼파라미터 선택
- 만족스러운 모델을 찾으면 일반화 오차의 추정값을 얻기 위해 테스트 세트로 단 한번의 최종 테스트 수행
- **교차검증** : 훈련 데이터에서 검증 세트로 너무 많은 양의 데이터를 뺏기지 않기 위해 훈련 세트를 여러 서브셋으로 나누고 각 모델을 서브셋의 조합으로 훈련시키고 나머지 부분으로 검증
- 모델과 하이퍼파라미터 선택되면 전체 훈련 데이터를 사용하여 최종 모델 훈련
- 테스트 세트에서 일반화 오차 측정

한눈에 보는 머신러닝

- ◆ 머신러닝은 명시적인 규칙을 코딩하지 않고, 기계가 데이터로부터 학습하여 어떤 작업을 더 잘하도록 만드는 것
- ◆ 여러 종류의 머신러닝 시스템
 - 지도 학습 / 비지도 학습
 - 배치 학습 / 온라인 학습
 - 사례 기반 학습 / 모델 기반 학습
- ◆ 훈련 세트에 데이터를 모아 학습 알고리즘에 주입
 - 학습 알고리즘이 모델 기반인 경우 훈련 세트에 모델 맞추기 위해 파라미터 조정
 - 사례 기반인 경우 샘플을 기억하는 것이 학습, 새로운 샘플 일반화 위해 유사도 측정 사용
- ◆ 훈련 세트가 너무 작거나 대표성이 없는 데이터이거나, 잡음이 많고 관련 없는 특성으로 오염되어 있다면 시스템 잘 작동하지 않음
- ◆ 모델이 너무 단순하거나(과소적합), 너무 복잡(과대적합) 하지 않아야 함

머신러닝 프로젝트

가상의 프로젝트 진행

2.2 큰그림 보기

- 2.2.1 문제정의
- 2.2.2 성능 측정 지표 선택
- 2.2.3 가정 검사

2.3 데이터 가져오기

- 2.3.1 작업 환경 만들기
- 2.3.2 데이터 다운로드
- 2.3.3 데이터 구조 훑어보기
- 2.3.4 테스트 세트 만들기

2.4 데이터 탐색 및 시각화

- 2.4.1 지리적 데이터 시각화
- 2.4.2 상관관계 조사
- 2.4.3 특성 조합으로 실험

2.5 데이터 준비

- 2.5.1 데이터 정제
- 2.5.2 텍스트와 범주형 특성 다루기
- 2.5.3 나만의 변환기
- 2.5.4 특성 스케일링
- 2.5.5 변환 파이프라인

2.6 모델 선택 및 훈련

- 2.6.1 훈련 세트에서 훈련하고 평가하기
- 2.6.2 교차 검증을 사용한 평가

2.7 모델 세부 튜닝

- 2.7.1 그리드 탐색
- 2.7.2 랜덤 탐색
- 2.7.3 앙상블 방법
- 2.7.4 최상의 모델과 오차 분석
- 2.7.5 테스트 세트로 시스템 평가하기

◆ 솔루션 제시

◆ 시스템 론칭

◆ 모니터링, 유지보수

2.1 실제 데이터로 작업하기

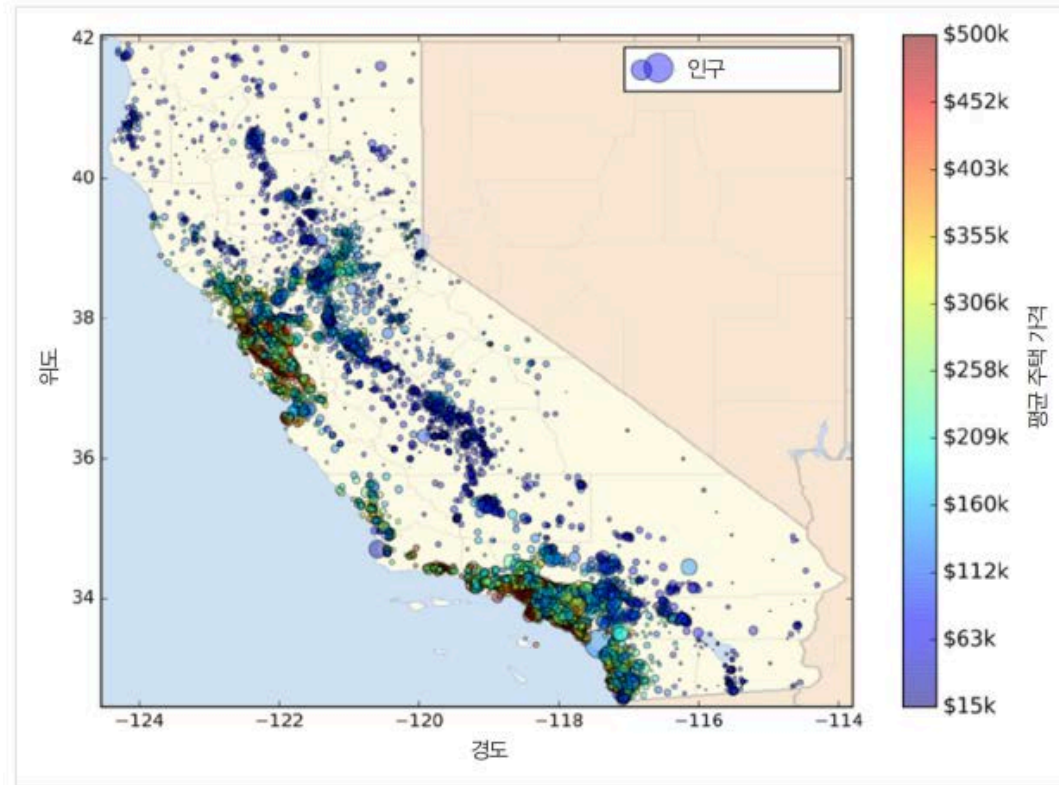
◆ 공개된 데이터셋

- 유명 공개 데이터 저장소
 - UC Irvine Machine Learning Repository (<http://archive.ics.uci.edu/ml/>)
 - Kaggle datasets (<https://www.kaggle.com/datasets>)
 - Amazon's AWS datasets (<http://aws.amazon.com/fr/datasets/>)
- 메타 포털
 - <http://dataportals.org/>
 - <http://opendatamonitor.eu/>
 - <http://quandl.com/>
- 인기 공개 데이터 저장소 리스트
 - Wikipedia's list of Machine Learning datasets (<https://goo.gl/SJHN2k>)
 - Quora.com question (<http://goo.gl/zDR78y>)
 - Datasets subreddit (<https://www.reddit.com/r/datasets>)

2.2 큰그림 보기

- ◆ 프로젝트 목적 (주택회사)
 - 캘리포니아 주택 가격 모델 작성
 - 다른 데이터가 주어졌을 때 구역의 **중간 주택 가격 예측**
- ◆ 캘리포니아 인구조사 데이터 사용하여 모델 작성
 - 캘리포니아 블록 그룹별 (600~3,000명 인구)
 - 인구, 중간 소득, 중간 주택 가격 포함

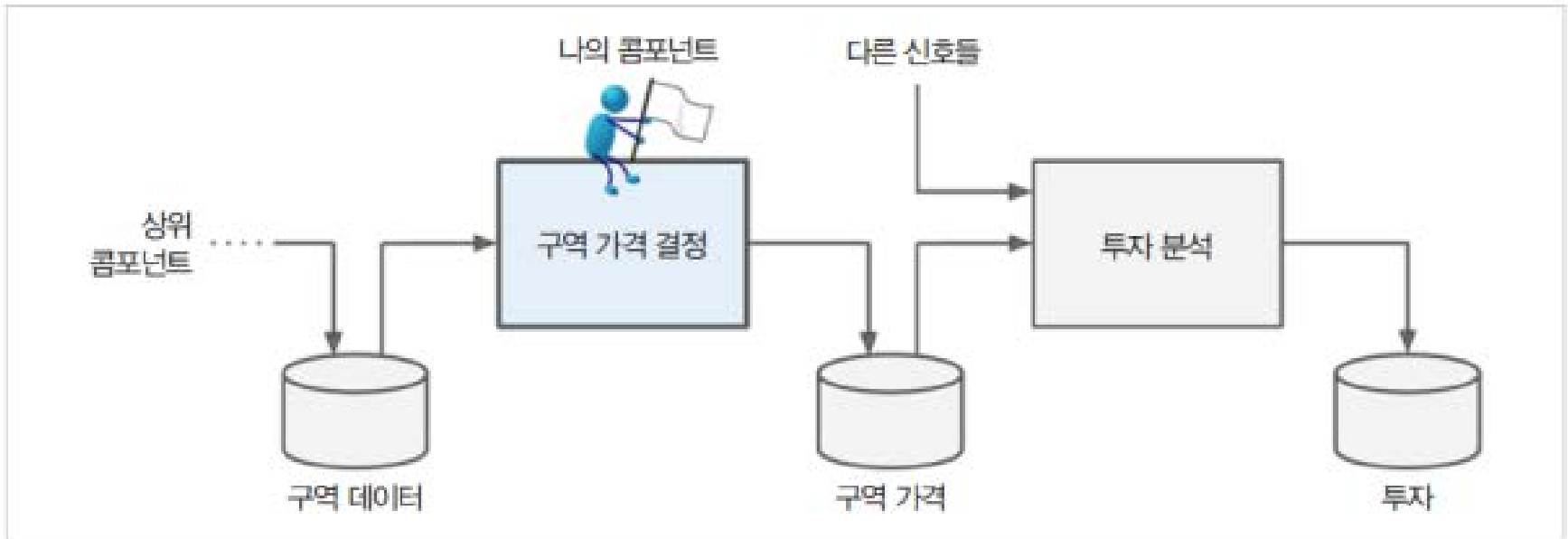
그림 2-1 캘리포니아 주택 가격



2.2.1 문제 정의 - 비즈니스 목적

- ◆ 비즈니스 목적 파악
 - 시스템 구성, 알고리즘, 측정 지표, 튜닝 시간 결정
- ◆ 비즈니스 목적 예) 중간 주택 가격 예측 출력 → 투자 결정 머신러닝 시스템 입력

그림 2-2 부동산 투자를 위한 머신러닝 파이프라인



2.2.1 문제 정의

◆ 현재 솔루션?

