



2주차 강의자료



요약

- ❖ 머신러닝이란?
- ❖ 왜 머신러닝을 사용하는가?
- ❖ 머신러닝 시스템의 종류 : 지도 학습 / 비지도 학습 / 준지도 학습 / 강화 학습
- ❖ 배치 학습과 온라인 학습
- ❖ 사례 기반 학습과 모델 기반 학습
- ❖ 주요 도전 과제

요약

❖ 부동산 분석 실습

- 시스템 설계
- 분석
- 데이터 가져오기
- 데이터 탐색하기
- 데이터 준비하기

❖ KDD Cup 99 데이터

- 데이터 설명
- 데이터 다운로드 하기
- 데이터 정형화하기



Machine Learning

머신 러닝 개요 (1/5)

❖ 머신러닝이란?

- 명시적인 프로그래밍 없이 컴퓨터가 학습하는 능력을 갖추게 하는 연구 분야 – 아서 사무엘, 1959
- 어떤 작업 T에 대한 컴퓨터 프로그램의 성능을 P로 측정했을 때 경험 E로 인해 성능이 향상됐다면, 이 컴퓨터 프로그램은 작업 T와 성능 측정 P에 대해 경험 E로 학습한 것이다. – 토머스 미첼, 1997
 - 어떤 작업 T: 문제 (예: 스팸 필터)
 - 경험 E: 훈련 데이터 (Training Set)
 - 성능 측정 P: 정확도 등
- 명시적인 규칙을 코딩하지 않고 기계가 데이터로부터 학습하여 어떤 작업을 더 잘하도록 만드는 거

❖ 왜 머신러닝을 사용하는가?

- 전통적인 접근 방법에서는 문제가 단순하지 않아 규칙이 점점 길고 복잡해지므로 유지보수가 매우 힘든 경우
- 전통적인 접근 방법에서는 너무 복잡하거나 알려진 알고리즘이 없는 경우
- 대용량의 데이터를 분석하여 겉으로는 보이지 않던 패턴을 발견함
- [데이터가] 유동적인 환경

머신 러닝 개요 (2/5)

❖ 왜 머신러닝을 사용하는가?

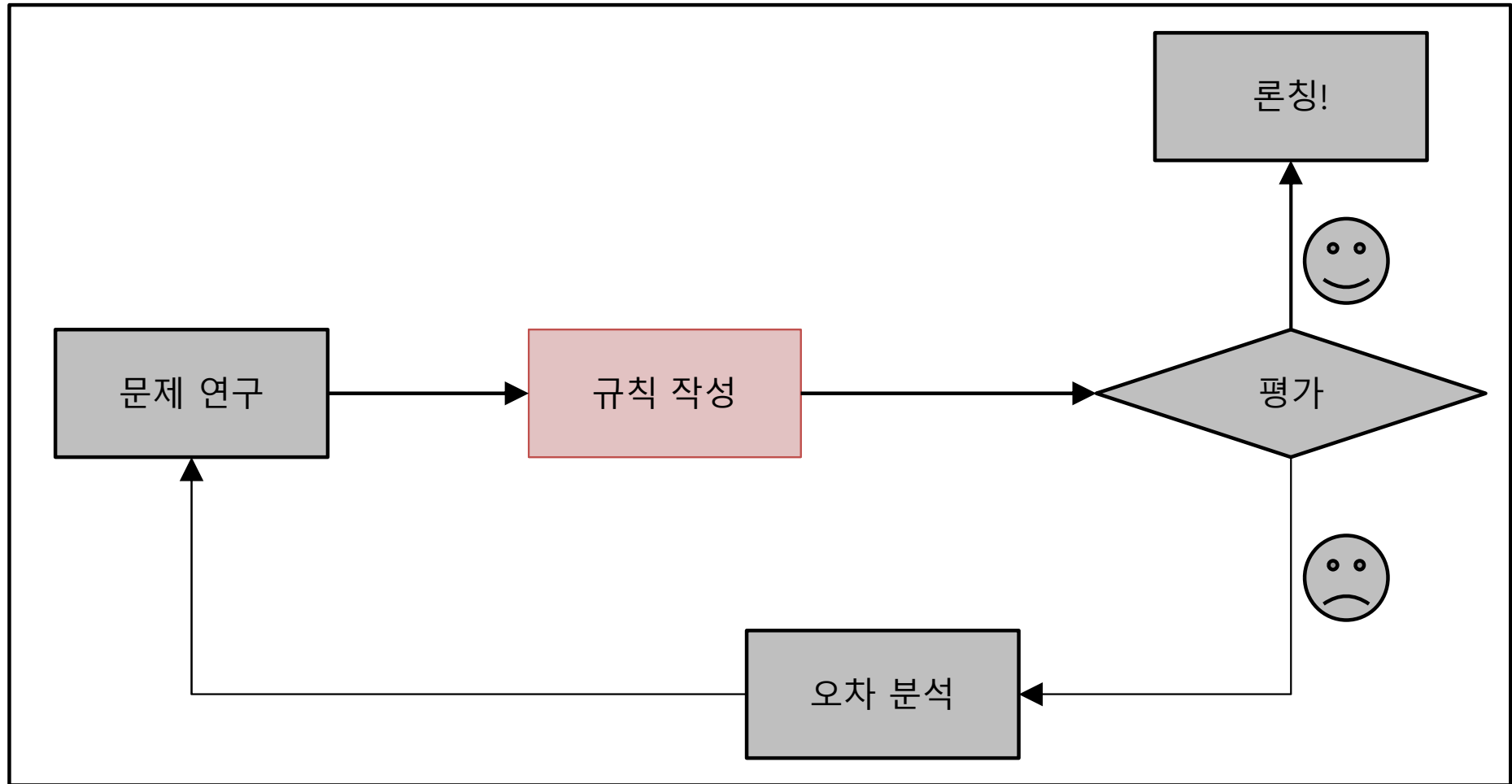


그림 1-1. 전통적인 접근 방법

머신 러닝 개요 (3/5)

❖ 왜 머신러닝을 사용하는가?

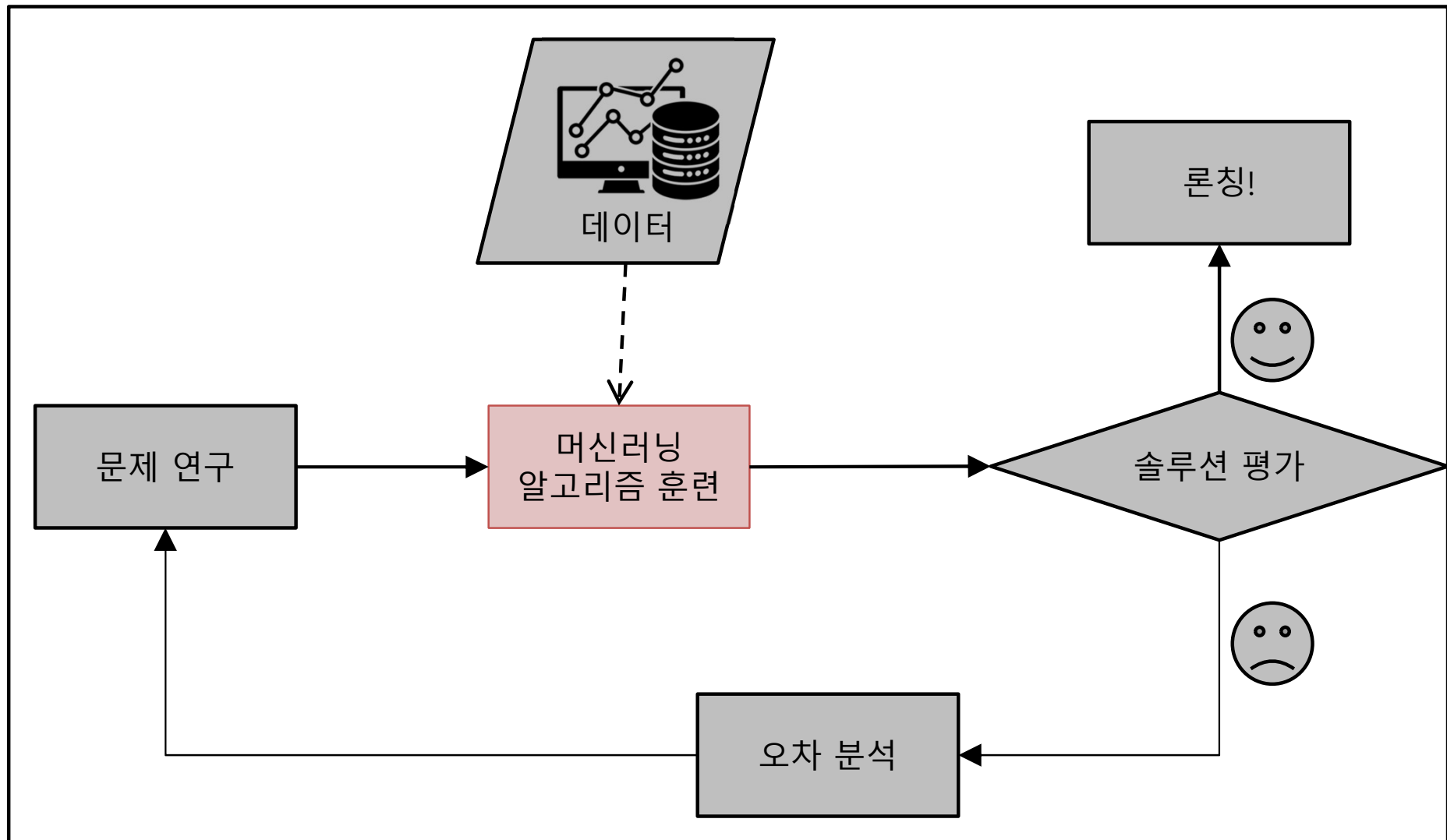


그림 1-2. 머신러닝 접근 방법

머신 러닝 개요 (4/5)

❖ 왜 머신러닝을 사용하는가?

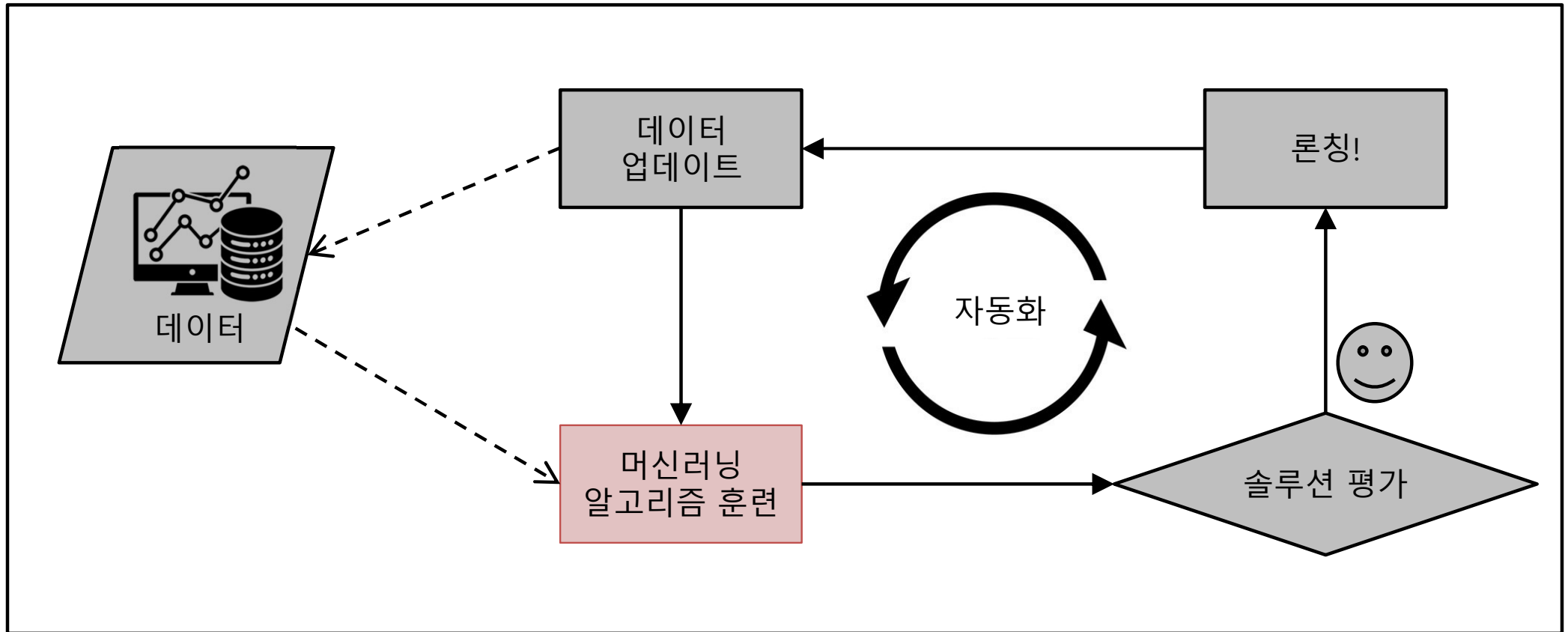


그림 1-3. 자동으로 변화에 적응함

머신 러닝 개요 (5/5)

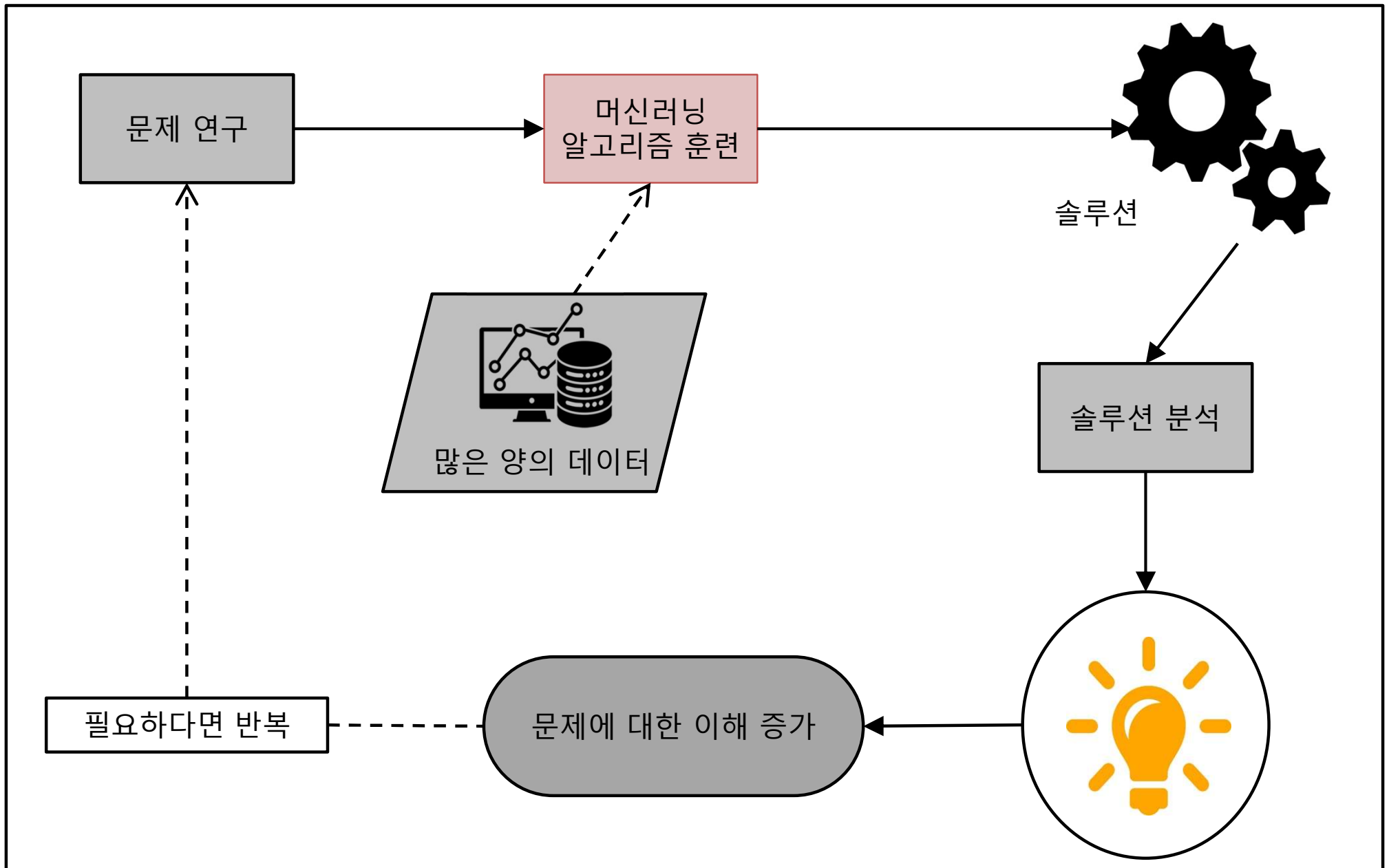


그림 1-4. 머신러닝을 통해 배웁니다.

