AI
WITH PYTHON

PART I

ABOUT AI

ABOUT AI

◆ AI 역사

- **1955년** 존 매카시 논문 <지능이 있는 기계를 만들기 위한 과학과 공학> 처음 등장
- **1956년 다스머스 학회에서 공식적으로 인공지능**이라는 **용어** 사용
- 뇌를 모사한 인공지능을 뜻하는 **퍼셉트론(Perceptron)** 용어도 탄생
- 1960년에 접어들면서 인공지능 연구가 본격화
- 1970년대 초까지 인간처럼 생각하고 문제를 푸는 인공지능 연구 계속
- 1970~1980년대 한계
- 1980~1990년대 신경망, 다층 인식론 등 연구
- **2000년대 머신러닝, 딥러닝**

ABOUT AI

◆ AI 발전 역사

* 출처 : I-Korea 4.0 실현을 위한 인공지능 R&D 전략 (과기정통부, 2018.5)
* 본 그림은 2018.5 현재까지의 인공지능분야 주요 결과물과 향후 2030년까지의 주요 기술동향을 함께 표시함

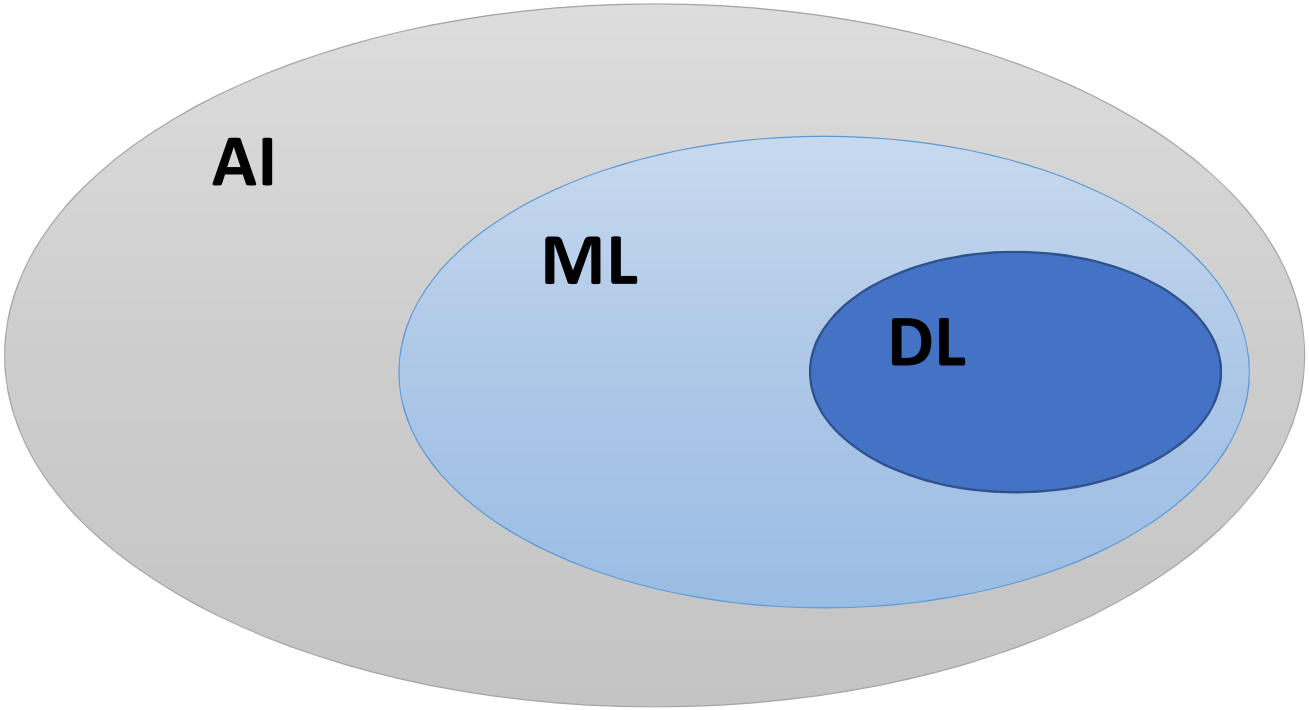
ABOUT AI

◆ AI 발전 역사

구분	1세대	2세대	3세대	4세대
시기	1950~1980년대	1990년대	2000년 대	2010년 대 ~ 현재
특징	제어 프로그램	경로 및 DB 탐색	**머신러닝**	**딥러닝**
내용	- 기계 및 가전제품에 탐재된 단순 제어 프로그램	-대량 정보와 규칙기반 경로 탐색 -DB검색 후 정답 파악 -**전문가 시스템**	-컴퓨터 스스로 규칙 및 지식 학습 -예측 방법 파악 -인공시경막	-추상화된 특징 표현 등 고급 지식 학습 -데이터 변형 및 인사이트 파악 -깊은 인공신경막
예시	자동 세탁기	검색 DB	문자 및 패턴인식	영상 및 음성인식 자연어 처리