- 전문가가 수동으로 추정 (10%이상 오류)

◆ 문제 정의

- 지도/비지도/강화학습 ?
- 분류/회귀학습 ?
- 배치/온라인학습 ?

- 레이블된 샘플 있음 → 지도학습
- 값을 예측 → 회귀학습 (여러 특성 : multivariate regression)
 - 구역의 인구, 중간 소득 특성을 사용
 - 1장 예시: 1인당 GDP 특성 사용 삶의 만족도 예측 (univariate regression)
- 오프라인, 비교적 느리고, 크지 않은 데이터 → 배치학습

2.2.2 성능 측정 지표

◆ 전형적인 회귀문제 성능 지표

- 평균 제곱근 오차 (Root Mean Square Error : RMSE)
 - 오차가 커질 수록 RMSE 값은 커짐 (예측에 많은 오류가 있음)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

- 평균 절대 오차 (Mean Absolute Error) , 평균 절대 편차
 - 이상치로 보이는 경우가 많은 경우 사용

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m |h(\mathbf{x}^{(i)}) - y^{(i)}|$$

- \mathbf{X} : 데이터셋에 있는 모든 샘플의 모든 특성값을 포함하는 행렬
- h : 시스템의 예측 함수 (가설 : hypothesis)
- m : 데이터셋에 있는 샘플 수
- $\mathbf{x}^{(i)}$: 데이터셋에 있는 i 번째 샘플의 전체 특성값 벡터
- $y^{(i)}$: 해당 샘플의 기대 출력값 (레이블)
- $\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$: 샘플 $\mathbf{x}^{(i)}$ 에 대한 예측함수(h)에 의한 예측값

2.2.2 성능 측정 지표

◆ 예측값과 타깃값 사이의 거리를 재는 방법 (norm)

- Euclidian norm : RMSE
- Manhattan norm : MAE
- 원소가 n 개인 벡터 v 의 l_k 노름
 - l_0 : 단순히 벡터에 있는 0이 아닌 원소의 수
 - l_∞ : 벡터에서 가장 큰 절댓값

$$\|v\|_k = (|v_0|^k + |v_1|^k + \dots + |v_n|^k)^{\frac{1}{k}}$$

$$\text{유클리디안 노름} = l_2 \text{ 노름} = \|v\|_2 = \|v\| = \sqrt{m} \times \text{RMSE}$$

$$\text{맨하탄 노름} = l_1 \text{ 노름} = \|v\|_1 = m \times \text{MAE}$$

- k (노름지수)가 클수록 큰 값의 원소에 치우치며 작은 값은 무시
- RMSE가 MAE보다 이상치에 더 민감
- 이상치가 매우 드물면 RMSE가 잘 맞아 일반적으로 널리 사용 (예: 종 모양 분포의 양 끝단)

2.2.2 성능 측정 지표 - 표기법

◆ 어떤 구역의 값

- 경도 -118.29, 위도 33.91
- 주민수 1,416명
- 중간소득 \$38,372

$$\mathbf{x}^{(1)} = \begin{pmatrix} -118.29 \\ 33.91 \\ 1,416 \\ 38,372 \end{pmatrix}$$

$$y^{(1)} = 156,400$$

- (레이블) 중간 주택 가격 \$156,400

◆ 데이터셋의 모든 샘플 값

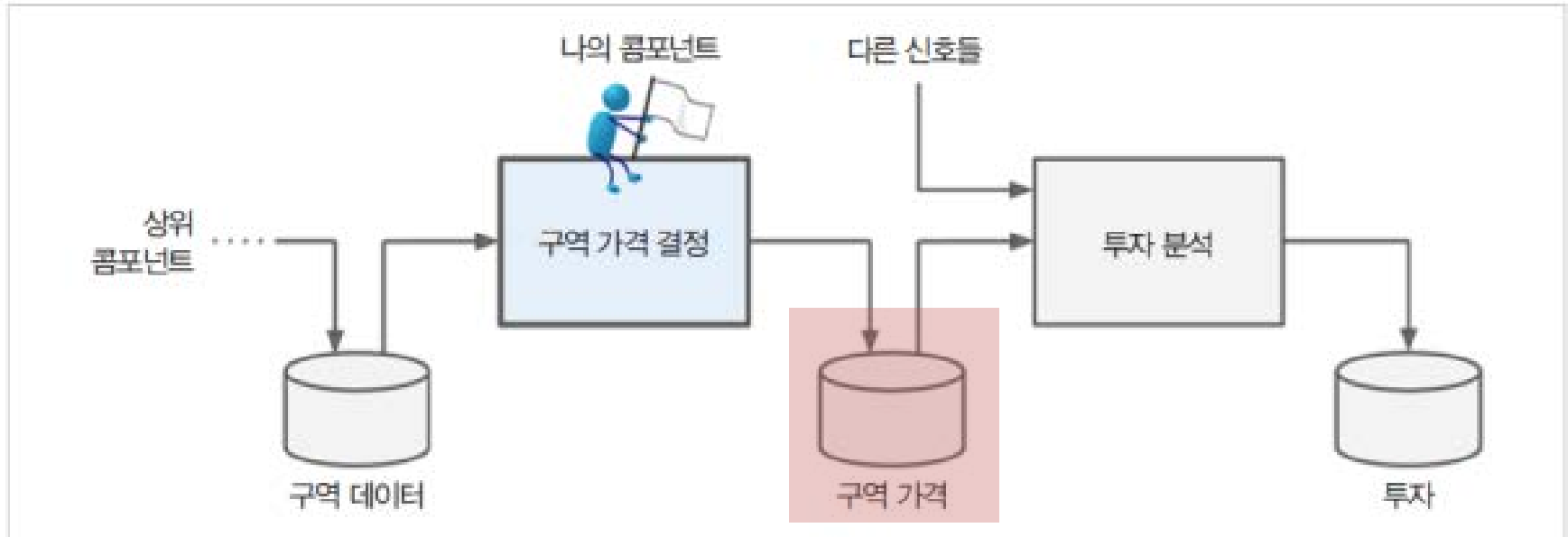
$$\mathbf{X} = \begin{pmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(1999)})^T \\ (\mathbf{x}^{(2000)})^T \end{pmatrix} = \begin{pmatrix} -118.29 & 33.91 & 1,416 & 38,372 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

◆ 하나의 샘플에 대한 예측값

$$\hat{y}^{(i)} = h(\mathbf{x}^{(i)})$$

2.2.3 가정 검사

- ◆ 지금까지 만든 가정을 나열하고 검사



→ 회귀가 아닌, 분류작업이 필요

- ◆ 너무 늦게 문제를 발견하지 않도록 주의...

2.3.1 작업환경 만들기

◆ 필요한 python package

- numpy, pandas, matplotlib
- jupyter notebook
- scikit-learn
- tensorflow
- ...