머신러닝 시스템의 종류

❖ 머신러닝 시스템의 종류

- 지도, 비지도, 준지도, 강화 학습
 - 학습하는 동안의 감독 형태나 정보량에 따라 분류
 - k-nearest neighbors, linear regression, logistic regression, support vector machine (SVM), decision tree & random forests, neural networks
- 온라인 학습과 배치 학습
 - 입력데이터의 스트림으로부터 실시간으로 점진적인 학습을 하는지 아닌지
 - Clustering: k-means, hierarchical cluster analysis (HCA), expectation maximization
 - Visualization & dimensionality reduction: Principal component analysis (PCA), Kernel PCA, Locally-Linear Embedding (LLE), t-distributed stochastic neighbor embedding (t-SNE)
 - Association rule learning: Apriori, Eclat
- 사례 기반 학습과 모델 기반 학습
 - 일반화 방법에 따른 분류
 - 단순히 알고 있는 데이터 포인트와 새 데이터 포인트를 비교하는 것인지 아니면 훈련 데이터 셋에서 과학자들처럼 패턴을 발견하여 예측 모델을 만드는지

❖ 머신 러닝 동작 과정

- 데이터 분석 → 모델 선택 → 훈련 데이터를 이용한 모델 학습 → 새로운 데이터를 이용한 예측(추론(inference)) → 일반화 오류 최소화

지도, 비지도, 준지도, 강화 학습

❖ 지도 학습 (Supervised Learning)

- 알고리즘에 주입하는 훈련 데이터에 레이블(label 혹은 class)이라는 원하는 답이 포함됨
- 분류(Classification)와 회귀(Regression)

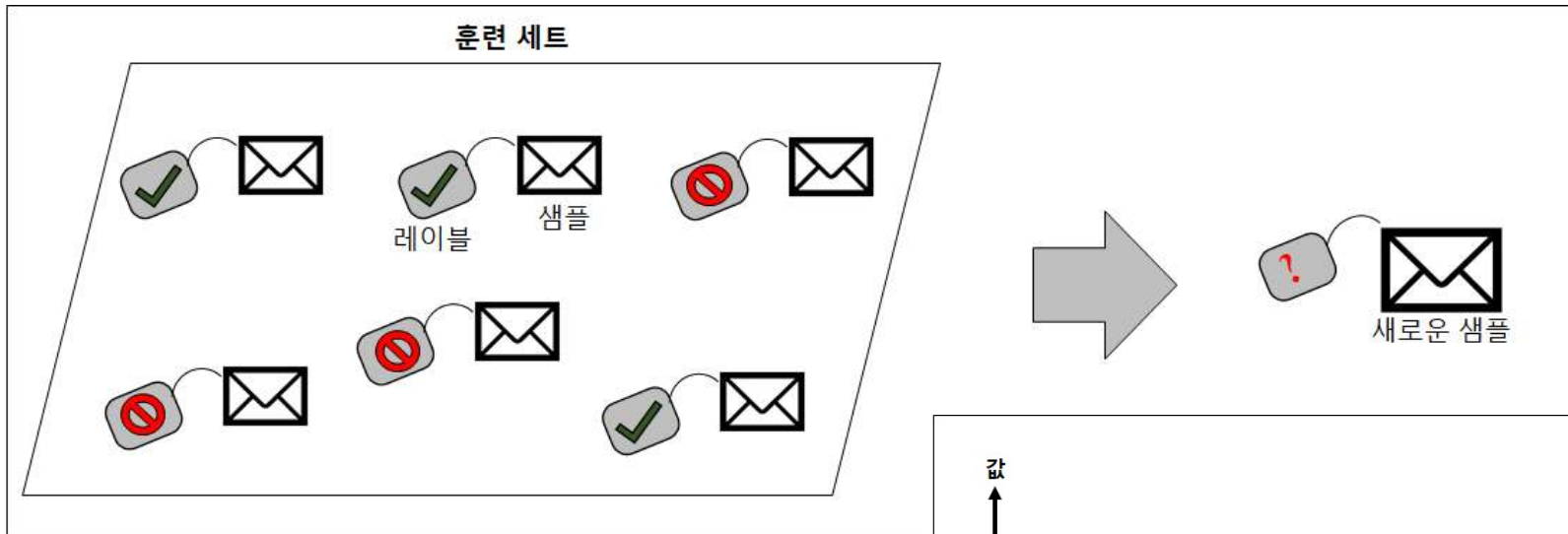


그림 1-5. 지도학습에서 레이블된 훈련 세트(예: 스팸 분류)

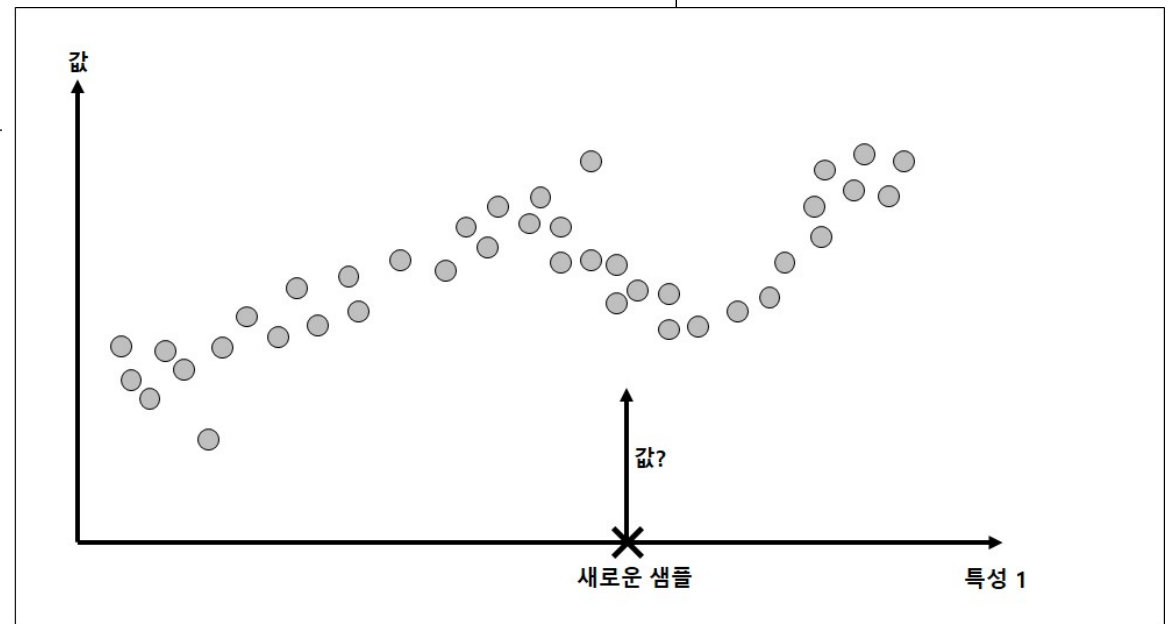


그림 1-6. 회귀

지도, 비지도, 준지도, 강화 학습

❖ 비지도 학습 (Unsupervised Learning)

- 훈련 데이터에 레이블이 없는 경우의 학습 방법
- 군집(clustering)

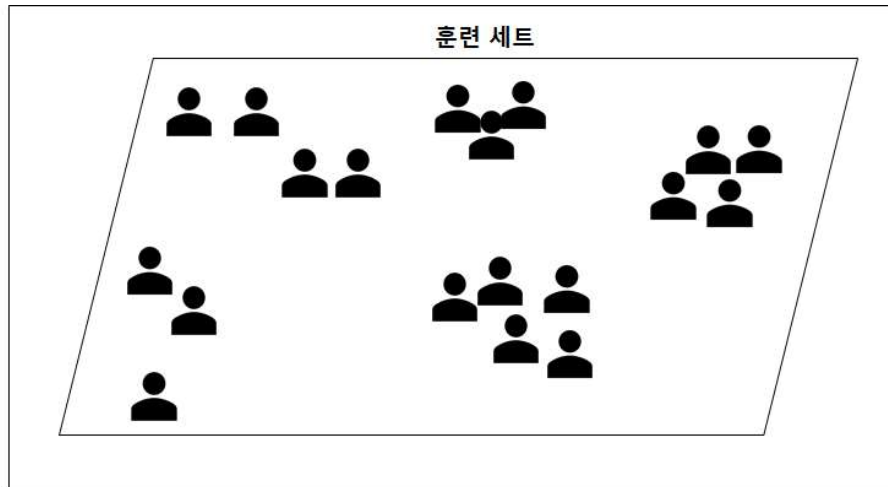


그림 1-7. 비지도 학습에서 레이블이 없는 훈련 세트 (예: 블로그 방문자)

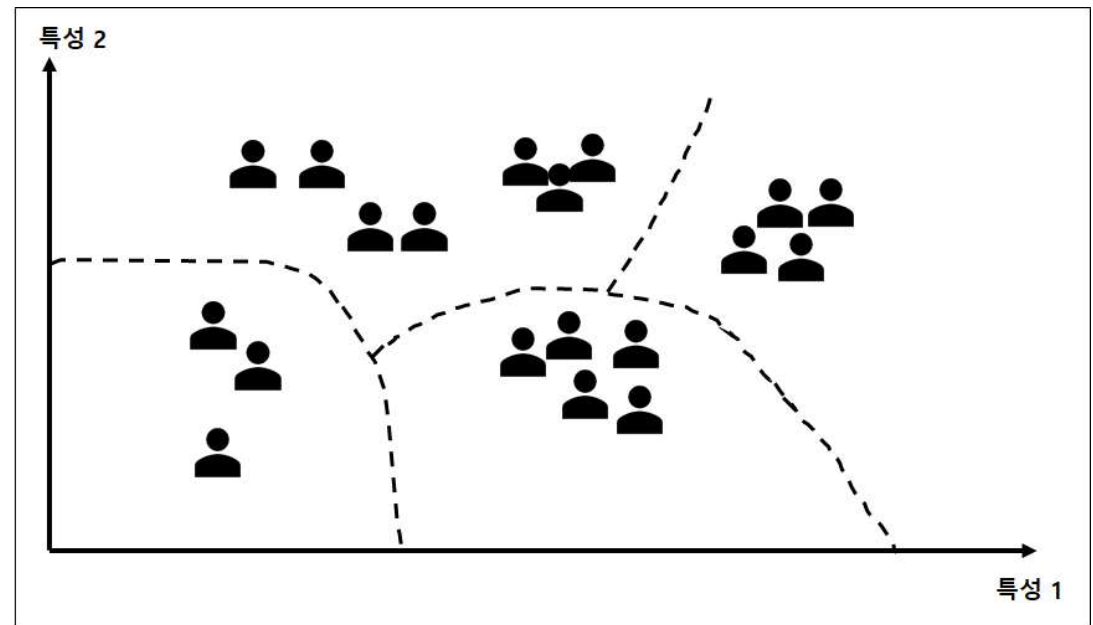


그림 1-8. 군집

지도, 비지도, 준지도, 강화 학습

❖ 비지도 학습 (Unsupervised Learning)

- 시각화(visualization)

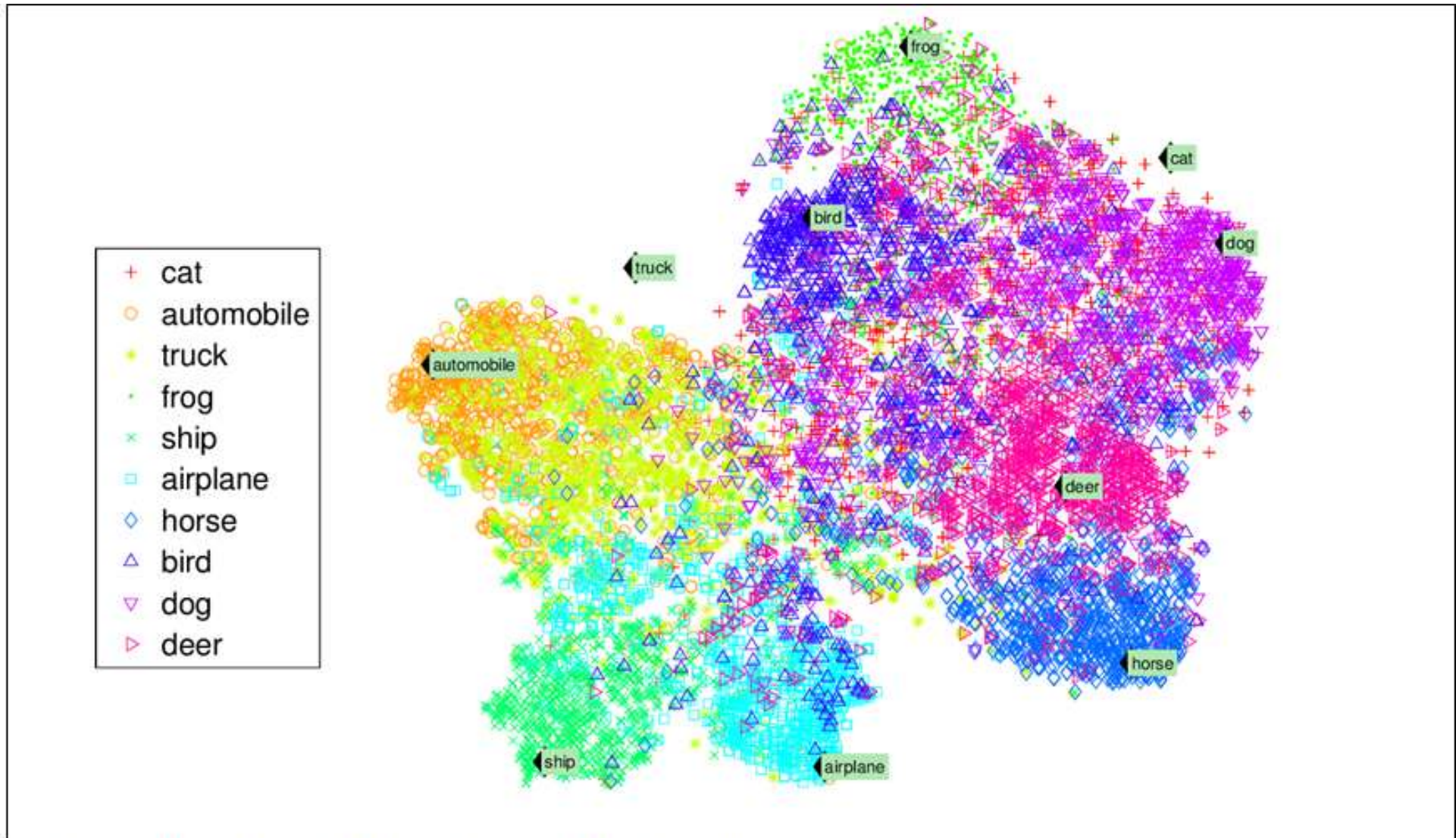


그림 1-9. 의미 있는 군집을 강조한 t-SNE 시각화의 예

지도, 비지도, 준지도, 강화 학습

❖ 비지도 학습 (Unsupervised Learning)

- 이상 탐지(Anomaly detection)