ABOUT AI

◆ 인공지능 & 머신러닝 & 딥러닝 관계



ABOUT AI

◆ 머신러닝(Machine Learning)

- 인공지능의 한 분야
- **패턴 인식과 컴퓨터 학습 이론**의 연구로부터 **진화한 분야**

- 1949년 Hubbian Learning Theroy를 발표하면서 시작
- 1952년 Arthur Samuel이 **경험으로부터 배우는 방법 사용한 최초의 머신러닝 프로그램인 체커** 개발

- **경험적 데이터 기반 학습** 및 예측 수행으로 **스스로 성능 향상**시키는 **시스템과 알고리즘 연구, 구축**하는 기술
- **입력 데이터를 기반으로 예측이나 결정**을 이끌어내기 위해 **특정한 모델 구축 방식**

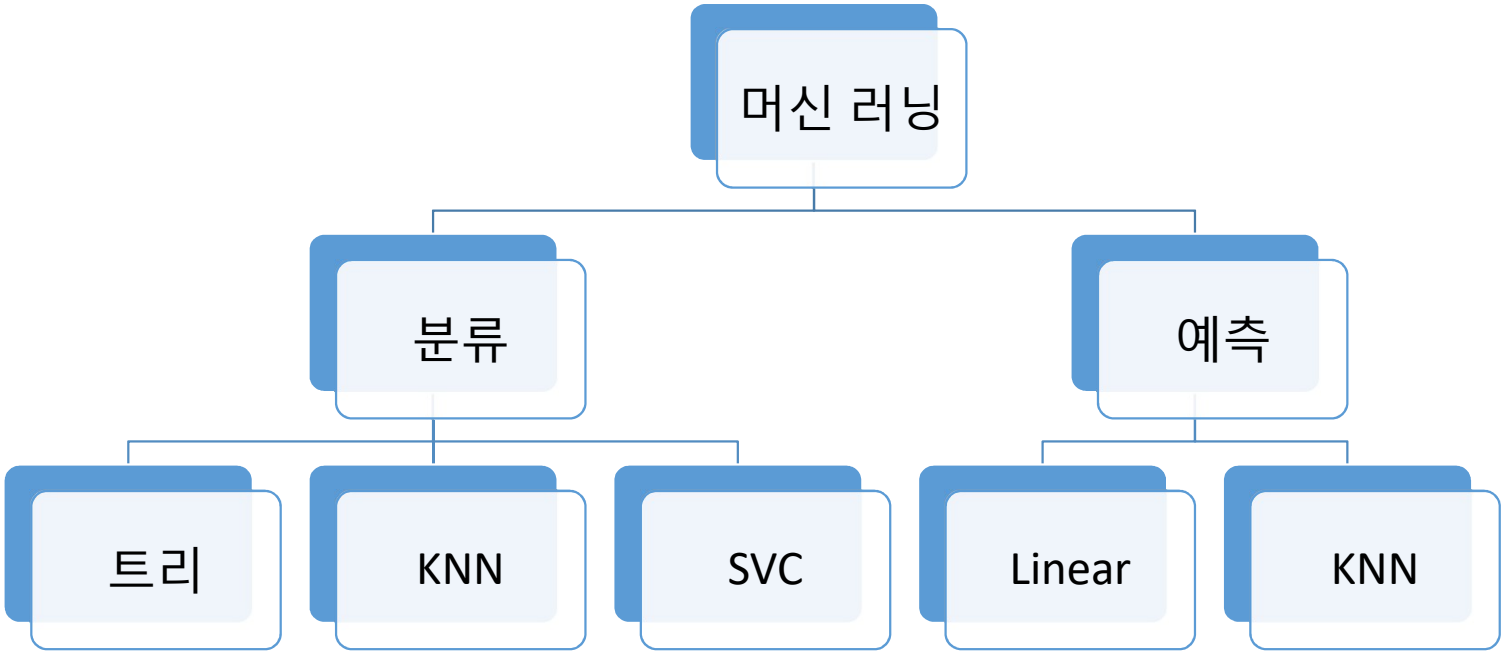
ABOUT AI

◆ 머신러닝(Machine Learning)

알고리즘	특징	
지도(supervised)학습	-입력 -출력 -기법	문제 + 결과 데이터 (사과 카드 + '사과') 예측 결과 제공 회귀, 분류, 랭킹
강화(reinforcement)학습	-입력 -출력 -기법	문제 데이터만 제공 결과, 결과 평가 후 보상 알고리즘 트레이닝 상 최대화 / 벌 최소화 방향
비지도(unsupervised)학습	-입력 -출력 -기법	문제 데이터만 제공 분석 결과 제공 군집화, 토픽 모델링, 밀도 추청, 차원 축소