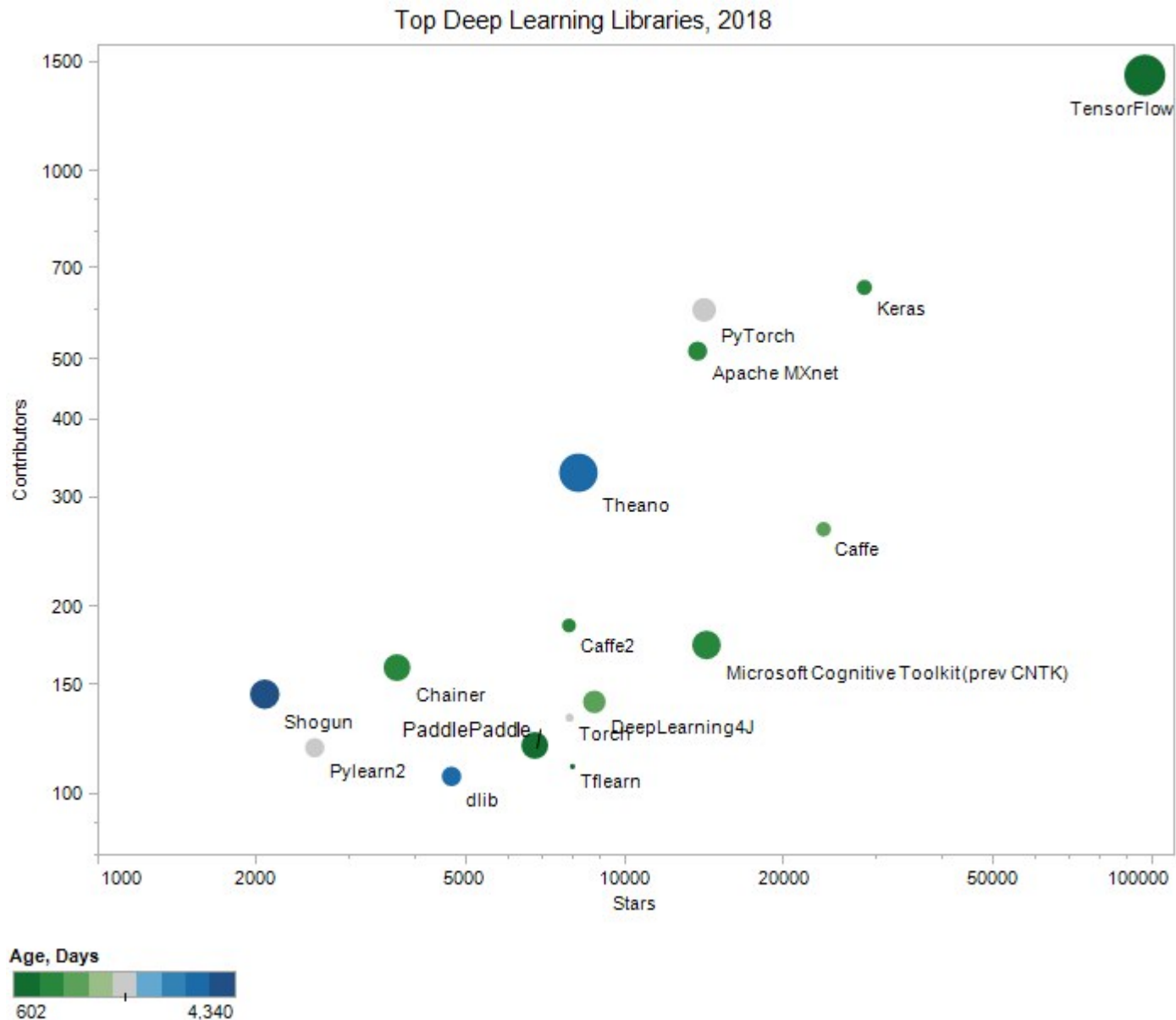
◆ Anaconda package

- 300개 이상 모듈 포함

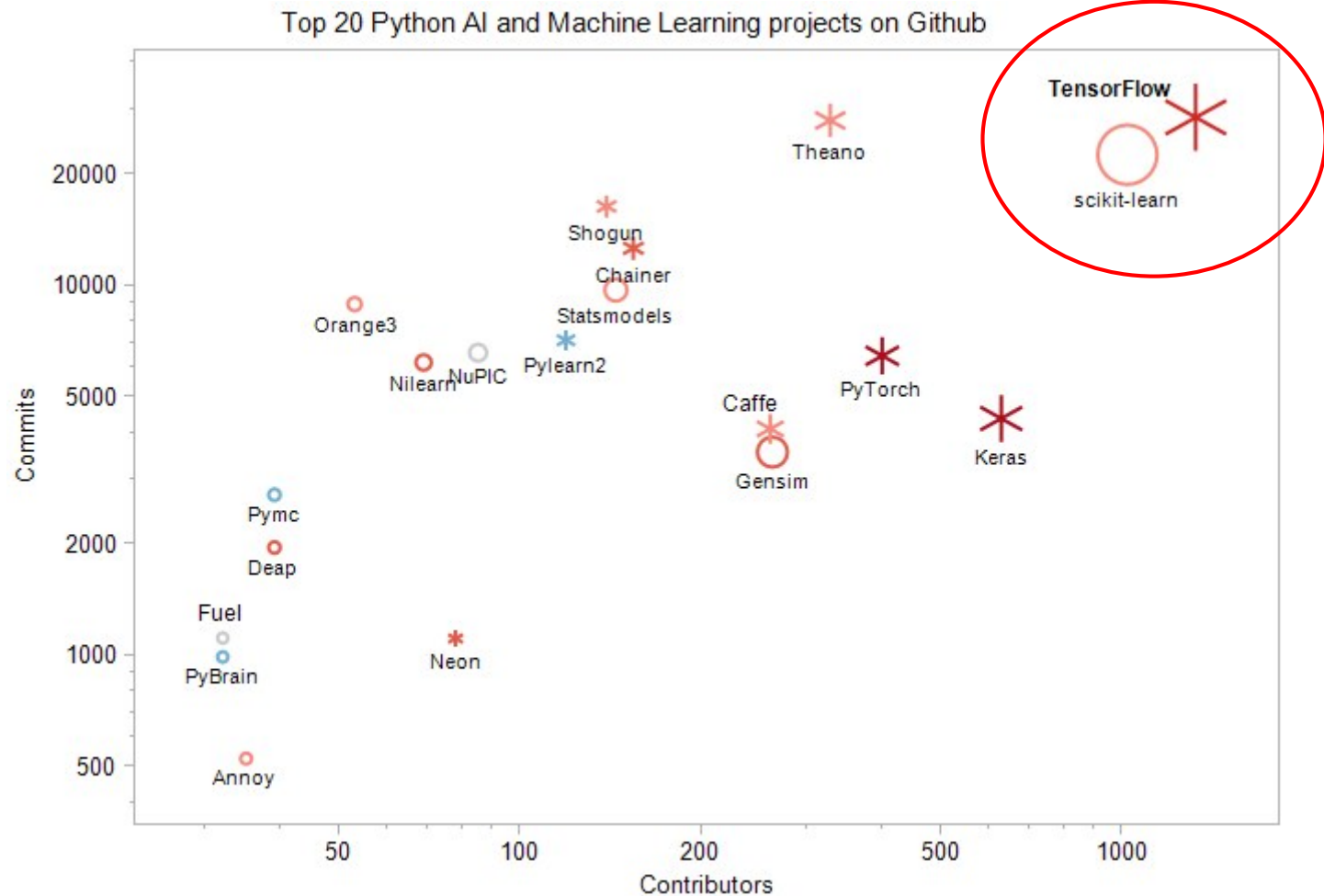
◆ Anaconda 설치 후 업데이트 확인 할 것

- Anaconda Prompt
 - > conda update scikit-learn
 - > conda update pandas

Deep Learning Libraries



Python AI and Machine Learning projects



2.3.2 데이터 다운로드

◆ 파일 다운로드 및 압축해제

```
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

```
fetch_housing_data()
```

pandas로 csv 파일 읽기

◆ 10개의 특성으로 구성

```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

```
housing = load_housing_data()
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

2.3.3 데이터 구조 훑어보기

◆ housing.info() - 데이터 정보 확인

- 총 20,640개 샘플
- total_bedrooms 특성은 20,433개만 널값이 아님 (207개 비어 있음)
- ocean_proximity 필드 데이터 타입
 - object (문자열), 카테고리 (예: NEAR BAY)

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude          20640 non-null float64
latitude           20640 non-null float64
housing_median_age 20640 non-null float64
total_rooms         20640 non-null float64
total_bedrooms     20433 non-null float64
population         20640 non-null float64
households         20640 non-null float64
median_income      20640 non-null float64
median_house_value 20640 non-null float64
ocean_proximity    20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

◆ ocean_proximity 필드 내 카테고리 확인

```
housing["ocean_proximity"].value_counts()
```

```
<1H OCEAN      9136  
INLAND          6551  
NEAR OCEAN      2658  
NEAR BAY        2290  
ISLAND           5  
Name: ocean_proximity, dtype: int64
```

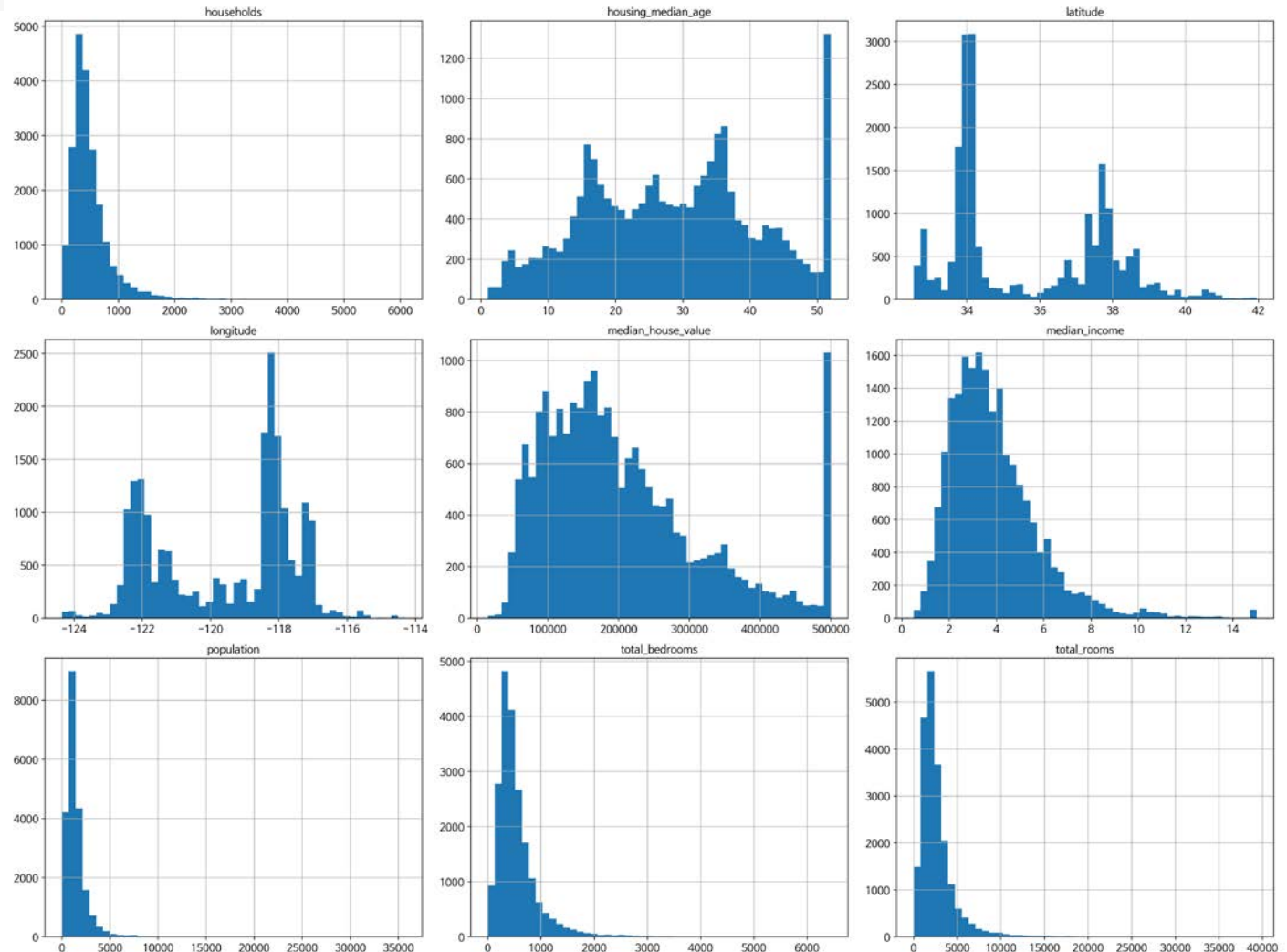
◆ 숫자형 특성 요약 정보

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

matplotlib 히스토그램 그리기

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



2.3.4 테스트 세트 만들기

- ◆ 데이터로부터 테스트 세트 분리
 - 전체 데이터에서 너무 많은 직관을 얻으면 과대적합된 모델이 생성될 가능성이 있음 (data snooping 편향)
 - ◆ 무작위로 샘플을 선택해서 데이터셋의 20% 정도 분리
 - ◆ 프로그램 재실행 시 이전의 데이터셋 불러와야...
 - 이전 세트 저장 후 불러오기
 - 동일한 난수 인덱스 생성하기
 - ◆ 데이터셋 업데이트 후에도 적용 가능해야...
- 고유식별자(예: ID)를 사용하여 테스트 세트로 결정

scikit-learn 테스트 세트 분리 함수

◆ 사이킷런 `train_test_split()` 함수

- 난수 초기값 지정 가능
- 동일 인덱스 기반 분리 가능

여러개의 배열을 넣을 수 있습니다
(파이썬 리스트, 넘파이, 판다스 데이터프레임)

`train_size`도 지정할 수 있습니다