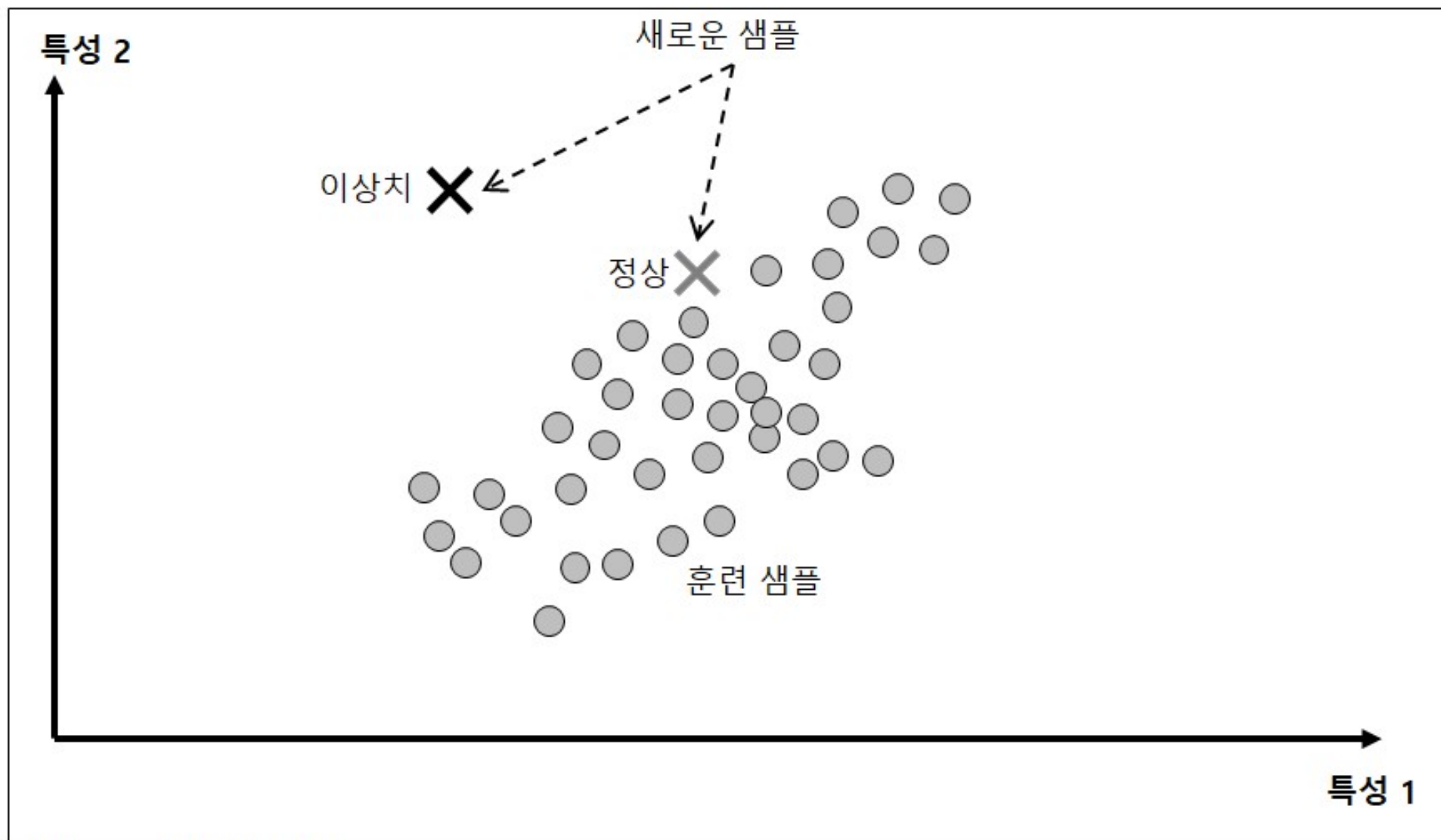


그림 1-10. 이상치 탐지

지도, 비지도, 준지도, 강화 학습

❖ 준지도 학습 (Semi-supervised Learning)

- 레이블이 없는 많은 데이터 + 레이블이 있는 일부 데이터
- 예: 구글 호스팅 서비스

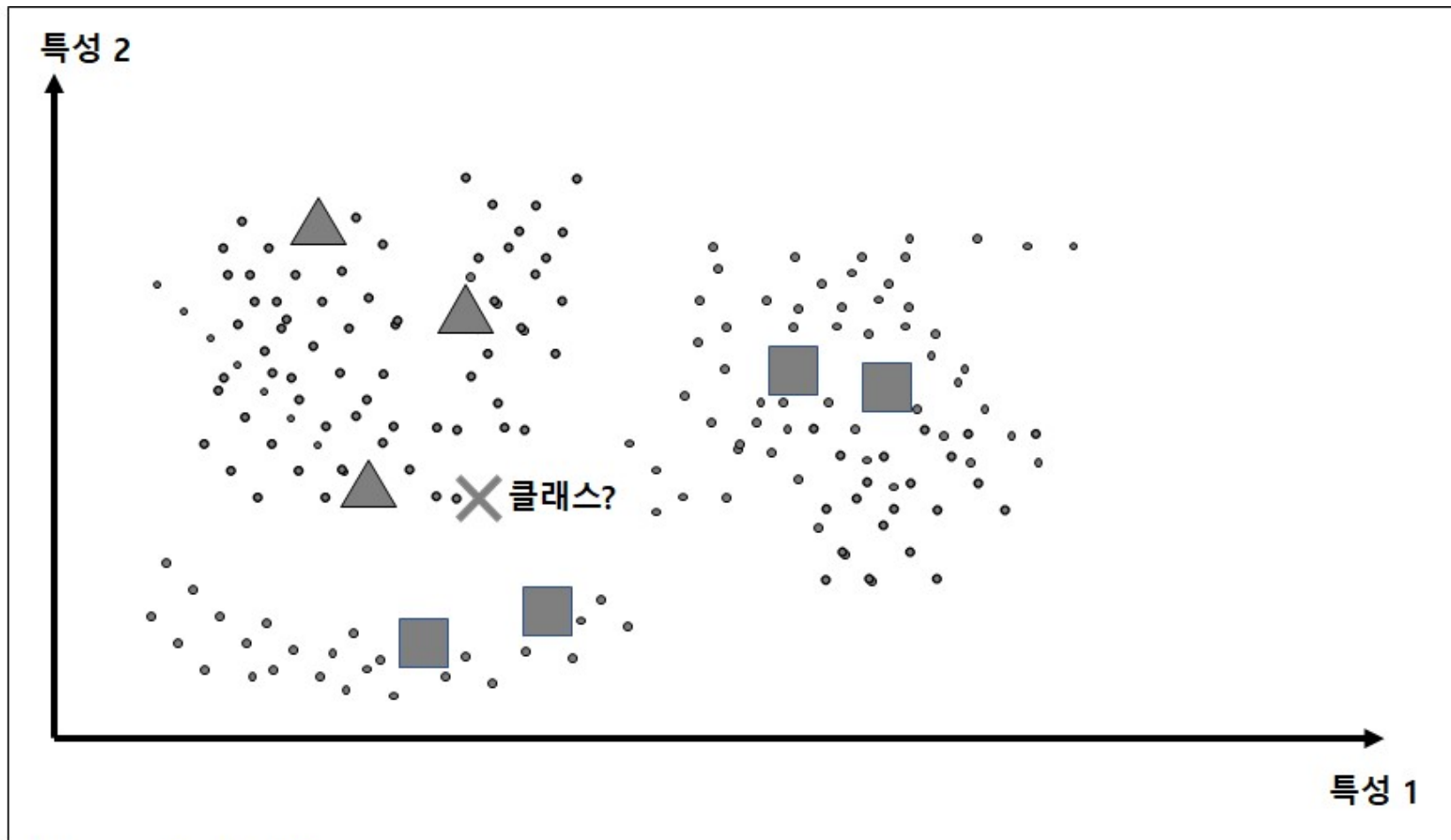


그림 1-11. 준지도 학습

지도, 비지도, 준지도, 강화 학습

❖ 강화 학습

- 학습하는 시스템인 agent가 환경(environment)을 관찰해서 행동(action)을 실행하고, 그 결과로 보상(또는 벌점)을 받음
- 가장 큰 보상을 얻는 주어진 상황에서 agent가 선택할 행동인 정책(policy)을 학습

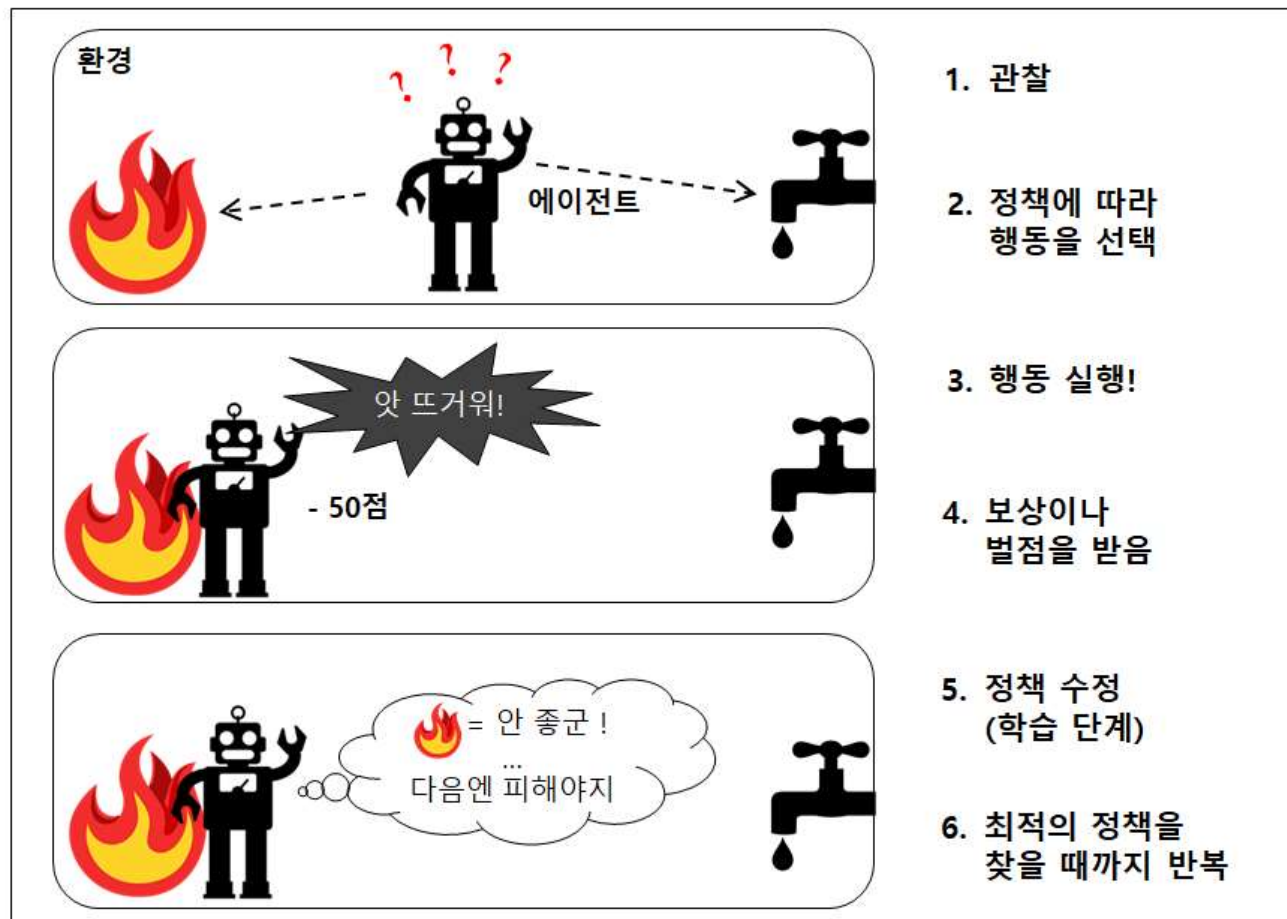


그림 1-12. 강화학습

배치 학습과 온라인 학습

❖ 배치 학습 (batch learning)

- 시스템이 점진적으로 학습할 수 없는 오프라인 학습
- 가용한 데이터를 모두 사용해 훈련시켜야 함

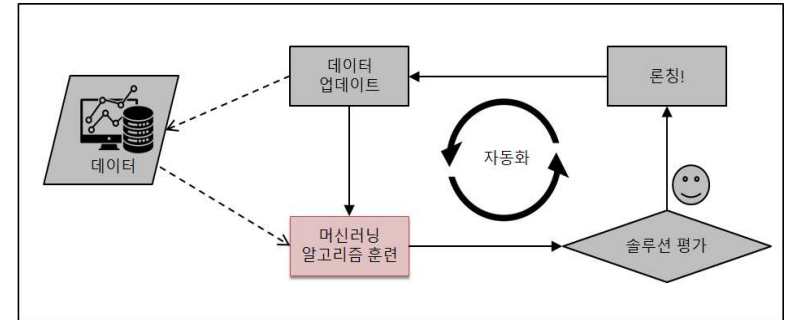


그림 1-3. 자동으로 변화에 적응함

❖ 온라인 학습 (online learning 혹은 incremental learning)

- 데이터를 순차적으로 한 개씩 또는 미니 배치 단위로 주입하여 시스템을 훈련시킴
- 주식 가격과 같이 연속적으로 데이터를 받고 빠른 변화에 스스로 적응해야 하는 경우 적합함
- 단점: 시스템에 나쁜 데이터가 주입될 때 시스템 성능이 점진적으로 감소함

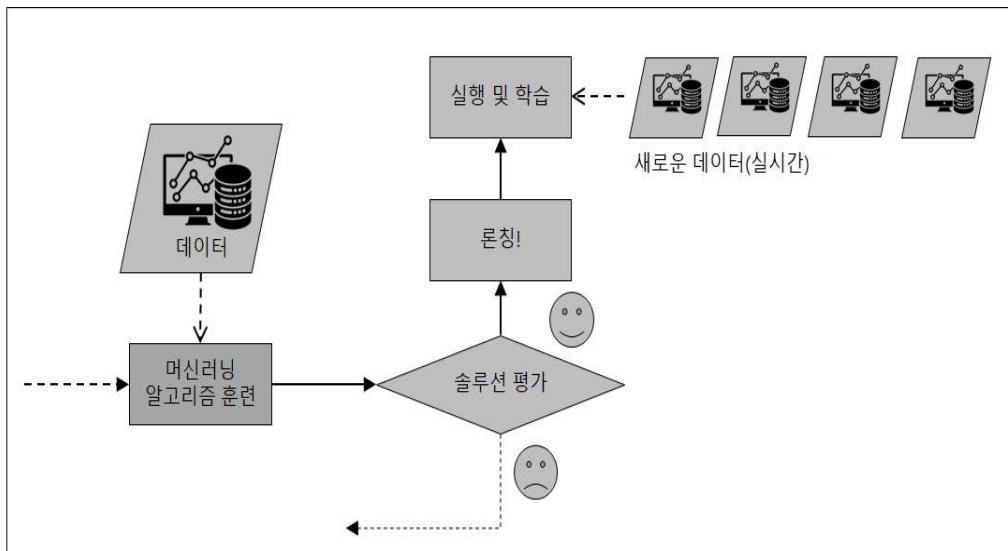


그림 1-13. 온라인 학습

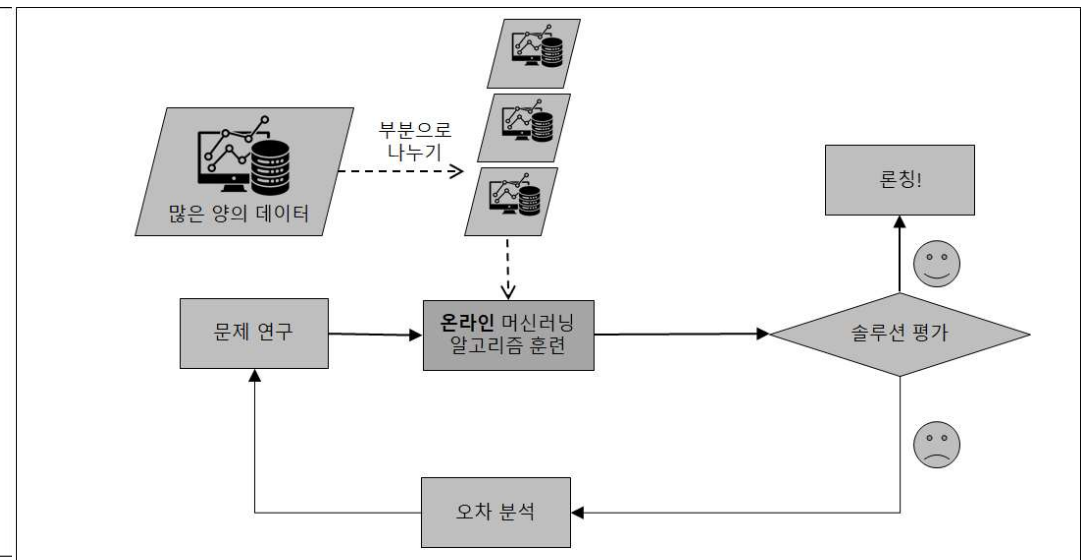


그림 1-14. 온라인 학습을 사용한 대량의 데이터 처리

사례 기반 학습과 모델 기반 학습

❖ 사례 기반 학습 (instance-based learning)