ABOUT AI

◆ 머신러닝(ML:Machine Learning)



ABOUT AI

◆ 머신러닝(ML:Machine Learning)

기법 및 모델	특징
분류(Classification)	데이터의 **특정 그룹 부여**하여 구분
회귀(Regression)	수치 데이터에 사용되는 기법, **연속형 결과 예측**
의사결정 나무 (Decision Tree)	트리 구조 형태의 예측 모델을 사용하는 기법
인공 신경망 (Neural Network)	생물의 신경 네트워크 구조와 기능 모방 기법
유전자 프로그래밍(Genetic Programming)	**생물의 진화 알고리즘에 기반**한 기법
군집화(Clustering)	관측된 예를 **군집하여 부분집합으로 배분 기법**
몬테카를로 방법 (Monter Carlo method)	무작위 추출된 난수 통해 **확률로 계산하는 기법**

ABOUT AI

◆ 딥러닝(DL:Deep Learning)

- 머신 러닝의 한 분야 => **신경망**
- 학습 데이터를 구분하는 **층(Layer)**을 많이 만들어 그 정확도를 올리는 방법

- 2016년 알파고에 적용
- 영상 인식, 음성 인식, 자연 언어 처리 등 분야에서 우수한 성능 발휘

ABOUT AI

◆ 딥러닝(DL:Deep Learning)

기법 및 모델	특징
퍼셉트론(Perceptron)	-**1957년 고안된 알고리즘** -**딥러닝 신경망의 기원이 되는 알고리즘** -다수의 신호를 입력 받아서 하나의 신호 출력(0 / 1) -뉴런 또는 노드로부터 신호 입력 & 신호 출력
인공신경망 **(ANN: Artificial Neural Network)**	-1980년대부터 활발히 연구 -뇌신경망의 패턴 인식 방식의 통계학적 학습 알고리즘 -선형 맞춤과 비선형 변환 반복 수행하여 최적화
심층신경망 **(DNN: Deep Neural Network)**	-신경망이 **다수의 층의 깊이로써 구성된 개념** -입력층과 출력층 사이 다수의 은닉층으로 구성된 ANN -**분류 및 수치 예측**을 위한 것 -**이미지 트레이닝, 문자인식 매우 유용**

ABOUT AI

◆ 머신러닝(ML) & 딥러닝(DL)용 필수 라이브러리

라이브러리	언어	특징
NumPy	Python	- www.numpy.org - 파이썬에서 수치해석, 통계, 과학 계산용 모듈 - array자료 생성, 색인, 처리, 연산 수행 - Scipy, Pandas, matplotlib 등 다른 Python 패키지와 함께 사용
SciPy	Python	- www.scipy.org/ - 과학 계산용 함수를 모아 놓은 오픈 소스 파이썬 패키지 - 선형 대수, 함수 최적화, 신호 처리, 특수 수학 함수, 통계 분포 등
Matplotlib	Python	- www.matplotlib.org - Matlab을 기반으로 파이썬으로 확장한 패키지로 그래프 패키지 - 데이터와 분석 결과를 다양한 관점으로 시각화 가능
Pandas	Python	- www.pandas.pydata.org/ - Series와 DataFrame 데이터 구조 제공 및 데이터 처리 패키지

ABOUT AI

◆ 머신러닝(ML) 라이브러리 프레임워크

라이브러리	언어	특징
Scikit-lean	Python	- 다양한 지도학습, 비지도학습 알고리즘 구현 - Python기반으로 NumPy, SciPy이용한 고속화 지원
Statsmodels	Python	- 검정 및 추정, 회귀분석, 시계열분석등 다양한 통계분석 - 회귀분석의 경우 patsy 패키지 포함
NLTK	Python	- http://www.nltk.org/ - Natural Language Toolkit약자 자연어 처리 라이브러리
MLib	Python,Java, Scala	- 아파치 스마크용 머신러닝 라이브러리
Weka	Java	- 자바로 쓰인 데이터마이닝 라이브러리 / GUI 제공
OpenCV	C/C++, Java, Python	- 2000년에 공개되었으며 이미지 데이터 처리 함수 풍부
Pytorch	Python	- 페이스북 인공지능 연구팀에서 개발