```
from sklearn.model_selection import train_test_split  
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

```
test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	INLAND
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	INLAND
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	NEAR BAY
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	<1H OCEAN
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	NEAR OCEAN

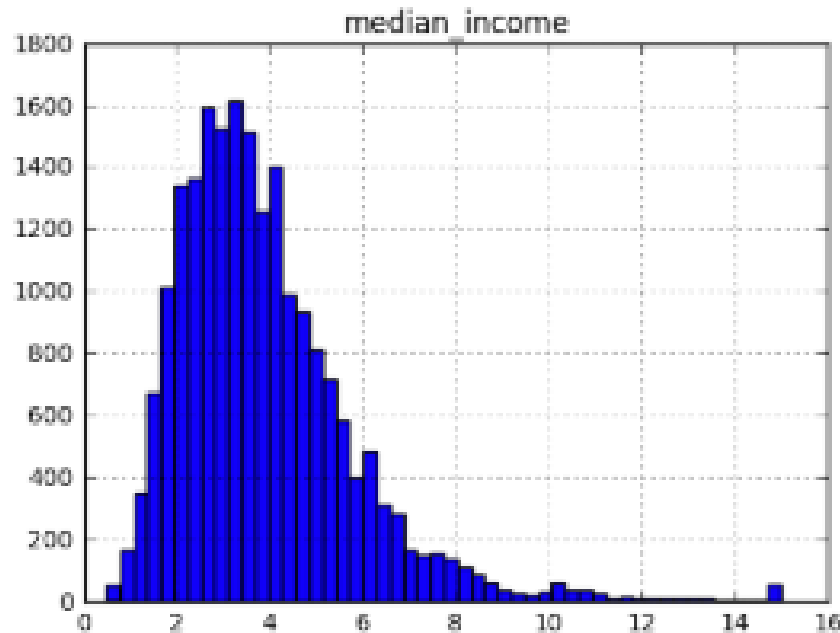
샘플링 편향

◆ 계층적 샘플링

- 샘플에서도 전체 비율 유지해야 함

◆ 중간 주택 가격 예측 시 중간 소득이 중요

- 소득 카테고리 특성 중요
- 중간 소득 히스토그램 : 대부분 \$20,000~\$50,000 / 일부 \$60,000 이상
- 계층별로 데이터셋에 충분한 샘플수 필요

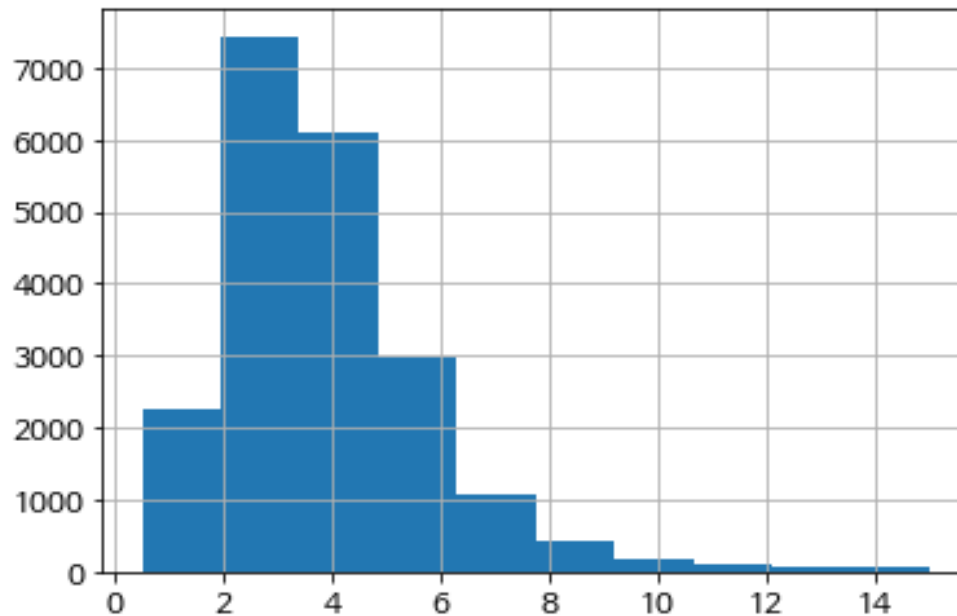


소득 카테고리 “income_cat”

- ◆ 대부분 데이터 : \$20,000 ~ \$50,000 사이에 모여 있음
- ◆ 일부 데이터만 \$60,000을 넘어감

```
housing["median_income"].hist()
```

<matplotlib.axes._subplots.AxesSubplot at 0x230675212b0>



소득 카테고리 “income_cat”

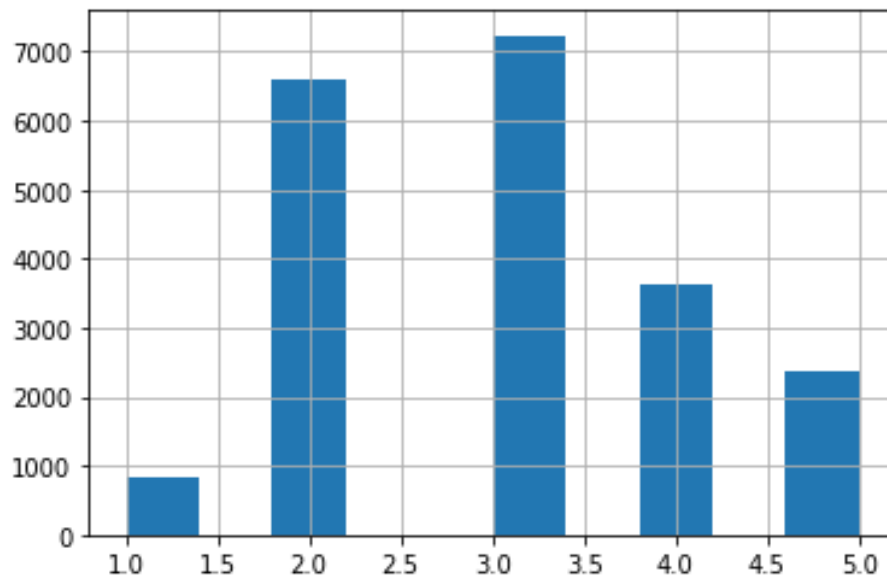
- ◆ 소득 카테고리 개수 제한 : 1.5로 나눔
- ◆ 5이상은 5로 레이블링

```
import numpy as np

# 소득 카테고리 개수를 제한하기 위해 1.5로 나눕니다.
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
# 5 이상은 5로 레이블합니다.
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

```
housing["income_cat"].hist()
```

<matplotlib.axes._subplots.AxesSubplot at 0x2536f4e8fc8>



소득 카테고리 기반 계층 샘플링

◆ scikit-learn의 StratifiedShuffleSplit

- StratifiedKFold의 계층 샘플링 + ShuffleSplit의 랜덤 샘플링
- test_size와 train_size 매개변수 합을 1이하로 지정 가능

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

```
housing["income_cat"].value_counts() / len(housing)
```

```
3.0    0.350581
2.0    0.318847
4.0    0.176308
5.0    0.114438
1.0    0.039826
```

```
Name: income_cat, dtype: float64
```

계층 샘플링 vs. 랜덤 샘플링 비교

- ◆ 계층 샘플링 : StratifiedShuffleSplit
- ◆ 랜덤 샘플링 : train_test_split

	Overall	Stratified	Random	Rand. %error	Strat. %error
1.0	0.039826	0.039729	0.040213	0.973236	-0.243309
2.0	0.318847	0.318798	0.324370	1.732260	-0.015195
3.0	0.350581	0.350533	0.358527	2.266446	-0.013820
4.0	0.176308	0.176357	0.167393	-5.056334	0.027480
5.0	0.114438	0.114583	0.109496	-4.318374	0.127011

- ◆ 샘플링 후 “income_cat” 특성 삭제

```
for set_ in (strat_train_set, strat_test_set):  
    set_.drop("income_cat", axis=1, inplace=True)
```

2.4 데이터 이해를 위한 탐색과 시각화

◆ train_set 복사본 생성 후 사용

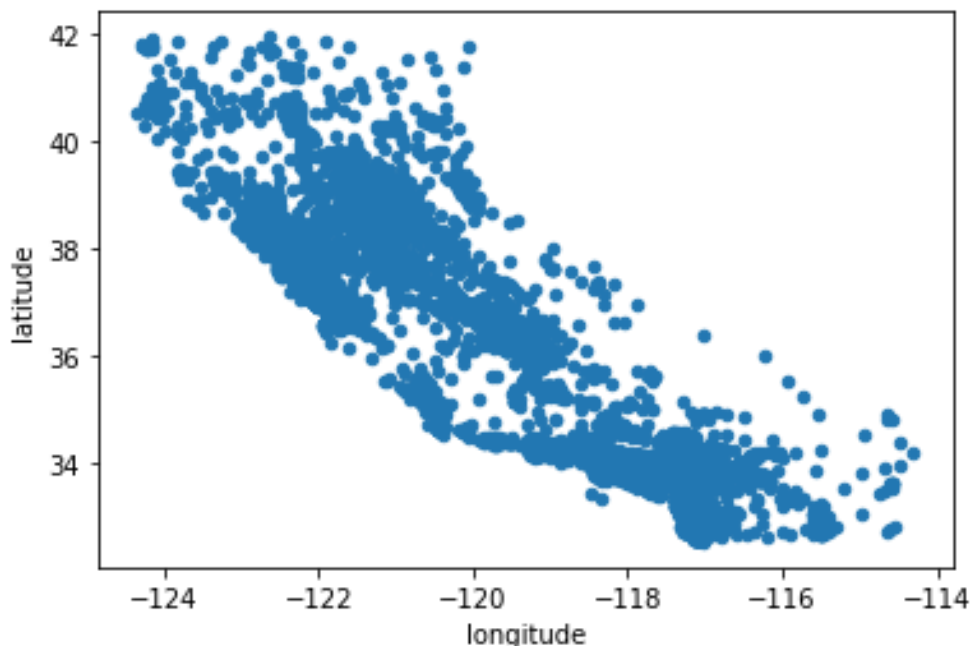
```
housing = strat_train_set.copy()
```