- 시스템에 사례를 기억함으로써 학습함
- 유사도(similarity)를 측정하여 새로운 데이터에 일반화 함

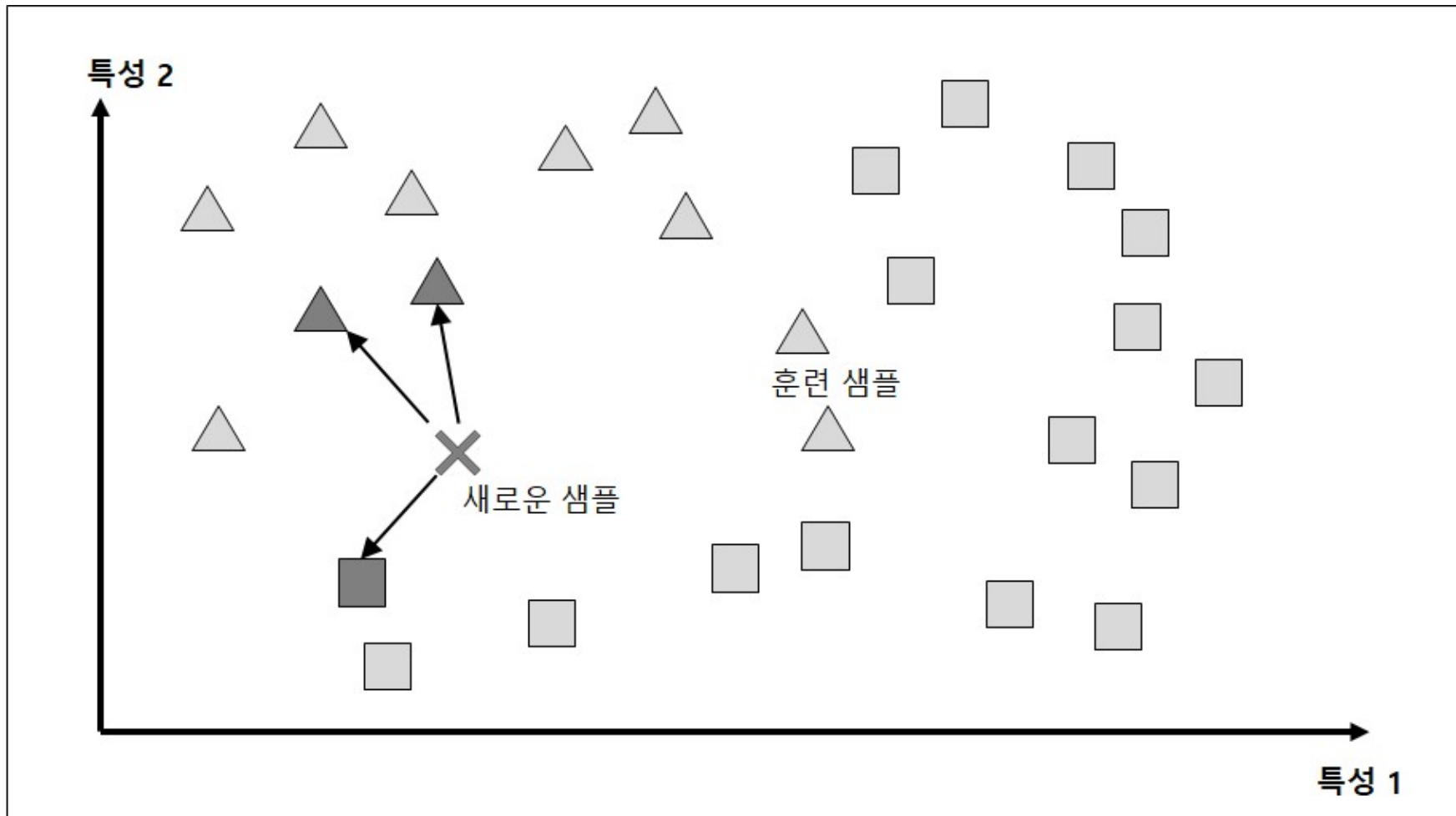


그림 1-15. 사례 기반 학습

사례 기반 학습과 모델 기반 학습

❖ 모델 기반 학습 (model-based learning)

- 샘플들의 모델을 만들어 예측에 사용함

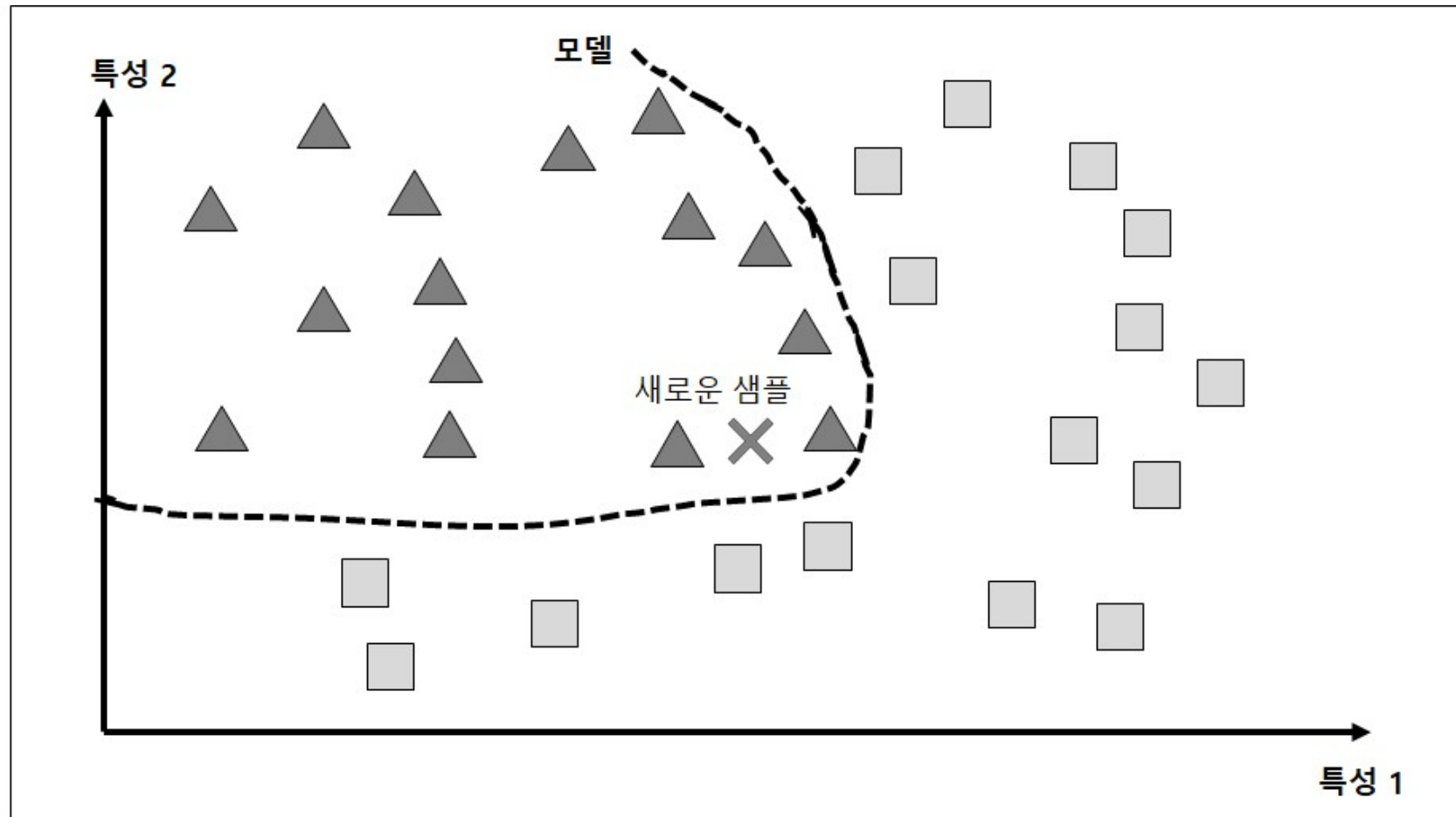


그림 1-16. 모델 기반 학습

주요 도전 과제

❖ 나쁜 데이터

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

- Feature selection과 feature extraction

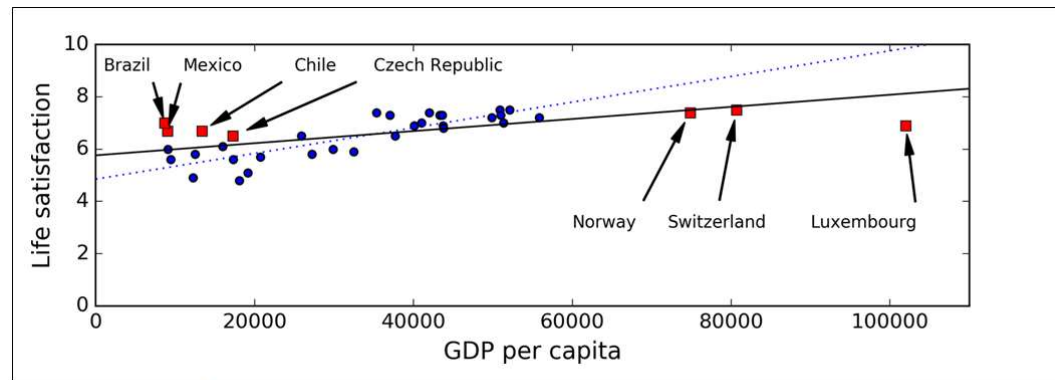


그림 1-21. 대표성이 더 큰 훈련 샘플

원인: sample noise 혹은 sampling bias

❖ 나쁜 알고리즘

- 훈련 데이터 과대적합 (overfitting)
 - 파라미터 수가 작은 모델을 사용
 - 훈련 데이터를 더 많이 모음
 - 훈련 데이터의 잡음을 제거
- 훈련 데이터 과소적합 (underfitting)
 - 파라미터가 더 많은 모델을 사용
 - 학습 알고리즘에 더 좋은 특성을 제공
 - 모델의 제약을 감소시킴

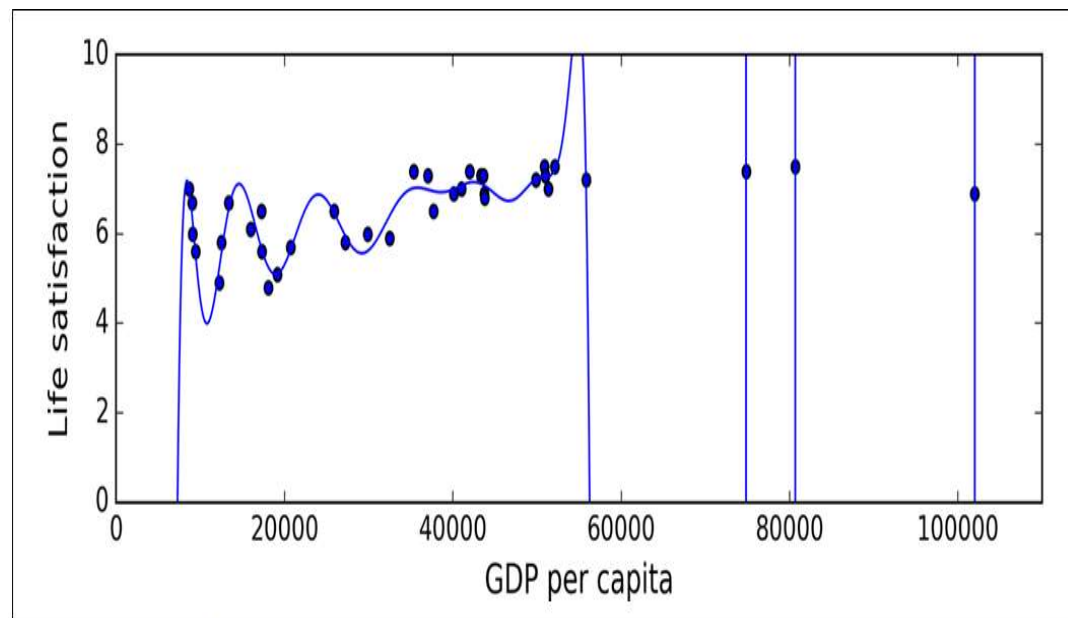


그림 1-22. 훈련 데이터에 과대적합

Q & A



데이터 다루어 보기

요약

❖ 부동산 분석 실습

- 시스템 설계
- 분석
- 데이터 가져오기
- 데이터 탐색하기
- 데이터 준비하기

❖ KDD Cup 99 데이터

- 데이터 설명
- 데이터 다운로드 하기
- 데이터 정형화하기

부동산 분석 실습



부산대학교 공과대학
전기컴퓨터공학부



부산대학교 공과대학
BIGDATA X CAMPUS

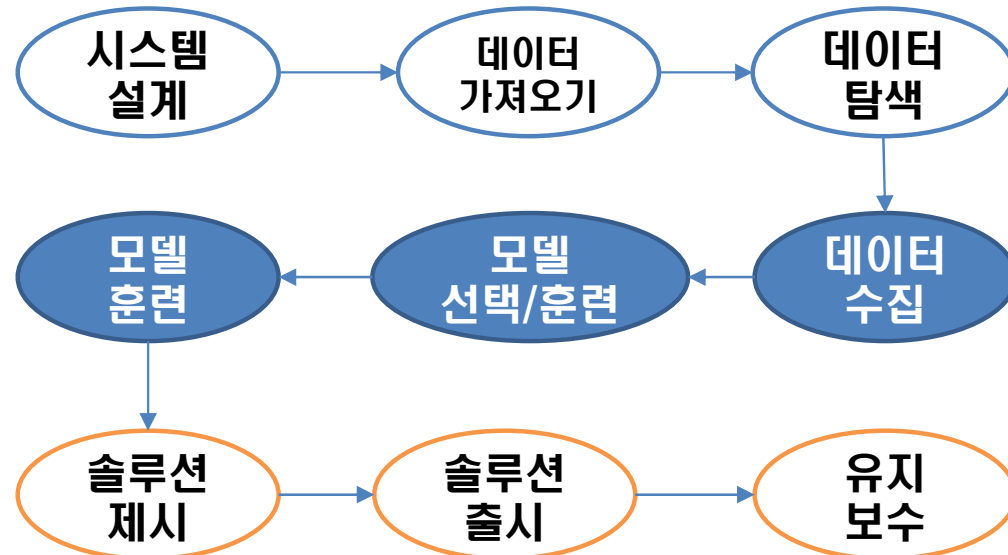


프로젝트 개요

❖ 예제 프로젝트

- 부동산 데이터 분석

❖ 절차



❖ 데이터 저장소

- 공개데이터 저장소
 - UC Irvine 머신러닝 저장소, <http://archive.ics.uci.edu/ml/>
 - Kaggle 데이터 셋, <http://www.kaggle.com/datasets>
 - Amazon AWS 데이터 셋, <http://aws.amazon.com/ko/datasets>
- 공개 데이터 나열
 - 메타 포털: <http://dataportals.org>, <http://opendatamonitor.eu>, <http://qunadl.com>
 - 위키백과 머신러닝 데이터 셋 목록, <http://goo.gl/SJHN2K>
 - 데이터셋 subreddit, <http://www.reddit.com/r/datasets>