◆ 2.4.1 지리적 데이터 시각화

- 모든 구역 산점도

```
ax = housing.plot(kind="scatter", x="longitude", y="latitude")  
ax.set(xlabel='longitude', ylabel='latitude')
```

```
[Text(0, 0.5, 'latitude'), Text(0.5, 0, 'longitude')]
```

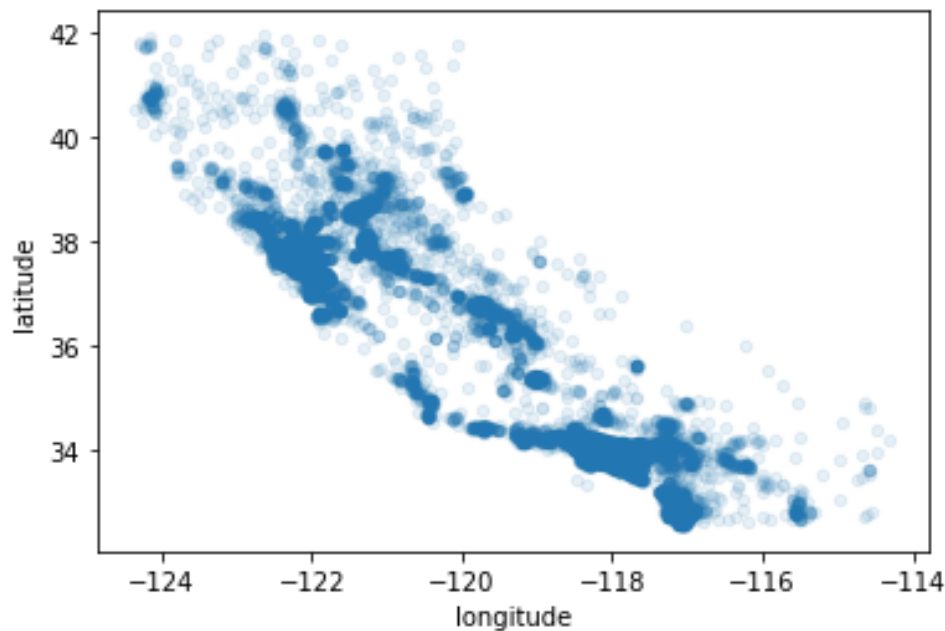


지리적 데이터 시각화

- 데이터 포인트가 밀집된 영역을 잘 보기 위해 투명도 추가
 - alpha option = 0.1

```
ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)  
ax.set(xlabel='longitude', ylabel='latitude')
```

```
[Text(0, 0.5, 'latitude'), Text(0.5, 0, 'longitude')]
```



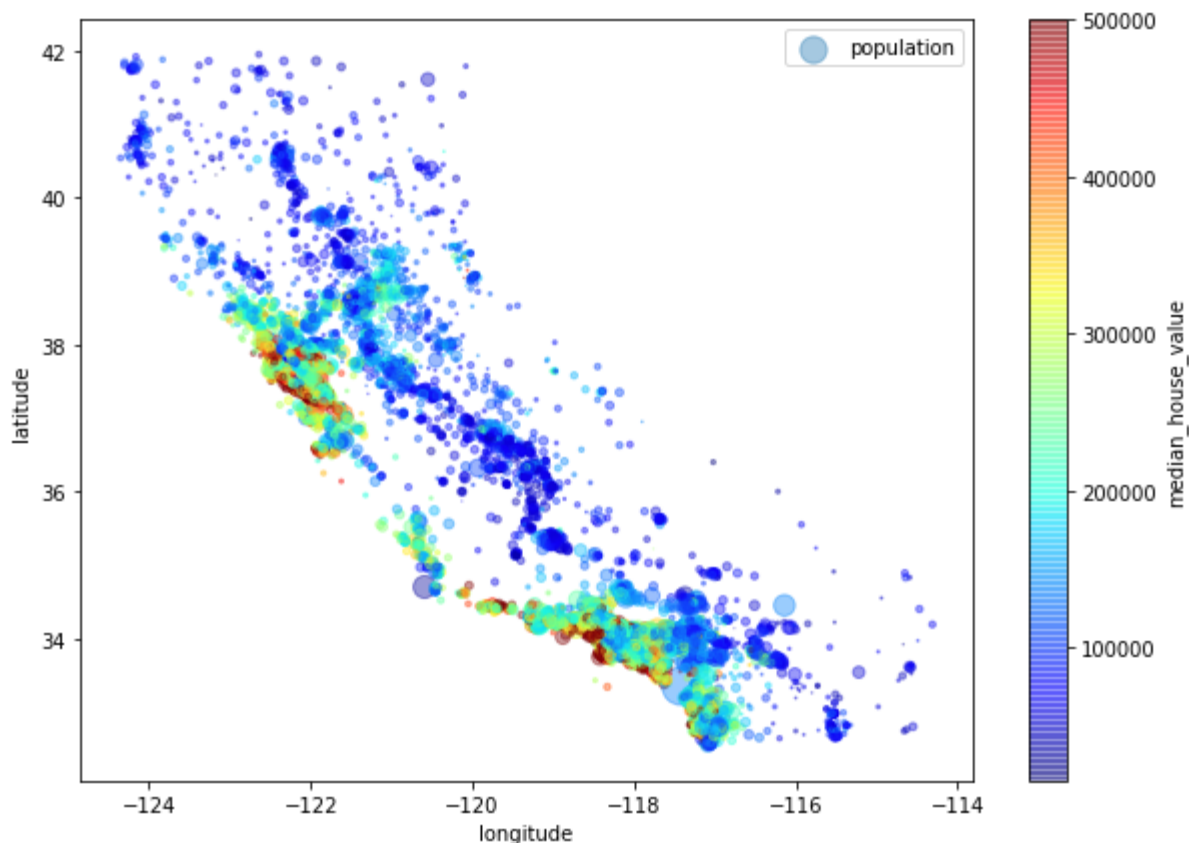
지리적 데이터 시각화

◆ 캘리포니아 주택 가격

- 원의 반지름 : 구역의 인구 (s)
- 색깔 : 가격 (c)

```
ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,  
                  s=housing["population"]/100, label="population", figsize=(10,7),  
                  c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True, sharex=False)  
plt.legend()
```

<matplotlib.legend.Legend at 0x25375b00c08>



2.4.2 상관관계 조사

◆ 표준 상관계수 (standard correlation coefficient)

- Pearson's r
- corr() 함수 이용

```
corr_matrix = housing.corr()  
corr_matrix["median_house_value"].sort_values(ascending=False)
```

median_house_value	1.000000
median_income	0.687160
total_rooms	0.135097
housing_median_age	0.114110
households	0.064506
total_bedrooms	0.047689
population	-0.026920
longitude	-0.047432
latitude	-0.142724

Name: median_house_value, dtype: float64

- 상관관계 범위 : -1 ~ 1
- 1에 가까우면 강한 양의 상관관계
 - 예) 중간 소득이 올라갈 때 주택 가격 증가
- -1에 가까우면 강한 음의 상관관계
 - 예) 위도가 커질수록(북쪽) 주택 가격이 조금씩 감소
- 0에 가까우면 선형적인 상관관계 없음

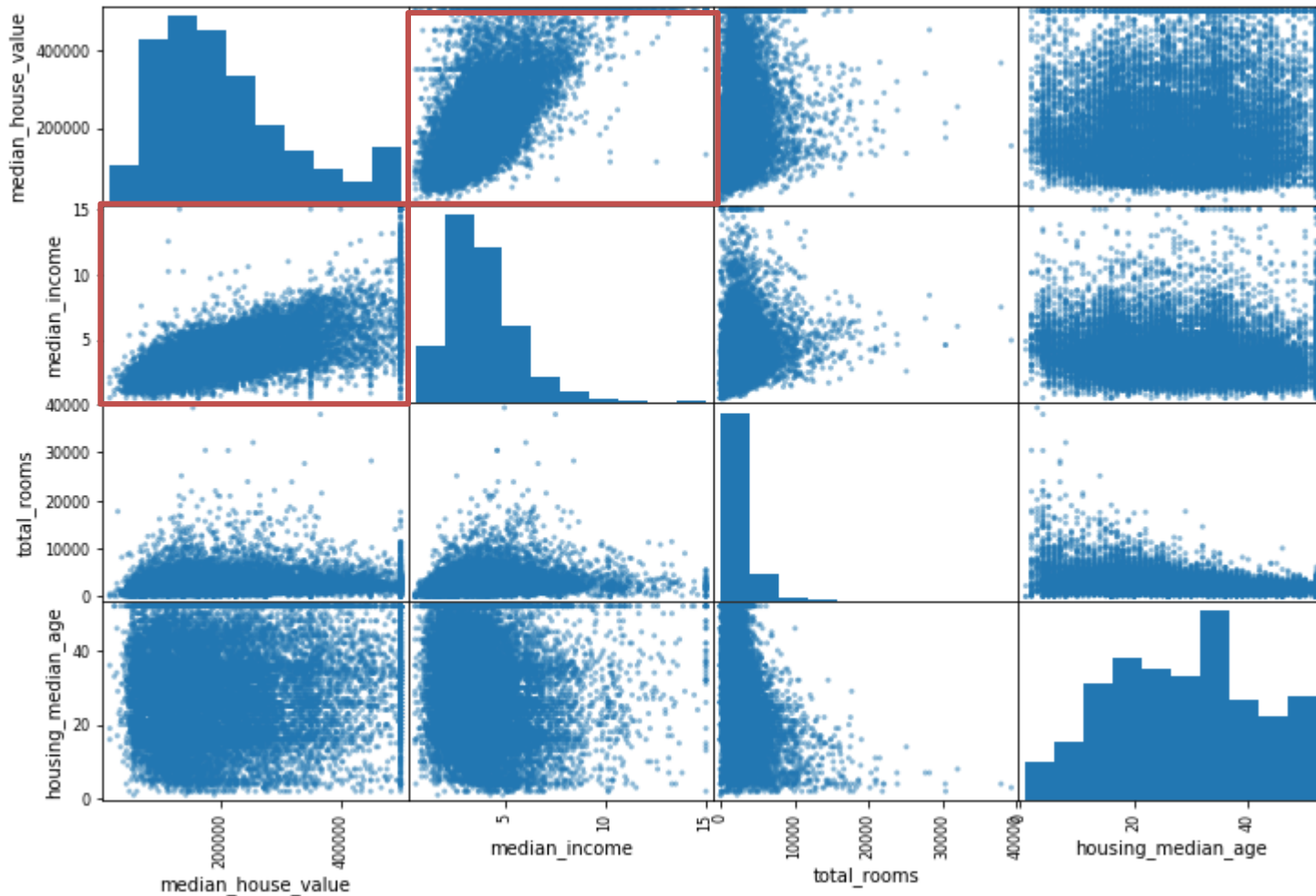
표준상관계수 그래프



◆ 중간 주택 가격과 상관 관계가 높아 보이는 특성 4개

```
from pandas.plotting import scatter_matrix
```

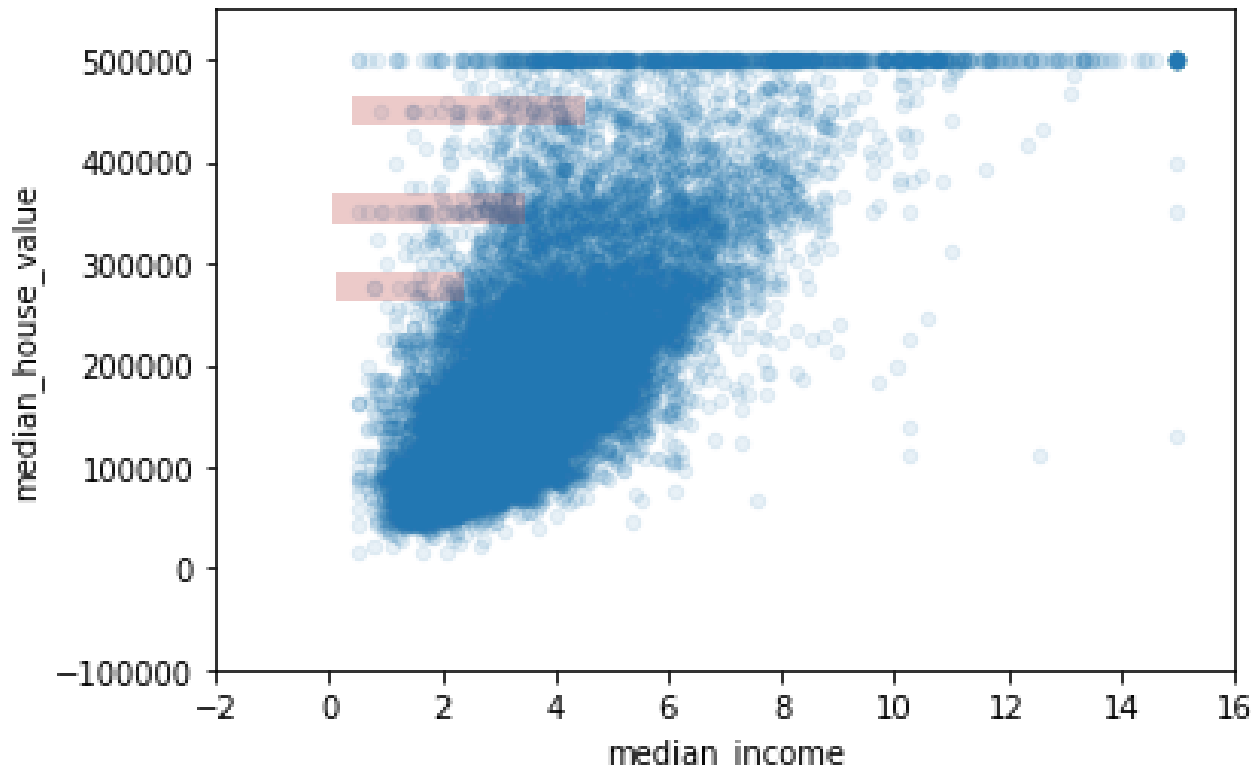
```
attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]  
scatter_matrix(housing[attributes], figsize=(12, 8))
```



중간 소득 vs. 중간 주택 가격

```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)  
plt.axis([-2, 16, -100000, 550000])
```

[-2, 16, -100000, 550000]



이상한 형태를
학습하지 않도록
해당 구역 제거하는
것이 좋음

2.4.3 특성 조합

◆ 머신러닝 알고리즘 적용 전 확인

- 이상한 데이터 확인 → data refining 필요
- 상관관계 확인
- 여러 특성 조합 시도
 - 예) 방 개수 - 가구수 (실제로 필요한 정보 : 가구당 방 개수)
 - 예) 침대 개수 보다는 방 개수와 비교가 더 나음
 - 예) 가구 당 인원

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]  
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]  
housing["population_per_household"] = housing["population"]/housing["households"]
```

```
corr_matrix = housing.corr()  
corr_matrix["median_house_value"].sort_values(ascending=False)
```

◆ 반복적으로 탐색

◆ 프로토타입 생성 → 실행 → 결과분석 → 다시 탐색

median_house_value	1.000000
median_income	0.687160
rooms_per_household	0.146285
total_rooms	0.135097
housing_median_age	0.114110
households	0.064506
total_bedrooms	0.047689
population_per_household	-0.021985
population	-0.026920
longitude	-0.047432
latitude	-0.142724
bedrooms_per_room	-0.259984

Name: median_house_value, dtype: float64

2.5 머신러닝 알고리즘을 위한 데이터 준비

◆ 데이터 준비 자동화

- 데이터 변환 손쉽게 반복 (예: 다음번에 새로운 데이터셋 사용 시)
- 다른 프로젝트에 재사용 가능 변환 라이브러리 구축
- 론칭 후 새 데이터에 적용 시 사용
- 데이터 변화 쉽게 시도, 최적의 조합을 찾는데 편리

◆ 예측 변수와 레이블 분리

- 레이블(정답) : 중간 주택 가격 해당열 (axis=1)

```
housing = strat_train_set.drop("median_house_value", axis=1) # 훈련 세트를 위해 레이블열 삭제
housing_labels = strat_train_set["median_house_value"].copy()
```

```
housing.head() # 레이블 제거된 데이터
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<1H OCEAN
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<1H OCEAN
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEAR OCEAN
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	INLAND
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<1H OCEAN

```
housing_labels.head() # 레이블 데이터
```

```
17606    286600.0
18632    340600.0
14650    196900.0
3230     46300.0
3555     254500.0
Name: median_house_value, dtype: float64
```

2.5.1 데이터 정제

◆ 누락된 특성 처리

- 예) total_bedrooms

```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0	2.2708	near bay
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0	5.1762	near bay
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0	4.6328	near bay
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0	1.6675	near bay
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0	3.1662	near bay

- 옵션 2: 전체 특성 삭제
- 옵션 3: 어떤 값을 채우기 (0, 평균, 중간값 등)

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"]) # 옵션 1
```

```
sample_incomplete_rows.drop("total_bedrooms", axis=1) # 옵션 2
```

```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # 옵션 3
```

2.5.1 데이터 정제

◆ scikit-learn **Imputer** 함수 : 누락된 값 처리

```
from sklearn.impute import SimpleImputer  
  
imputer = SimpleImputer(strategy="median")
```

- 중간값은 수치형 특성에서만 계산 가능. 텍스트 특성 제외 복사본 생성
- fit() 메서드를 사용해 훈련 데이터에 적용

```
housing_num = housing.drop('ocean_proximity', axis=1)  
# 다른 방법: housing_num = housing.select_dtypes(include=[np.number])
```

```
imputer.fit(housing_num)
```

```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,  
              missing_values=nan, strategy='median', verbose=0)
```

- imputer는 결과를 객체의 statistics_ 속성에 저장 → 데이터 확인

```
imputer.statistics_
```

```
array([[-118.51 ,  34.26 ,  29.    , 2119.5 ,  433.    , 1164.    ,  
        408.    ,  3.5409])
```

각 특성의 중간 값이 수동으로 계산한 것과 같은지 확인해 보세요:

```
housing_num.median().values
```

```
array([[-118.51 ,  34.26 ,  29.    , 2119.5 ,  433.    , 1164.    ,  
        408.    ,  3.5409])
```

2.5.1 데이터 정제

- 학습된 imputer 객체를 사용해 훈련 세트에서 누락된 값 → 학습한 중간값으로 변경
- 그 결과는 변형된 특성들이 있는 평범한 Numpy 배열
- 다시 pandas DataFrame으로 변경

```
X = imputer.transform(housing_num)
```

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,  
                          index = list(housing.index.values))
```

```
housing_tr.loc[sample_incomplete_rows.index.values]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	4.6328
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662

2.5.2 텍스트와 범주형 특성 다루기

◆ “ocean_proximity” 카테고리 텍스트 → 숫자로 변형

- 각 카테고리를 다른 정수값으로 매핑 : pandas의 factorize() 함수

```
housing_cat = housing["ocean_proximity"]  
housing_cat.head(10)
```

```
17606    <1H OCEAN  
18632    <1H OCEAN  
14650    NEAR OCEAN  
3230     INLAND  
3555     <1H OCEAN  
19480     INLAND  
8879     <1H OCEAN  
13685     INLAND  
4937     <1H OCEAN  
4861     <1H OCEAN  
Name: ocean_proximity, dtype: object
```

```
housing_cat_encoded, housing_categories = housing_cat.factorize()  
housing_cat_encoded[:10]
```

```
array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0], dtype=int32)
```

```
housing_categories
```

```
Index(['<1H OCEAN', 'NEAR OCEAN', 'INLAND', 'NEAR BAY', 'ISLAND'], dtype='object')
```