시스템 설계

❖ 해야 할 일

- 캘리포니아 인구조사 데이터를 사용해
캘리포니아의 주택 가격 모델 생성하여, 다른 측정
데이터가 주어졌을 때 블록 그룹(block group)의
중간 주택 가격을 예측
 - 블록 그룹(block group) 혹은 구역: 600~3,000명의
인구를 포함
- 캘리포니아 인구조사 데이터
 - StatLib 저장소, <http://lib.stat.cmu.edu/datasets/>
 - 캘리포니아 주택 (California Housing) 가격,
<http://goo.gl/QgRbUL>
 - 블록 그룹(block group) 마다 인구(population), 중간
소득(median income), 중간 주택 가격(median
housing price) 등을 포함

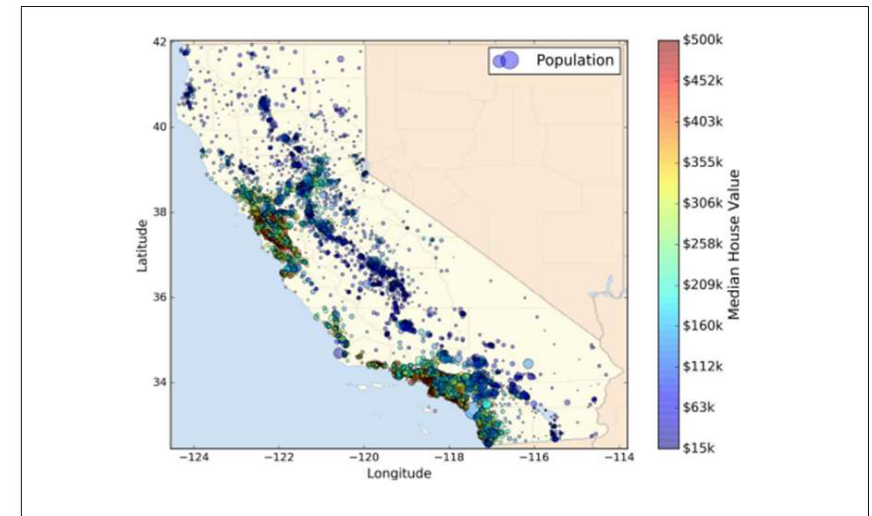


그림 2-1. 캘리포니아 주택 가격

❖ 해야 할 일

문제 정의

성능 측정 지표 선택

가정 검사

시스템 설계

❖ 문제 정의

- 데이터 분석의 목적이 무엇인가?

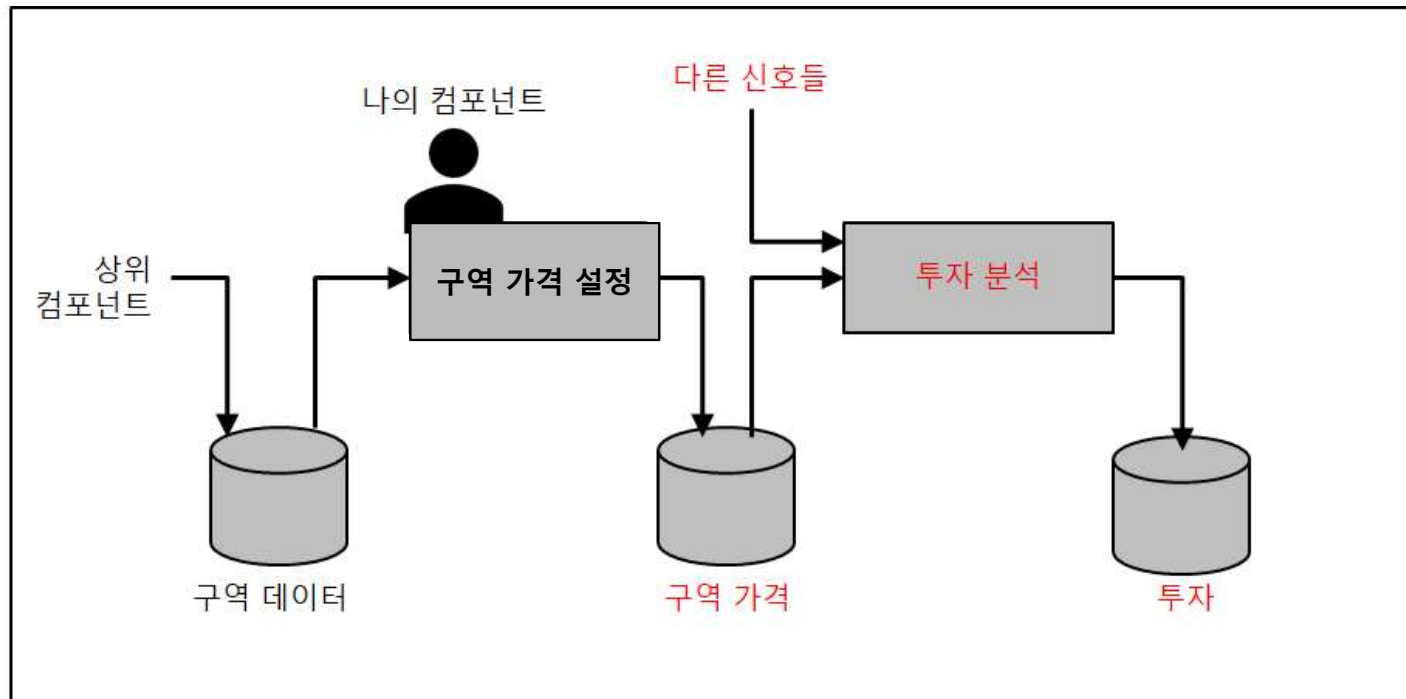


그림 2-2. 부동산 투자를 위한 머신러닝 파이프라인

- 현재 솔루션은 어떻게 구성되어 있나요?
- 어떤 작업이 필요한가?
 - 힌트: 레이블(구역의 중간 주택 가격)된 훈련 샘플 →
 - 힌트: 여러 개의 특성으로 부터 값을 예측해야 함 →

시스템 설계

❖ 성능 지표 선택

▪ RMSE(Root Mean Square Error)

- 회귀 문제의 전형적인 성능 지표

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

- m : 측정할 데이터 셋에 있는 샘플 수 (구역의 수)
- $x^{(i)}$: 데이터 셋에 있는 i 번째 샘플(레이블은 제외)의 전체 특성 값의 벡터
 - » 구역의 경도, 위도, 주민, 중간 소득
- $y^{(i)}$: 해당 레이블(해당 샘플의 기대 출력 값)
 - » 중간 주택 가격
- X : 데이터 셋에 있는 모든 샘플의 모든 특성 값(레이블은 제외)을 포함하는 행렬
- h : 시스템의 예측 함수 (hypothesis)

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

$$X = \begin{pmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{pmatrix}$$

▪ MAE(Mean Absolute Error)

- 이상치로 보이는 구역이 많은 경우

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

시스템 설계

❖ 성능 지표 선택

- 예측 값의 벡터와 타겟 값의 벡터 사이의 거리(norm)를 재는 방법
- RMSE: Euclidian norm
 - l_2 norm 혹은 $||\cdot||_2$ norm / $||\cdot||$ norm 으로 표기
- MAE: Manhattan norm
 - l_1 norm 혹은 $||\cdot||_1$ 으로 표기
- l_k norm은 $||v||_k = (|v_0|^k + |v_1|^k + \dots + |v_n|^k)^{\frac{1}{k}}$ 으로 정의함
 - l_0 norm: 벡터에 있는 0이 아닌 원소의 수
 - l_∞ norm: 벡터에서 가장 큰 절대값
- norm의 지수가 클수록 큰 값의 원소에 치우치며 작은 값은 무시된다.

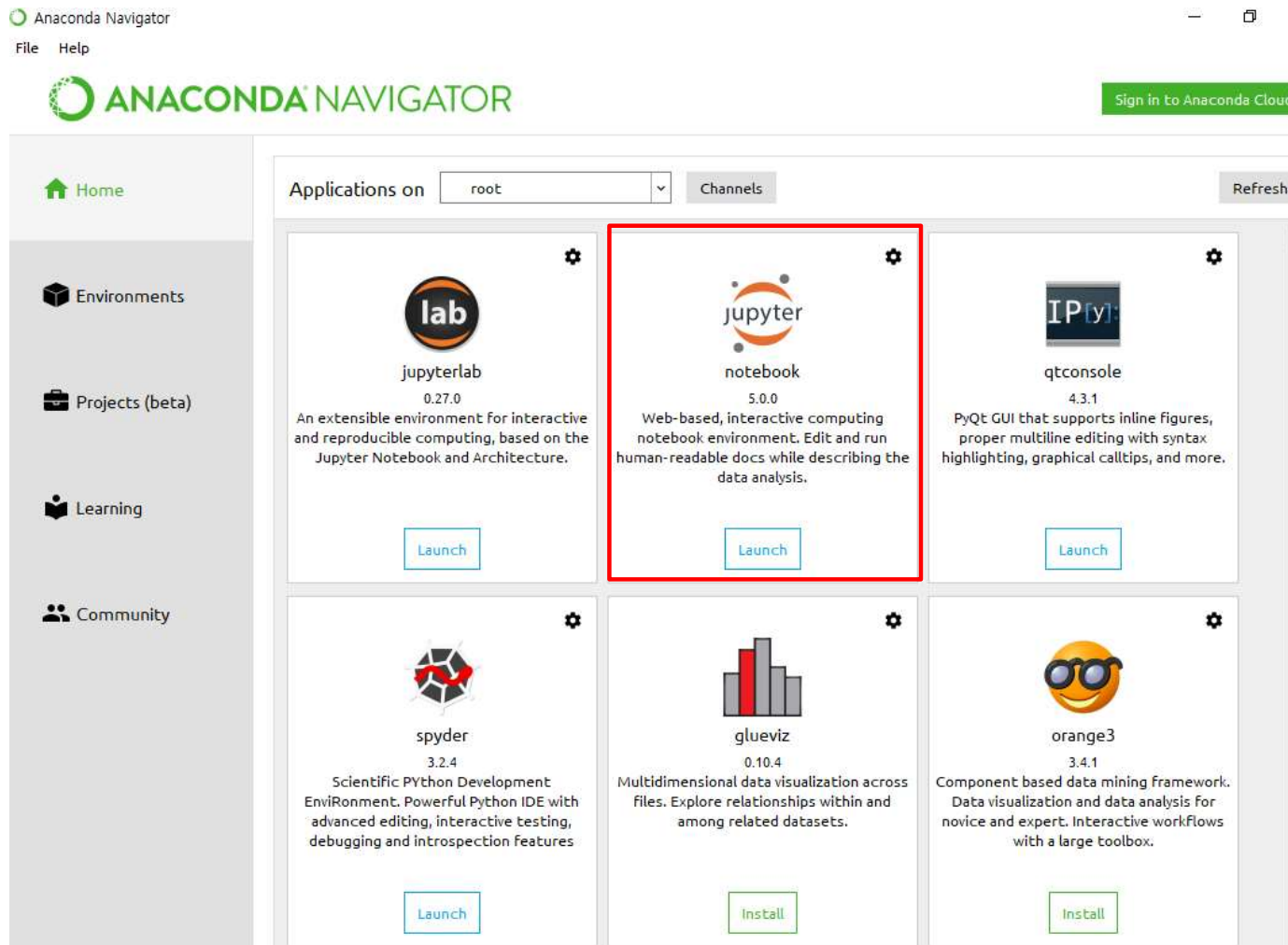
❖ 가정 검사

- 예
 - 정확한 가격을 구하는 것이 전혀 중요하지 않고, 올바른 카테고리를 구하는 시스템이 필요함
 - 회귀 시스템이 아니라 분류 시스템이 됨

실습 환경 실행

❖ Anaconda Navigator 실행하기

- 시작 – Anaconda Navigator – 선택
- Anaconda Navigator – Jupyter notebook – 선택



실습 환경 실행 (linux)

❖ 작업 환경 만들기

여러가지 버전을 사용하는 컴퓨터만

- 머신러닝 코드와 데이터셋을 저장할 작업디렉토리를 만든다.
 - `$ export ML_PATH = "$HOME/ml"`
 - `$ mkdir -p $ML_PATH`
- 독립적인 환경 구축하기
 - `$ pip install --user --upgrade virtualenv`
 - `$ cd $ML_PATH`
 - `$ virtualenv env`
 - `$ cd $ML_PATH`
 - `$ source env/bin/activate`
- 필요한 패키지와 의존성으로 연결된 패키지를 설치
 - `$ pip install --upgrade jupyter matplotlib numpy pandas scipy scikit-learn`
- 설치 확인
 - `$ python3 -c "import jupyter matplotlib numpy pandas scipy scikit-learn"`
- 주피터를 실행 및 접근
 - Jupyter notebook
 - <http://localhost:8888/> 혹은 <http://localhost:8889/>

실습 환경 실행 (windows)

❖ 작업 환경 만들기

여러가지 버전을 사용하는 컴퓨터만

- 머신러닝 코드와 데이터셋을 저장할 작업디렉토리를 만든다.

- > cd c:\temp
- > mkdir ml
- > cd ml

독자적인 환경

- > Conda create -name virtualenv python=3.5
- > conda env list
- > conda activate virtualenv

- 환경 구축하기

- > conda update -n base conda
- > conda update -all
- > python -V // python 3 version OK

삭제 : > conda remove -name virtualenv --all

- 필요한 패키지와 의존성으로 연결된 패키지를 설치

- > pip install -upgrade jupyter matplotlib numpy pandas scipy scikit-learn

- 설치 확인

- > python3 -c "import jupyter matplotlib numpy pandas scipy scikit-learn"

- 주피터를 실행 및 접근

- > conda install jupyter // --notebook-dir="C:\temp\ml"
- > jupyter notebook
- <http://localhost:8888/> 혹은 <http://localhost:8889/>

실습 환경 실행

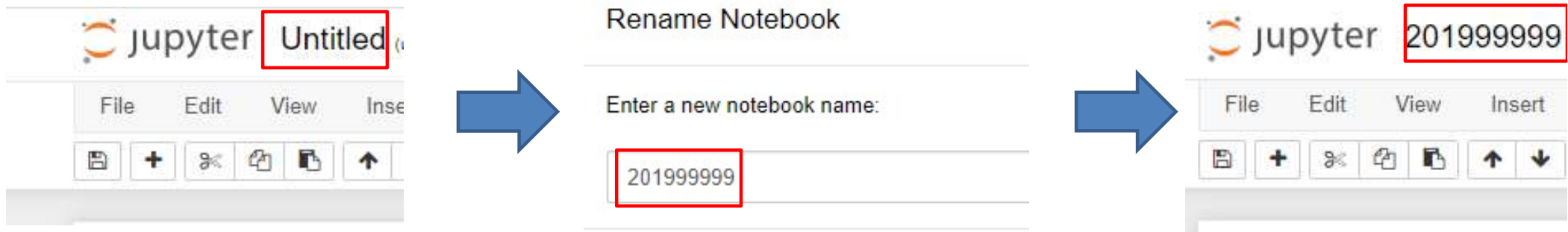
❖ Jupyter 에서 Python 3 생성하기

- New – Python3 – 선택



- 이름 변경하기

- Untitled 에서 “학번 ” 으로 변경
- 저장하기 (Cntl + S)



데이터 가져오기

❖ 데이터 다운로드

- Fetch_housing_data()를 호출하여
 - 작업 공간에 datasets/housing 디렉토리를 만들고,
 - housing.tgz 파일을 내려받아 같은 디렉토리에 압축을 풀어 housing.csv 파일을 만든다.

```
In [1]: 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()
```

-
- 판다스의 데이터프레임 객체를 반환한다.
-

```
In [2]: 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)
```

데이터 가져오기

❖ 데이터 다운로드

- DataFrame의 head() 메서드를 사용해 처음 다섯 행을 확인한다.

```
In [3]: fetch_housing_data()  
housing = load_housing_data()  
housing.head()
```

Out [3]:

	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

	A	B	C	D	E	F	G	H	I	J	K
1	longitude	latitude	housing_n	total_room	total_bedr	population	household	median_in	median_ho	ocean_proximity	
2	-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY	
3	-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY	
4	-122.24	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY	
5	-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY	

데이터 가져오기

❖ 데이터 다운로드

- info()메서드를 이용해 데이터에 대한 간략한 설명과 특히 전체 행 수, 각 특성의 데이터 타입과 널(null)이 아닌 값의 개수를 확인한다.