- one-hot encoding으로 변환 필요 (한 특성만 1이고, 나머지는 0)

scikit-learn OneHotEncoder

- ◆ 숫자로 된 범주형 값을 one-hot vector로 변환
- ◆ `fit_transform(2차원 배열)` : `reshape`으로 2차원으로 변형 필요
 - 출력 형태 : SciPy의 sparse matrix (희소 행렬)
 - '0'이 아닌 원소의 위치만 저장 (메모리 절약)

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(categories='auto')
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot
```

```
<16512x5 sparse matrix of type '<class 'numpy.float64''>'
  with 16512 stored elements in Compressed Sparse Row format>
```

- ◆ `numpy` 배열로 변형 : `toarray()` 함수

```
housing_cat_1hot.toarray()

array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

2.5.3 나만의 변환기

- ◆ **scikit-learn**은 유용한 변환기를 많이 제공
- ◆ **나만의 변환기**
 - pipeline 클래스와 연계 가능
 - 파이썬 클래스 생성해서 사용
 - `fit()`, `transform()`, `fit_transform()` 함수
 - `TransformerMixin` 상속 → `fit_transform()` 함수 제공
 - `BaseEstimator` 상속 → `get_params()`, `set_params()` 함수 제공

2.5.3 나만의 변환기

◆ 조합 특성 추가 변환기 구현

```
from sklearn.base import BaseEstimator, TransformerMixin

# 컬럼 인덱스
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kwargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                          bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

2.5.3 나만의 변환기

◆ 변환기를 거친 데이터 확인

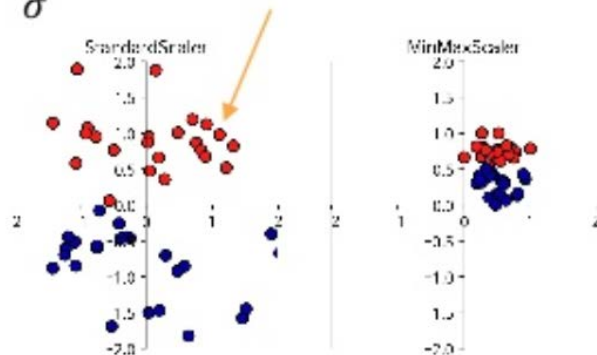
```
housing_extra_attribs = pd.DataFrame(  
    housing_extra_attribs,  
    columns=list(housing.columns)+["rooms_per_household", "population_per_household"])  
housing_extra_attribs.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	rooms_per_household
0	-121.89	37.29	38	1568	351	710	339	2.7042	<1H OCEAN	4.62537
1	-121.93	37.05	14	679	108	306	113	6.4214	<1H OCEAN	6.00885
2	-117.2	32.77	31	1952	471	936	462	2.8621	NEAR OCEAN	4.22511
3	-119.61	36.31	25	1847	371	1460	353	1.8839	INLAND	5.23229
4	-118.59	34.23	17	6592	1525	4459	1463	3.0347	<1H OCEAN	4.50581

2.5.4 특성 스케일링(feature scaling)

- ◆ 머신러닝 알고리즘은 입력 숫자 특성들의 scale이 많이 다르면 잘 작동하지 않음
 - 예) 전체 방 개수 범위 6~39,320 / 중간소득범위 0~15
- ◆ 특성의 범위를 같도록...
 - min-max scaling : 정규화 (normalization)
 - 0~1 범위에 들도록 값 이동 및 스케일 조정
 - 데이터-최소값 / 최대값-최소값
 - scikit-learn MinMaxScaler 변환기
 - 표준화 (standardization)
 - 평균을 뺀 후, 표준편차로 나누어 결과 분포의 분산이 1이 되도록 함
 - scikit-learn StandardScaler 변환기

$\frac{x - \bar{x}}{\sigma}$ 표준점수, z-점수: 평균 0, 분산 1



$$\frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

모든 특성이
0과 1사이 에 위치
이상치에 민감

2.5.5 변환 파이프라인

◆ Pipeline 클래스

- scikit-learn에서 연속된 변환을 순서대로 처리

◆ 숫자 특성 처리 파이프라인

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
housing_num_tr
```

```
array([[ -1.15604281,  0.77194962,  0.74333089, ..., -0.31205452,
        -0.08649871,  0.15531753],
       [ -1.17602483,  0.6596948 , -1.1653172 , ...,  0.21768338,
        -0.03353391, -0.83628902],
       [  1.18684903, -1.34218285,  0.18664186, ..., -0.46531516,
        -0.09240499,  0.4222004 ],
       ...,
       [  1.58648943, -0.72478134, -1.56295222, ...,  0.3469342 ,
        -0.03055414, -0.52177644],
       [  0.78221312, -0.85106801,  0.18664186, ...,  0.02499488,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,  1.85670895, ..., -0.22852947,
        -0.09586294,  0.10180567]])
```

◆ pandas의 dataframe 처리하는 변환기 작성

- DataFrameSelector 클래스 : 나머지는 버리고, 필요한 특성을 선택하여 데이터프레임을 numpy 배열로 변경 (수치형만 다루는 파이프라인)

```
from sklearn.base import BaseEstimator, TransformerMixin

# 사이킷런이 DataFrame을 바로 사용하지 못하므로
# 수치형이나 범주형 컬럼을 선택하는 클래스를 만듭니다.
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

```
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attribs)),
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attribs)),
    ('cat_encoder', CategoricalEncoder(encoding="onehot-dense")),
])
```

수치형 파이프라인

범주형 파이프라인

◆ 두 파이프라인 연결

```
from sklearn.pipeline import FeatureUnion

full_pipeline = FeatureUnion(transformer_list=[
    ("num_pipeline", num_pipeline),
    ("cat_pipeline", cat_pipeline),
])
```

```
housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
```

```
<16512x16 sparse matrix of type '<class 'numpy.float64'>'
  with 198144 stored elements in Compressed Sparse Row format>
```

```
housing_prepared.shape
```

```
(16512, 16)
```

2.6 모델 선택과 훈련

◆ 2.6.1 훈련 세트에서 훈련하고 평가하기

◆ 모델선택1) 선형 회귀 모델

```
from sklearn.linear_model import LinearRegression
```

```
lin_reg = LinearRegression()  
lin_reg.fit(housing_prepared, housing_labels)
```

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

● 샘플에 적용 확인

```
# 훈련 샘플 몇 개를 사용해 전체 파이프라인을 적용 테스트  
some_data = housing.iloc[:5]  
some_labels = housing_labels.iloc[:5]  
some_data_prepared = full_pipeline.transform(some_data)  
  
print("예측:", lin_reg.predict(some_data_prepared))
```

예측: [210644.60467221 317768.80664627 210956.43338006 59218.98902052
189747.55852462]

```
# 실제 값과 비교  
print("레이블:", list(some_labels))
```

레이블: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]

RMSE 측정

◆ scikit-learn의 mean_square_error() 함수 사용

```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

68628.19819848923

```
housing_labels.mean(), housing_labels.std()
```

(206990.9207243217, 115703.01483031521)

```
lin_reg.score(housing_prepared, housing_labels)
```

0.6481624842804428

- RMSE 예측 오차 : \$68,628 → 낮을수록 좋음
- score 높을수록 좋음 : 0.648... → 과소적합 (underfitting)
- 모델이 train data에 과소적합된 사례
 - 모델이 너무 단순해서 데이터의 내재된 구조를 학습하지 못할 때
 - 해결방법 1) 파라미터가 더 많은 **강력한 모델** 선택 → **먼저 시도**
 - 해결방법 2) 학습 알고리즘에 더 좋은 특성 제공
 - 해결방법 3) 모델 규제 감소 (→ 이 예제에서는 규제 사용 안함)

모델 선택2) 결정 트리

◆ DecisionTreeRegressor 모델 훈련

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=42, splitter='best')
```

- 훈련 세트로 평가

```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

0.0

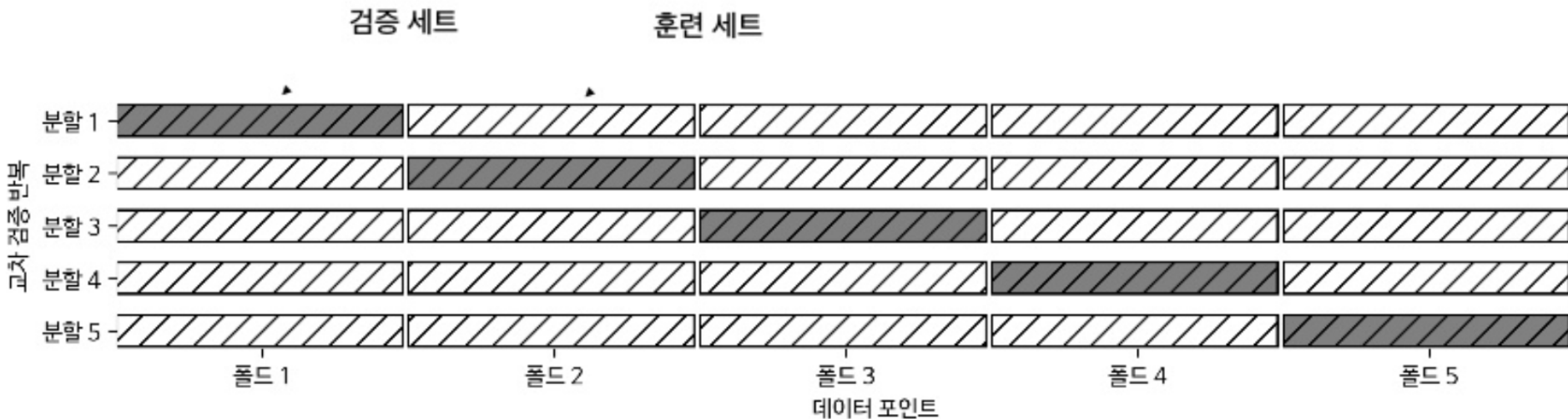
```
tree_reg.score(housing_prepared, housing_predictions)
```

1.0

- 오차 = 0.0
- score = 1.0 → 과대적합 (overfitting)
 - model이 train data에 너무 잘 맞지만, 일반성이 떨어짐

2.6.2 교차 검증을 사용한 평가

- ♦ train set 중 일부를 사용하여 검증에 사용
- ♦ scikit-learn cross-validation (k-fold cross-validation)
 - fold : subset
 - 훈련 세트를 10개의 폴드(서브셋)로 분할 (무작위)
 - 결정트리모델을 10번 훈련하고 평가 (매번 다른 폴드를 선택해서 평가, 나머지 9개 폴드는 훈련에 사용)
 - **결과**) 10개의 평가 점수가 담긴 배열



◆ 결정 트리 검증 결과

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                          scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

```
def display_scores(scores):
    print("점수:", scores)
    print("평균:", scores.mean())
    print("표준편차:", scores.std())
```

```
display_scores(tree_rmse_scores)
```

```
점수: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
       71115.88230639 75585.14172901 70262.86139133 70273.6325285
       75366.87952553 71231.65726027]
평균: 71407.68766037929
표준편차: 2439.4345041191004
```

◆ 회귀 모델 검증 결과

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                              scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
점수: [66782.73840648 66960.11770454 70347.95253774 74739.57050136
       68031.1338941 71193.84183701 64969.63057974 68281.61137785
       71552.9156973 67665.10083676]
평균: 69052.46133728891
표준편차: 2731.6740318925777
```

RandomForestRegressor

- ◆ 무작위로 특성을 선택해서 많은 DecisionTree를 생성하고, 그 예측을 평균 냄
 - **앙상블 학습** : 여러 다른 모델을 모아서 하나의 모델을 만드는 것
 - 머신러닝 알고리즘 성능 극대화 방법 중 하나

```
from sklearn.ensemble import RandomForestRegressor
```

```
forest_reg = RandomForestRegressor(n_estimators=10, random_state=42)  
forest_reg.fit(housing_prepared, housing_labels)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,  
                        max_features='auto', max_leaf_nodes=None,  
                        min_impurity_decrease=0.0, min_impurity_split=None,  
                        min_samples_leaf=1, min_samples_split=2,  
                        min_weight_fraction_leaf=0.0, n_estimators=10,  
                        n_jobs=None, oob_score=False, random_state=42, verbose=0,  
                        warm_start=False)
```

```
housing_predictions = forest_reg.predict(housing_prepared)  
forest_mse = mean_squared_error(housing_labels, housing_predictions)  
forest_rmse = np.sqrt(forest_mse)  
forest_rmse
```

21933.31414779769

```
from sklearn.model_selection import cross_val_score
```

```
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,  
                                scoring="neg_mean_squared_error", cv=10)  
forest_rmse_scores = np.sqrt(-forest_scores)  
display_scores(forest_rmse_scores)
```

점수: [51646.44545909 48940.60114882 53050.86323649 54408.98730149
50922.14870785 56482.50703987 51864.52025526 49760.85037653
55434.21627933 53326.10093303]

평균: 52583.72407377466

표준편차: 2298.353351147122

2.7 모델 세부 튜닝

- ◆ 가능성 있는 2~5개 정도의 모델을 선정하여 저장해 두면 편리함
- ◆ 최적의 하이퍼파라미터를 찾아야 함
- ◆ 가장 단순한 방법 → 만족할만한 하이퍼파라미터 조합을 찾을 때까지 수동으로 조정

2.7.1 그리드 탐색

◆ scikit-learn의 GridSearchCV 사용

- 탐색하고자 하는 하이퍼파라미터와 시도값 지정
- 가능한 모든 하이퍼파라미터 조합에 대해 교차 검증을 사용해 평가

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
]
forest_reg = RandomForestRegressor()

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)

grid_search.fit(housing_prepared, housing_labels)
```

```
GridSearchCV(cv=5, error_score='raise',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                                              max_features='auto', max_leaf_nodes=None,
                                              min_impurity_decrease=0.0, min_impurity_split=None,
                                              min_samples_leaf=1, min_samples_split=2,
                                              min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                                              oob_score=False, random_state=None, verbose=0, warm_start=False),
             fit_params=None, iid=True, n_jobs=1,
             param_grid=[{'max_features': [2, 4, 6, 8], 'n_estimators': [3, 10, 30]}, {'bootstrap': [False], 'max_features': [2, 3, 4], 'n_estimators': [3, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_squared_error', verbose=0)
```

```
grid_search.best_params_
```

```
{'max_features': 6, 'n_estimators': 30}
```

```
grid_search.best_estimator_
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,  
                        max_features=6, max_leaf_nodes=None, min_impurity_decrease=0.0,  
                        min_impurity_split=None, min_samples_leaf=1,  
                        min_samples_split=2, min_weight_fraction_leaf=0.0,  
                        n_estimators=30, n_jobs=1, oob_score=False, random_state=None,  
                        verbose=0, warm_start=False)
```

```
cvres = grid_search.cv_results_  
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):  
    print(np.sqrt(-mean_score), params)
```

```
64223.2850771 {'max_features': 2, 'n_estimators': 3}  
55407.8687859 {'max_features': 2, 'n_estimators': 10}  
52932.3550944 {'max_features': 2, 'n_estimators': 30}  
60222.8024295 {'max_features': 4, 'n_estimators': 3}  
52951.7955765 {'max_features': 4, 'n_estimators': 10}  
50280.7716783 {'max_features': 4, 'n_estimators': 30}  
59048.4337212 {'max_features': 6, 'n_estimators': 3}  
52588.478215 {'max_features': 6, 'n_estimators': 10}  
50031.7461754 {'max_features': 6, 'n_estimators': 30}  
58077.5052279 {'max_features': 8, 'n_estimators': 3}  
51545.0350056 {'max_features': 8, 'n_estimators': 10}  
50083.4490232 {'max_features': 8, 'n_estimators': 30}  
63065.7241353 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}  
54091.1788418 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}  
59798.9622518 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}  
52314.2649346 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}  
59226.5630142 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}  
51963.370483 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

◆ 2.7.1 그리드 탐색

- 비교적 적은 수의 조합 탐구에 적합
- 가능한 모든 조합을 시도
- 하이퍼파라미터마다 몇 개의 값만 탐색

◆ 2.7.2 랜덤 탐색

- 탐색 공간이 커지면 랜덤 탐색 방식이 유용
- 각 반복마다 하이퍼파라미터에 임의의 수를 대입하여, 지정한 횟수만큼 평가
- 하이퍼파라미터마다 각기 다른 값 탐색
- 단순히 반복 횟수를 조절하는 것만으로 하이퍼파라미터 탐색에 투입할 컴퓨팅 자원 제어 가능

◆ 2.7.3 앙상블 방법

- 단일 모델을 연결하여 모델의 그룹으로 만듦
- 예) 결정 트리의 앙상블 → 랜덤 포레스트

2.7.2 랜덤 탐색

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
```

```
param_distributions = {
    'n_estimators': randint(low=1, high=200),
    'max_features': randint(low=1, high=8),
}
```

```
forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distributions,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error',
                                random_state=42, n_jobs=-1)
rnd_search.fit(housing_prepared, housing_labels)
```

```
RandomizedSearchCV(cv=5, error_score='raise',
                    estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
oob_score=False, random_state=42, verbose=0, warm_start=False),
fit_params=None, iid=True, n_iter=10, n_jobs=-1,
param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x0BEBE110>, 'n_estimators': <scipy.st
ats._distn_infrastructure.rv_frozen object at 0x0BEBE6F0>},
pre_dispatch='2*n_jobs', random_state=42, refit=True,
return_train_score=True, scoring='neg_mean_squared_error',
verbose=0)
```

```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49147.1524172 {'max_features': 7, 'n_estimators': 180}
51396.8768969 {'max_features': 5, 'n_estimators': 15}
50797.0573732 {'max_features': 3, 'n_estimators': 72}
50840.744514 {'max_features': 5, 'n_estimators': 21}
49276.1753033 {'max_features': 7, 'n_estimators': 122}
50775.4633168 {'max_features': 3, 'n_estimators': 75}
50681.383925 {'max_features': 3, 'n_estimators': 88}
49612.1525305 {'max_features': 5, 'n_estimators': 100}
50473.0175142 {'max_features': 3, 'n_estimators': 150}
64458.2538503 {'max_features': 5, 'n_estimators': 2}
```

테스트 세트 평가

◆ 마지막에 한번 수행

- test set → predictor, label 데이터
- full pipeline 사용하여 데이터 변환 (transform())
- test set에서 최종 모델 평가

```
final_model = grid_search.best_estimator_  
  
X_test = strat_test_set.drop("median_house_value", axis=1)  
y_test = strat_test_set["median_house_value"].copy()  
  
X_test_prepared = full_pipeline.transform(X_test)  
  
final_predictions = final_model.predict(X_test_prepared)  
  
final_mse = mean_squared_error(y_test, final_predictions)  
final_rmse = np.sqrt(final_mse)
```

```
final_rmse
```

```
48403.473415816981
```

- 최종 RMSE 오차 확인

2.8 론칭, 모니터링, 시스템 유지보수

◆ 전처리와 예측을 포함한 파이프라인

- preparation + linear model

```
full_pipeline_with_predictor = Pipeline([
    ("preparation", full_pipeline),
    ("linear", LinearRegression())
])

full_pipeline_with_predictor.fit(housing, housing_labels)
full_pipeline_with_predictor.predict(some_data)
```

```
array([ 210644.60459286,  317768.80697211,  210956.43331178,
        59218.98886849,  189747.55849879])
```

◆ 모델 저장

- 하이퍼파라미터, 모델 파라미터 모두 저장

```
my_model = full_pipeline_with_predictor
```

```
from sklearn.externals import joblib
joblib.dump(my_model, "my_model.pkl") # DIFF
#...
my_model_loaded = joblib.load("my_model.pkl") # DIFF
```

◆ 론칭

- 실시간 성능 체크를 위한 모니터링 코드 개발
- 분석가의 성능 평가 (예: 해당 분야의 전문가)
- 입력 데이터 모니터링 코드 개발
- 정기적 훈련을 위한 자동화

Any Questions...
Just Ask!