In [4] : housing.info()

Out [4] : <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

경도, 위도, 주택 중위 연령, 총 객실 수,
총 침실 수, 인구, 가구 수, 중위 소득, 중
위 집 가격, 바닷가와의 근접성

데이터 가져오기

❖ 데이터 다운로드

- ocean_proximity 분석: 범주형(categorical)
 - 어떤 카테고리가 있고 각 카테고리마다 얼마나 많은 구역이 있는지 value_counts() 메서드로 확인

```
In [5]: housing["ocean_proximity"].value_counts()
```

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

- describe() 메서드를 이용한 숫자형 특성의 요약 정보 출력

```
In [6]: housing.describe()
```

```
Out [6]:
```

	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

PERCENTILE



데이터 가져오기

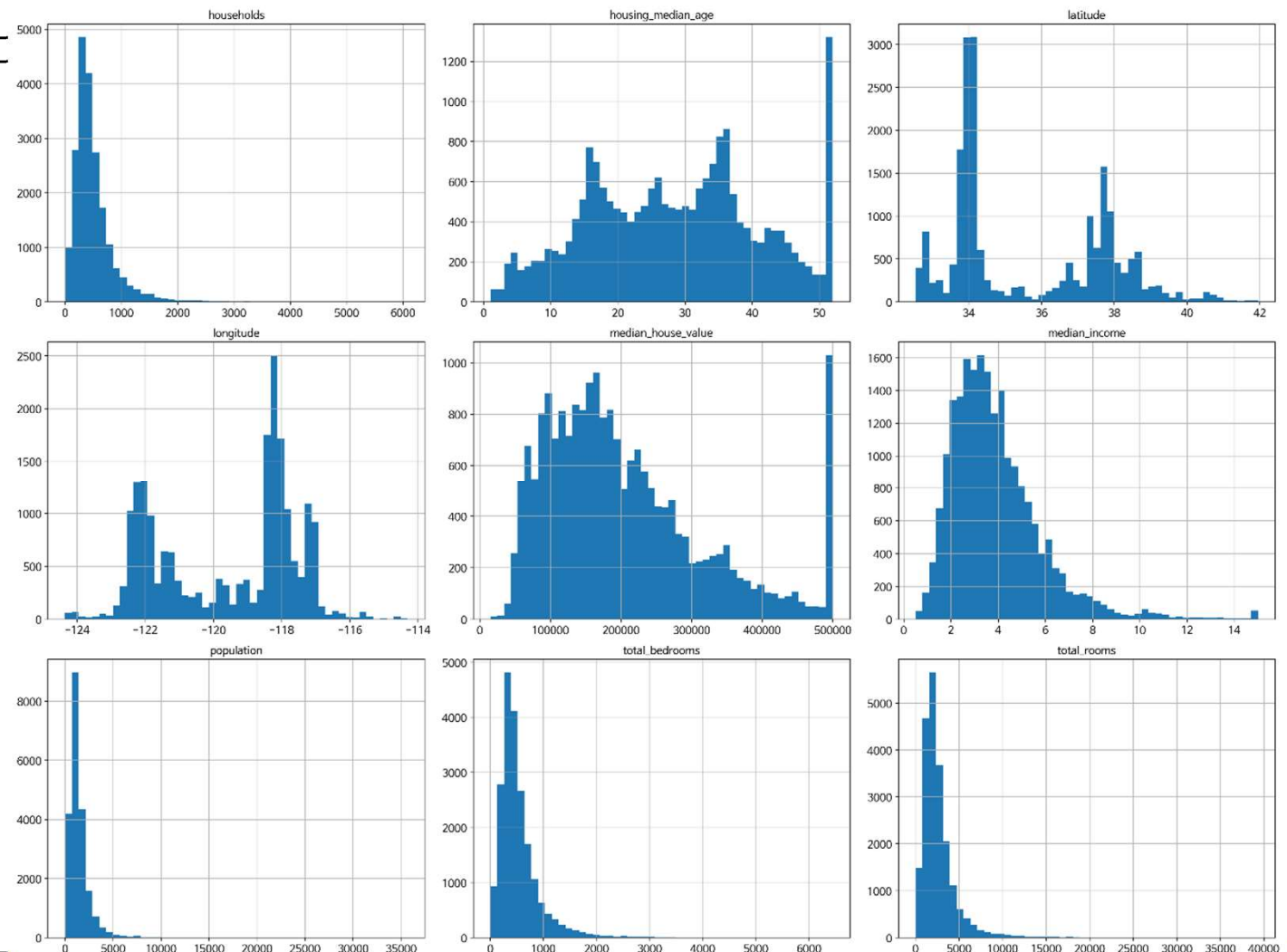
❖ 데이터 다운로드

- hist() 메소드를 활용하여 각 숫자형 특성을 히스토그램으로 그려보기

```
In [7]: %matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

Out [7]:

```
%matplotlib inline
%matplotlib tk
```



데이터 가져오기

❖ 테스트 세트 만들기

- 방법 1: 어떤 샘플을 선택해서 데이터 셋의 20%정도를 테스트 세트로 사용

```
In [8]: import numpy as np
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]

train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
```

Out [8]: 16512 train + 4128 test

← 단점? : 실행할 때마다 데이터를 분할하는 기준이 달라짐

- 해결책
 - 처음 실행에서 테스트 세트를 저장하고 다음번 실행에서 이를 불러들여 사용함
 - 항상 같은 난수 인덱스가 생성되도록 np.random.permutation()을 호출하기 전에 초깃값을 지정
 - 샘플의 식별자를 사용하여 테스트 세트로 보낼지 말지 정함

데이터 가져오기

❖ 테스트 세트 만들기

- 행의 인덱스를 id로 사용

```
In [9]: from zlib import crc32
def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

```
In [10]: housing_with_id = housing.reset_index()
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

```
>>> df = pd.DataFrame([('bird', 389.0),
...                    ('bird', 24.0),
...                    ('mammal', 80.5),
...                    ('mammal', np.nan)],
...                    index=['falcon', 'parrot', 'lion', 'monkey'],
...                    columns=('class', 'max_speed'))
```

```
>>> df
   class  max_speed
falcon  bird    389.0
parrot  bird     24.0
lion    mammal    80.5
monkey  mammal     NaN
```

```
>>> df.reset_index()
   index  class  max_speed
0  falcon  bird    389.0
1  parrot  bird     24.0
2    lion  mammal    80.5
3  monkey  mammal     NaN
```

```
>>> df = pd.DataFrame([[1, 2], [4, 5], [7, 8]],
...                    index=['cobra', 'viper', 'sidewinder'],
...                    columns=['max_speed', 'shield'])
```

```
>>> df
   max_speed  shield
cobra         1      2
viper         4      5
sidewinder    7      8
```

```
>>> df.loc['viper']
max_speed    4
shield       5
Name: viper, dtype: int64
```

데이터 가져오기

❖ 테스트 세트 만들기

▪ 방법 2: 계층적 샘플링 (stratified sampling)

- 전체 모수는 계층(strata)이라는 동질의 그룹으로 나누고, 테스트 세트가 전체 모수를 대표하도록 각 계층에서 올바른 수의 샘플을 추출합니다.
- 계층별로 데이터셋에 충분한 샘플 수가 있어야 한다.

▪ 예제

- 중간 소득을 1.5로 나누고, ceil 함수를 사용하여 반올림해서 소득 카테고리 특성을 만들고, 5보다 큰 카테고리는 5로 합칩니다.

```
In [11]: housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

- 사이킷런의 StratifiedShuffleSplit을 사용하여 소득 카테고리를 기반으로 계층 샘플링을 수행

```
In [12]: 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]
```

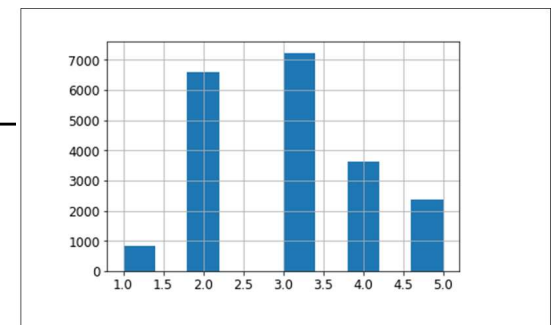


그림 2-9. 수입 카테고리의 히스토그램

데이터 가져오기

❖ 테스트 세트 만들기

- 방법 2: 계층적 샘플링 (stratified sampling)
 - 전체 주택 데이터셋에서 소득 카테고리의 비율

```
In [13]: housing["income_cat"].value_counts() / len(housing)
```

```
Out [13]: 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
```

- income_cat 특성을 삭제해서 데이터를 원래 상태로 되돌림

```
In [14]: for set_ in (strat_train_set, strat_test_set):
          set_.drop("income_cat", axis=1, inplace=True)
```

데이터 탐색

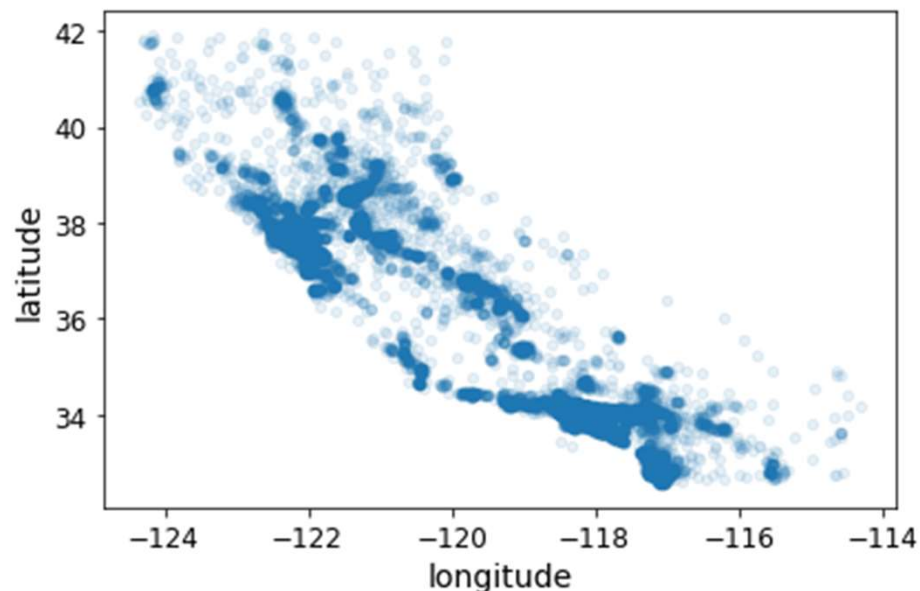
❖ 훈련 세트를 손상시키지 않기 위해 복사본을 생성

```
In [15]: housing = strat_train_set.copy()
```

❖ 지로정보(위도와 경도)가 있으니 모든 구역을 산점도로 만들어 데이터를 시각화 (밀집도)

```
In [16]: %matplotlib inline  
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

Out [16]: <matplotlib.axes._subplots.AxesSubplot at 0x11446ae10>



데이터 탐색

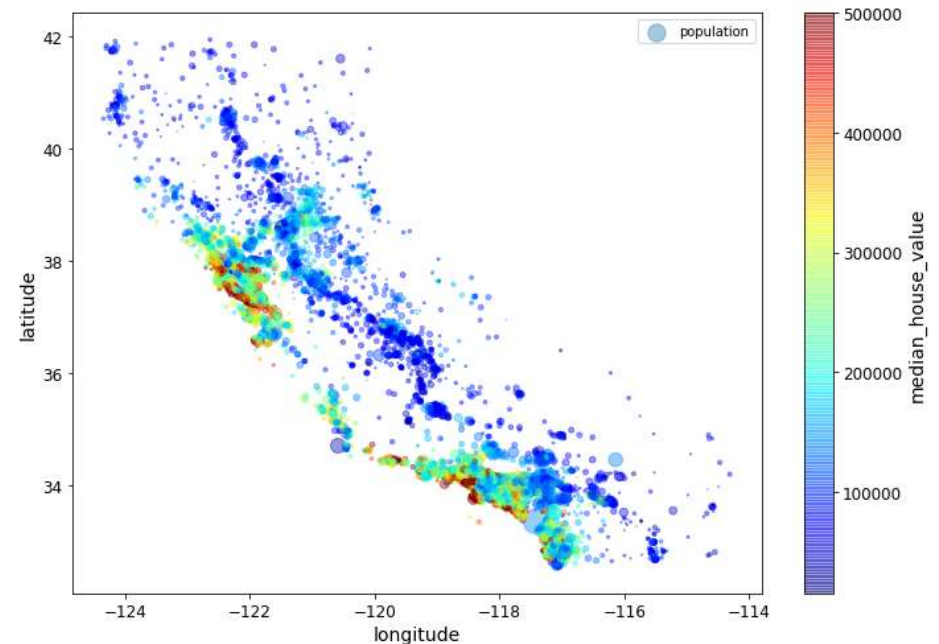
❖ 주택 가격 표시

- 원의 반지름: 구역의 인구 (매개변수 s)
- 색깔: 가격 (매개변수 c)
 - 컬러 맵: jet을 사용. 파란색 – 낮은 가격, 빨간색 – 높은 가격 (매개변수 cmap)

```
In [18]: 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()
```

Out [18]: <matplotlib.legend.Legend at 0x1145a52e8>



데이터 탐색

❖ 상관 관계 조사

- 방법 1: 모든 특성 간의 표준 상관관계수(standard correlation coefficient: 피어슨 r)를 `corr()` 메서드를 이용하여 쉽게 계산, $-1 \leq r \leq 1$
- 중간 주택 가격과 다른 특성 사이의 상관관계 크기

```
In [19]: corr_matrix = housing.corr()
         corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
Out [19]: 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
```

데이터 탐색

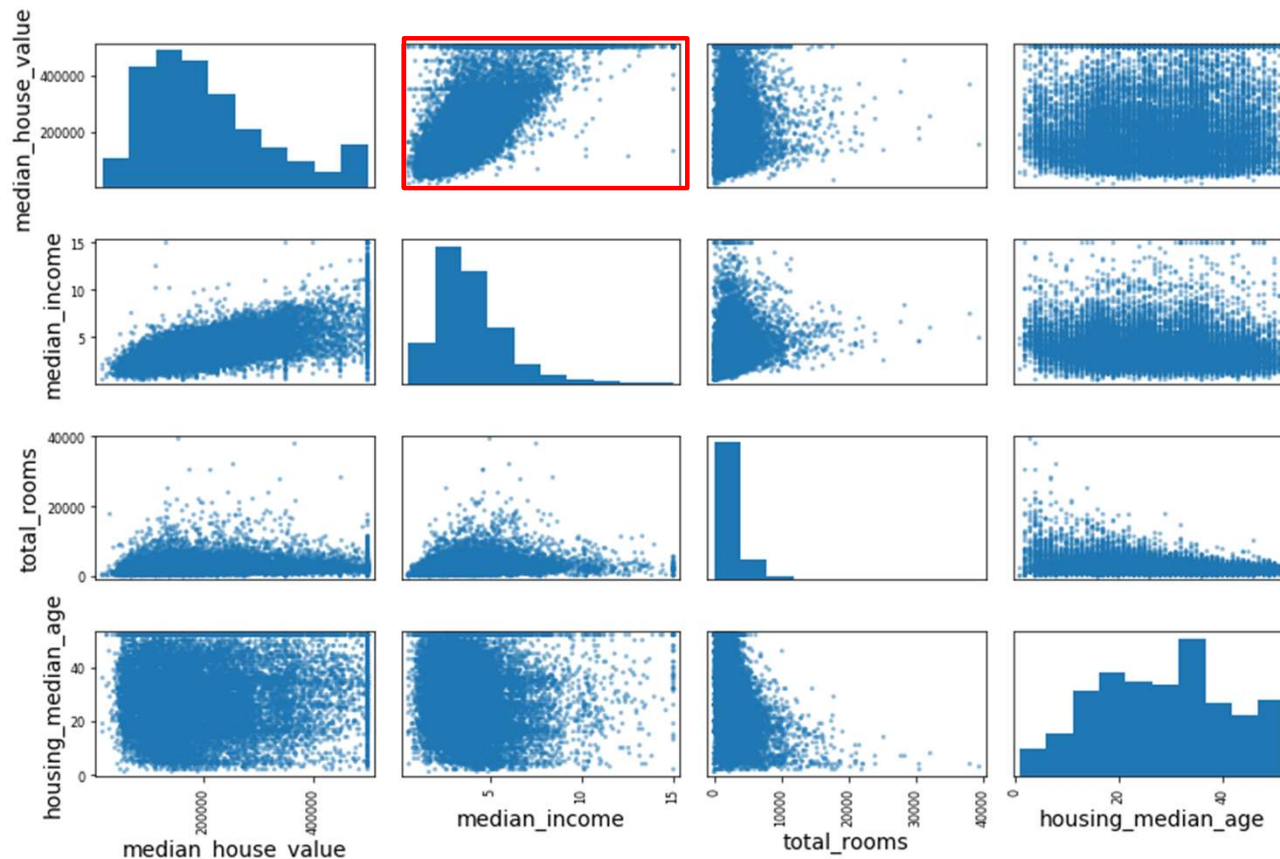
❖ 상관 관계 조사

- 방법 2: 숫자형 특성 사이에 산점도를 그려주는 판다스의 scatter_matrix 함수를 사용

In [20]: `from pandas.plotting import scatter_matrix`

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

Out [20]:



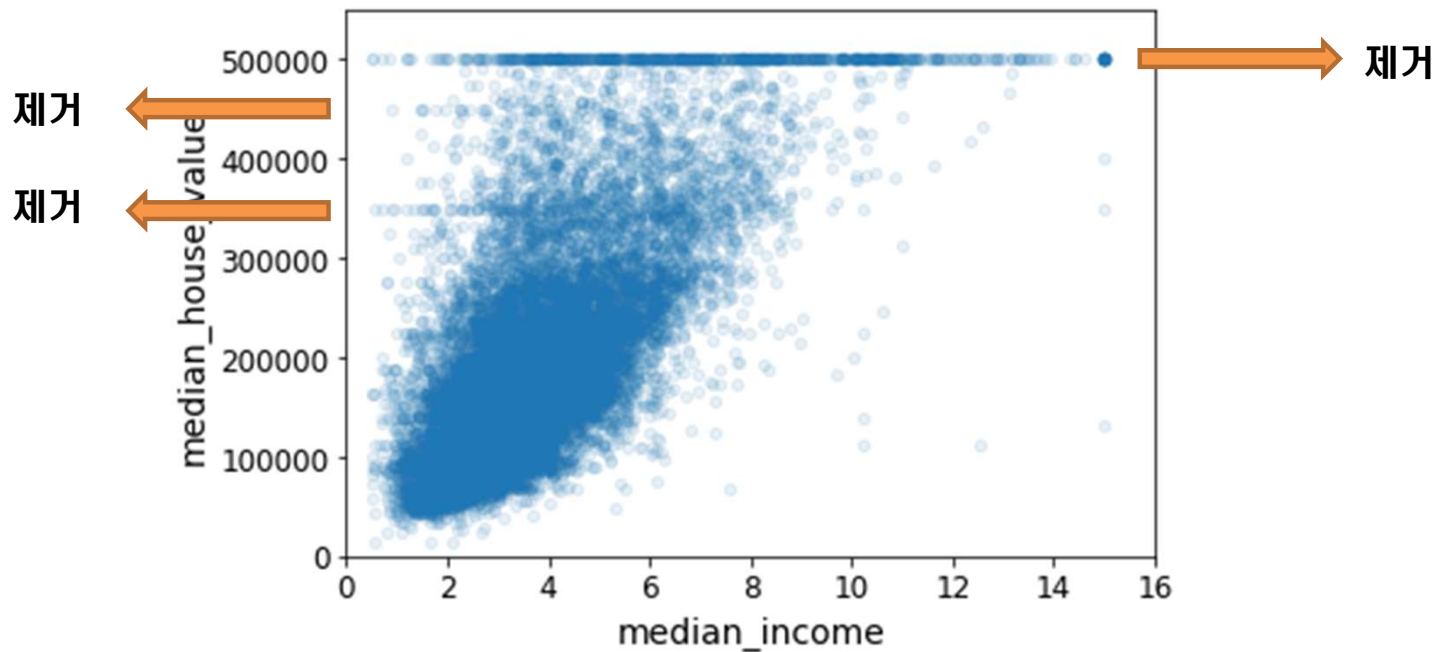
데이터 탐색

❖ 상관 관계 조사

- 중간 주택 가격을 예측하는데 가장 유용할 것 같은 특성을 갖는 것으로 나타난 중간 소득과의 상관관계 산점도를 확대 표시

```
In [21]: housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
```

Out [21]:



데이터 탐색

❖ 여러 특성의 조합을 시도

- 예를 들어, 특정 구역의 방 개수는 얼마나 많은 가구수가 있는지 모른다면 그다지 유용하지 않음 → 가구당 방 개수를 계산

```
In [22]: 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"]
```

```
In [23]: corr_matrix = housing.corr()  
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
Out [23]: 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
```

데이터 준비

- ❖ 원래 훈련 세트로 복원하고(strat_train_set을 복사), 예측 변수와 타깃 값에 같은 변형을 적용하지 않기 위해 예측 변수와 레이블을 분리

```
In [24]: housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

❖ 데이터 정제

- 누락된 특성을 다루기 위한 함수 사용
 - 해당 구역을 제거 → dropna()
 - 전체 특성을 삭제 → drop()
 - 어떤 값으로 채움 (0, 평균, 중간 값 등) → fillna()

```
In [25]: sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows

sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # 선택 사항 1

sample_incomplete_rows.drop("total_bedrooms", axis=1)      # 선택 사항 2

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

데이터 준비

❖ 데이터 정제

- 사이킷런의 Imputer 사용

- 누락된 값을 특성의 중간 값으로 대체하여 지정하고 Imputer 객체를 생성

```
In [26]: from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(strategy="median")
```

- 텍스트 특성인 ocean_proximity를 제외한 데이터 복사본을 생성

```
In [27]: housing_num = housing.drop("ocean_proximity", axis=1)
```

- Imputer 객체의 fit() 메서드를 사용해 훈련 데이터에 적용
 - Imputer는 각 특성의 중간 값을 계산해서 그 결과를 객체의 statistics_ 속성에 저장

```
In [28]: imputer.fit(housing_num)
imputer.statistics_
housing_num.median().values
```

```
Out [28]: array([ -118.51 ,   34.26 ,    29.   ,  2119.5 ,   433.   ,   1164.   ,   408.   ,   3.5409])
array([ -118.51 ,   34.26 ,    29.   ,  2119.5 ,   433.   ,   1164.   ,   408.   ,   3.5409])
```

- 훈련 세트에서 누락된 값을 학습한 중간값으로 변경
 - 판다스 데이터프레임으로 변경

```
In [30]: X = imputer.transform(housing_num) # 변형된 특성들이 있는 NumPy 배열
```

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

데이터 준비

❖ 텍스트와 범주형 특성 다루기

- 방법 1: factorize() 메소드를 사용하여 텍스트에서 숫자로 변경

```
In [32]: housing_cat = housing["ocean_proximity"]  
housing_cat_encoded, housing_categories = housing_cat.factorize()  
housing_cat_encoded[:10]
```

```
Out [32]: array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0])
```

- 카테고리 리스트

```
In [33]: housing_categories
```

```
Out [33]: Index(['<1H OCEAN', 'NEAR OCEAN', 'INLAND', 'NEAR BAY', 'ISLAND'], dtype='object')
```

- 문제점? : 값이 가까우면 비슷한 값이라고 판단함
- 해결책: 카테고리별 이진 특성을 만들 → one-hot encoding

```
In [34]: from sklearn.preprocessing import OneHotEncoder  
  
encoder = OneHotEncoder()  
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))  
housing_cat_1hot
```

데이터 준비

❖ 텍스트와 범주형 특성 다루기

- 방법 1: `factorize()` 메소드를 사용하여 텍스트에서 숫자로 변경
 - 문제점?
 - `fit_transform()` 메서드는 2차원 배열을 사용하지만 `housing_cat_encoded`는 1차원 배열을 사용
 - 출력이 SciPy 희소 행렬이므로, 메모리 낭비를 초래
 - 해결책
 - [밀집된] `toarray()` 메소드를 호출하여 NumPy 배열로 바꿈

```
In [35]: housing_cat_1hot.toarray()
```

```
Out [35]: 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.]])
```

데이터 준비

❖ 특성 스케일링 (feature scaling)

- 머신러닝 알고리즘은 입력 숫자 특성들의 스케일이 많이 다르면 잘 작동하지 않음
- min-max 스케일링(normalization)과 표준화(standardization)를 사용
 - 훈련 데이터에서는 fit() 메소드를 적용 → 추정기
 - 훈련 및 테스트 데이터에서는 transform() 메소드를 적용 → 변환기
- min-max 스케일링(normalization)
 - 0~1 범위에 들도록 값을 이동하고 스케일을 조정
 - 데이터에서 최소값을 뺀 후 최댓값과 최솟값의 차이로 나눔
 - SciKit-Learn 에는 MinMaxScalar 변환기를 제공하며, feature_range 매개변수로 범위를 변경
- 표준화(standardization)
 - 평균을 뺀 후 표준편차로 나누어 결과 분포의 분산이 1이 되도록 함
 - 범위의 상한과 하한이 없어 신경망과 같은 알고리즘에서는 문제가 될 수 있음
 - 이상치에 영향을 덜 받음
 - SciKit-Learn 에는 StandardScalar 변환기를 제공함

데이터 준비

❖ 특성 스케일링 (feature scaling)

- min-max 스케일링

```
In [35] : from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
housing_num = housing.drop('ocean_proximity', axis=1)  
scaler.fit(housing_num)
```

```
print(scaler.data_max_)
```

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

```
Out [35] : array([[0.24501992, 0.50478215, 0.7254902 , ..., 0.01981558, 0.06292009, 0.15201859],  
 [0.24103586, 0.47927736, 0.25490196, ..., 0.00849239, 0.02072442, 0.40837368],  
 [0.71215139, 0.02444208, 0.58823529, ..., 0.02614984, 0.08588499, 0.1629081 ],  
 ...,  
 [0.79183267, 0.16471838, 0.15686275, ..., 0.05871801, 0.14245706, 0.19119736],  
 [0.6314741 , 0.1360255 , 0.58823529, ..., 0.03792147, 0.0660941 , 0.24569316],  
 [0.18924303, 0.55579171, 1. , ..., 0.03548306, 0.11893204, 0.21207294]])
```

KDD Cup 99 데이터



부산대학교 공과대학
전기컴퓨터공학부



부산대학교 공과대학
BIGDATA X CAMPUS

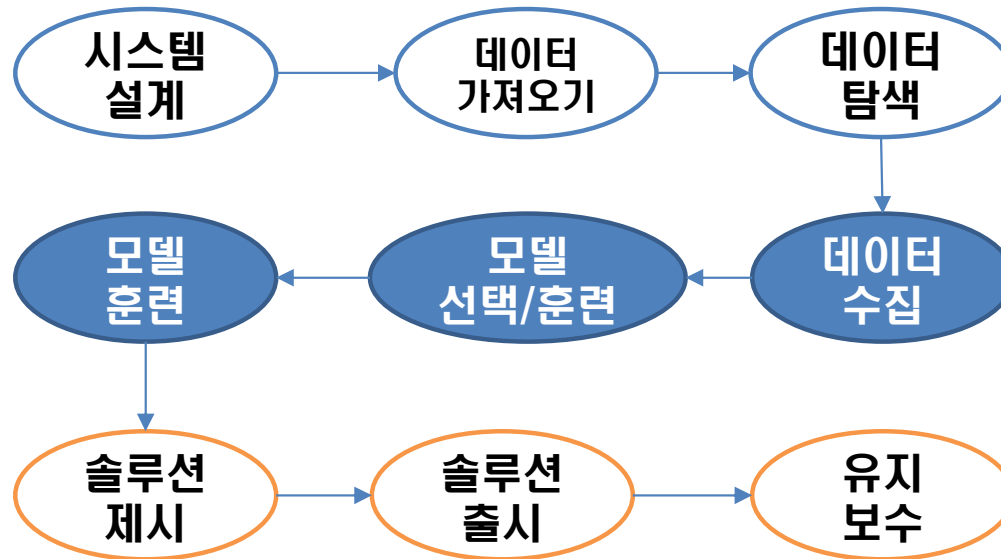


KDD cup 1999

❖ 예제 프로젝트

- KDD cup 99 데이터 분석

❖ 절차



❖ 데이터 저장소

- KDD Cup 1999 Data
 - <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>

❖ 해야 할 일



KDD cup 1999

❖ 소개

- 네트워크 침입 탐지 시스템은 내부자를 포함하여 권한이 없는 사용자로부터 컴퓨터 네트워크를 보호함
- 침입 탐지 시스템의 작업은 정상 연결과 비정상 연결을 구분함
- 패킷 등을 요약한 데이터를 이용해서 네트워크 침입을 감지하는 탐지 시스템을 구현하는 대회
- 공격은 4가지 대분류와 24가지 세분류 로 나뉨
 - 서비스 거부 공격(DoS)
 - 원격 무단 액세스(R2L)
 - 로컬 무단 액세스(U2R)
 - 감시 및 기타 프로빙(Probing)



Knowledge Discovery and Data Mining

Attack class	Attack type
Dos	back, land, neptune, pod, smurf, teardrop
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster
U2R	buffer_overflow, loadmodule, perl, rootkit
Probe	ipsweep, nmap, portsweep, satan

KDD cup 1999

❖ 데이터 속성(42개)

KDD Cup 1999 Data의 task description 참조

- 독립 TCP 연결의 기본 속성(9개)
 - Duration, Protocol_type, Service, src_bytes, dst_bytes, Flag, Land, Wrong_fragment, Urgent
- 도메인 지식으로 제안된 연결 내의 콘텐츠 기능(13개)
 - hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_hot_login, is_guest_login
- 2초 시간, Window를 사용해서 계산한 트래픽 기능(9개) X 2 – normal / host
 - count, serror_rate, error_rate, same_srv_rate, diff_srv_rate, srv_count, srv_serror_rate, srv_error_rate, srv_diff_host_rate
- 기타 (2개)
 - dst_host_same_src_port_rate
 - 해당 데이터의 레이블

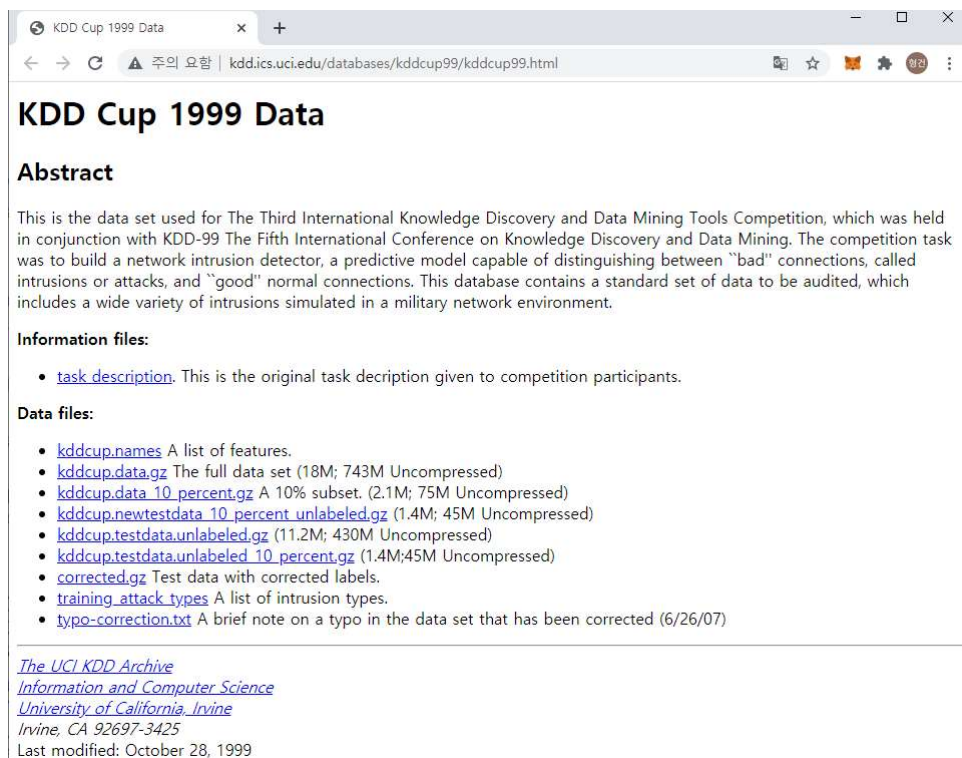
0,tcp,http,RSTR,45908,7300,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.50,0.50,1.00,0.00,0.00,2,2,1.00,0.00,0.50,0.00,0.00,0.00,0.50,0.50,back.

KDD cup 1999

❖ 해야 할 일

▪ 데이터 다운로드 하기

- KDD Cup 1999 Data 접속
- Data files 에서 kddcup.data.gz, kddcup.data_10_percent.gz 선택 및 다운로드



The screenshot shows a web browser window with the URL kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. The page title is "KDD Cup 1999 Data". Under the "Abstract" section, it describes the data set used for the Third International Knowledge Discovery and Data Mining Tools Competition. The "Information files:" section lists the [task description](#). The "Data files:" section lists several files, with [kddcup.data.gz](#) and [kddcup.data_10_percent.gz](#) highlighted by a red box. At the bottom, it mentions "The UCI KDD Archive" and "Last modified: October 28, 1999".

KDD Cup 1999 Data

Abstract

This is the data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining. The competition task was to build a network intrusion detector, a predictive model capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

Information files:

- [task description](#). This is the original task description given to competition participants.

Data files:

- [kddcup.names](#) A list of features.
- [kddcup.data.gz](#) The full data set (18M; 743M Uncompressed)
- [kddcup.data_10_percent.gz](#) A 10% subset. (2.1M; 75M Uncompressed)
- [kddcup.newtestdata_10_percent_unlabeled.gz](#) (1.4M; 45M Uncompressed)
- [kddcup.testdata.unlabeled.gz](#) (11.2M; 430M Uncompressed)
- [kddcup.testdata.unlabeled_10_percent.gz](#) (1.4M; 45M Uncompressed)
- [corrected.gz](#) Test data with corrected labels.
- [training_attack_types](#) A list of intrusion types.
- [typo-correction.txt](#) A brief note on a typo in the data set that has been corrected (6/26/07)

The UCI KDD Archive
Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
Last modified: October 28, 1999

Information files:

- [task description](#). This is the original task description given

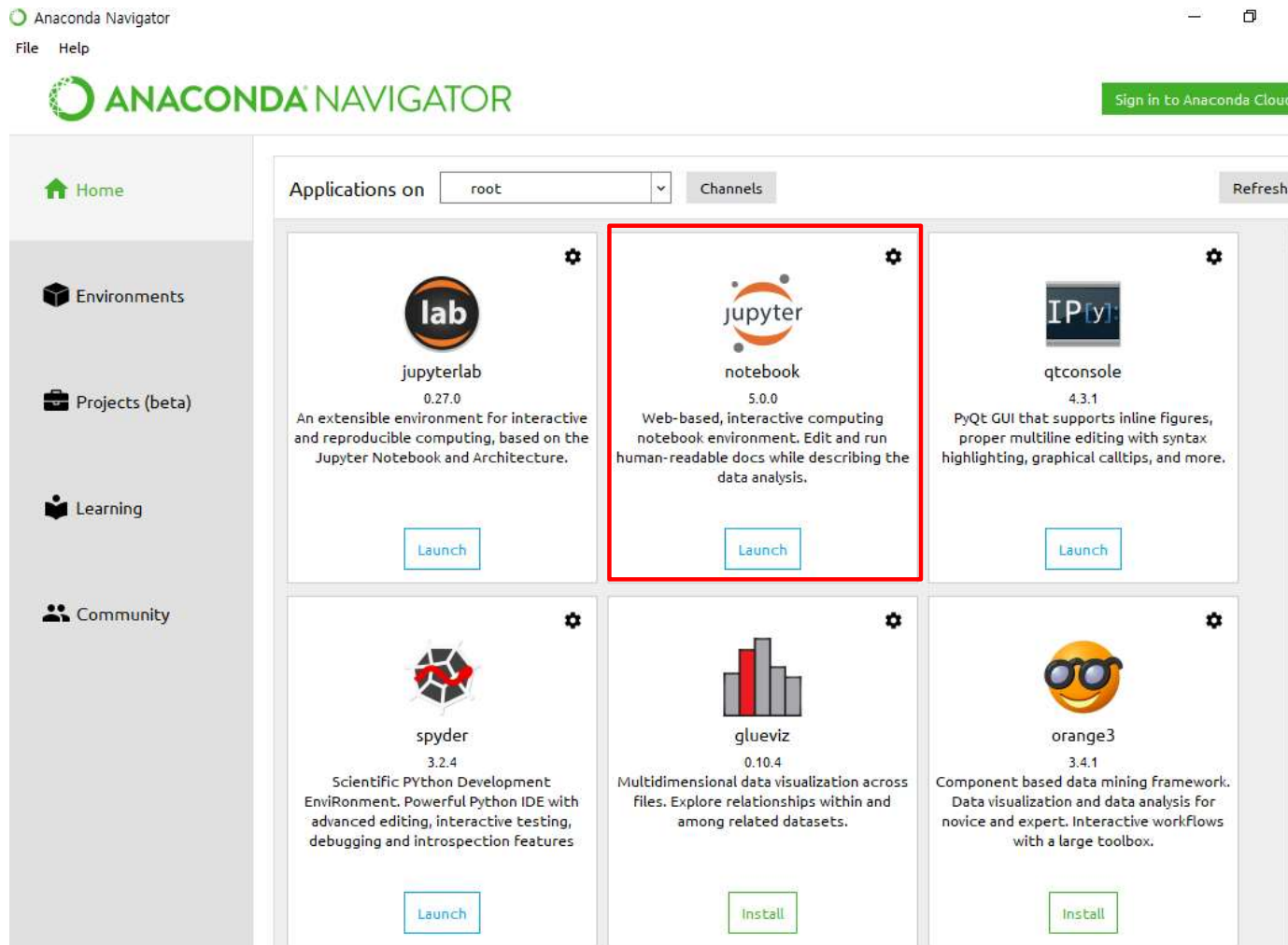
Data files:

- [kddcup.names](#) A list of features.
- [kddcup.data.gz](#) The full data set (18M; 743M Uncompressed)
- [kddcup.data_10_percent.gz](#) A 10% subset. (2.1M; 75M Uncompressed)
- [kddcup.newtestdata_10_percent_unlabeled.gz](#) (1.4M; 45M Uncompressed)
- [kddcup.testdata.unlabeled.gz](#) (11.2M; 430M Uncompressed)
- [kddcup.testdata.unlabeled_10_percent.gz](#) (1.4M; 45M Uncompressed)
- [corrected.gz](#) Test data with corrected labels.
- [training_attack_types](#) A list of intrusion types.
- [typo-correction.txt](#) A brief note on a typo in the data set

실습 환경 실행

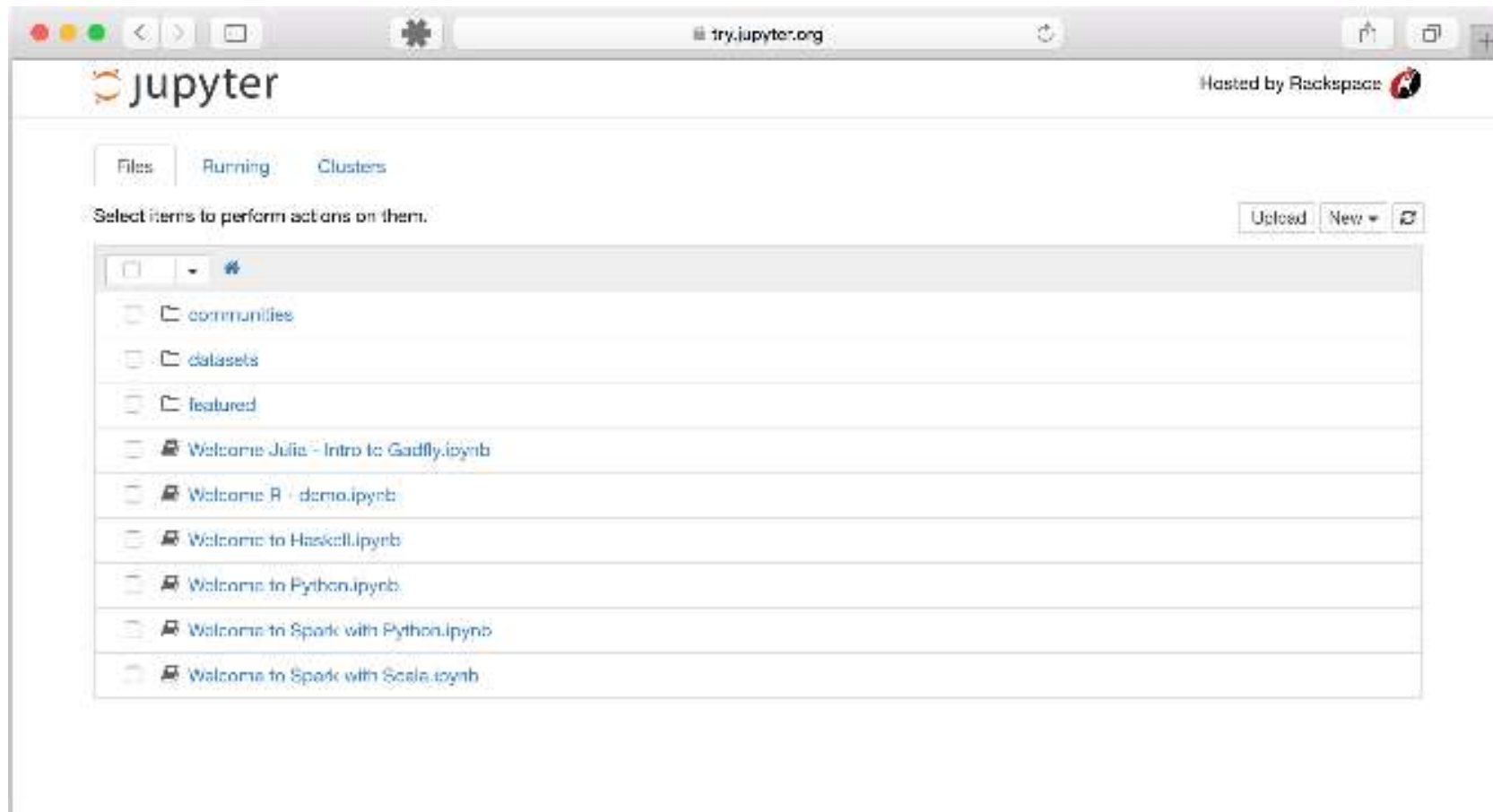
❖ Anaconda Navigator 실행하기

- 시작 – Anaconda Navigator – 선택
- Anaconda Navigator – Jupyter notebook – 선택



실습 환경 실행

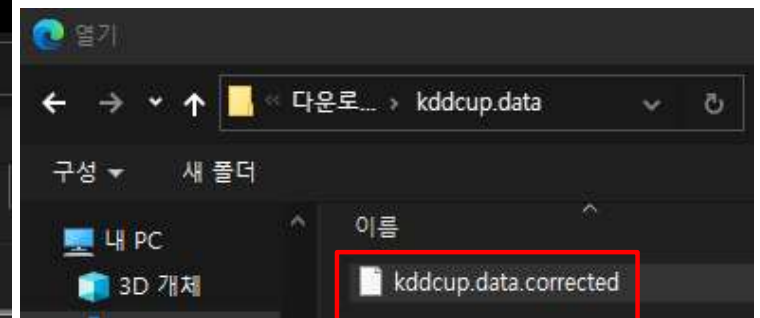
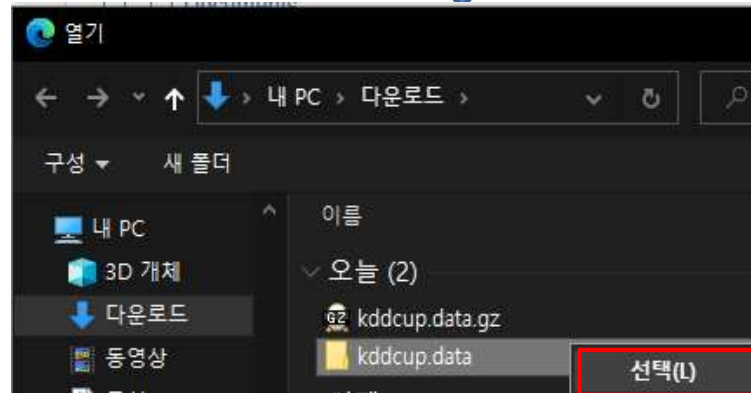
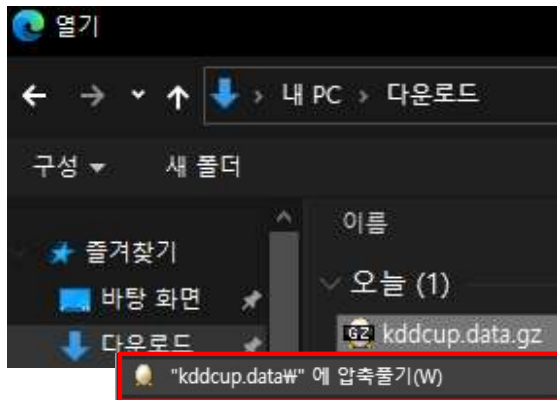
- ❖ 정상적으로 Notebook 서버를 시작하면 웹 응용 프로그램의 URL [기본적으로 `http://localhost:8889/tree`]을 포함하여 노트북 서버에 대한 정보가 인쇄되면서 다음과 같은 Notebook Dashboard가 열림



실습 환경 실행

❖ 데이터 가져오기

- kddcup.data.gz, kddcup.data_10_percednt.gz



Large file size warning

The file size is 708 MB. Do you still want to upload it?

Cancel

Ok

kddcup.data.corrected

Upload

Cancel

실습 진행

❖ 데이터 정형화 하기

- `kddcup.data.gz`, `kddcup.data_10_percednt.gz`
- `info()`, `describe()`, `corr()` 함수들을 이용한 데이터 파악
- 데이터 정형화 수행
 - 범주형
 - 정형화
 - 표준화

Q & A