ABOUT AI

◆ 딥러닝(DL) 라이브러리 프레임워크

라이브러리	언어	특징
Tensorflow **Keras**	Python/ C++	▪ **구글 AI 연구팀**에서 개발한 **오픈소스 라이브러리** ▪ 머신 러닝 & 딥 러닝 알고리즘 충족 ▪ 풍부한 자연어용 딥러닝 함수, 데이터 모델 라이브러리 제공
Caffe	Python, Matlab	• caffe.berkeleyvision.oeg) • 캘리포니아 버클리대 오픈소스로 개발된 딥러닝 프레임워크 • 유연한 구조, 코드 확장성, 고속으로 처리가능, 커뮤니티 활성화 • 미리 학습된 다양한 분류 모델 사용 가능
PyTorch	C, Lua	▪ **Facebook AI팀**에서 개발한 Python 위한 오픈소스 머신 러닝 라이브러리 ▪ 뉴욕 대학에서 C, LuaJIT로 개발한 라이브러리 ▪ 자연어 처리 및 애플리케이션 위해 사용

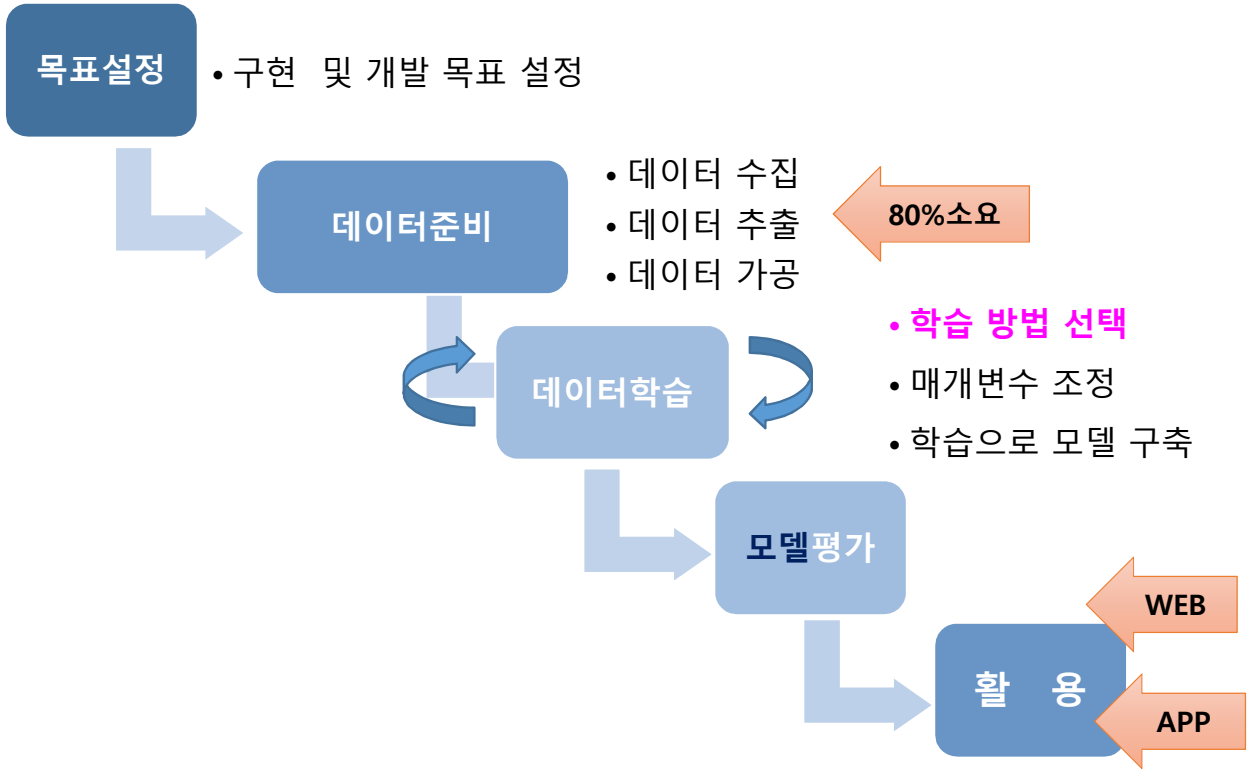
ABOUT AI

◆ 딥러닝(DL) 라이브러리 프레임워크

라이브러리	GPU	특징
CUDA	NVIDIA	▪ NVIDIA에서 만든 **GPU 플랫폼이자 API** ▪ 그래픽 처리 장치에서 수행하는 알고리즘을 C 프로그래밍 언어 비롯한 산업 표준 언어를 사용하여 작성할 수 있도록 하는 **GPGPU 기술** ▪ 많은 딥러닝 라이브러리에서 CUDA 지원 ▪ NVIDIA의 GPU에서 사용
OpenCL	Intel, AMD	▪ Apple에서 개발하고 Khronos Group에서 관리하는 연산 플랫폼 ▪ CPU/GPU 동시동작용 프로그램 개발 라이브러리 ▪ Intel/AMD 외 이기종 컴퓨터 표준 제공을 위해 개발 ▪ 많은 딥러닝 프레임워크에서 OpenCL 미지원

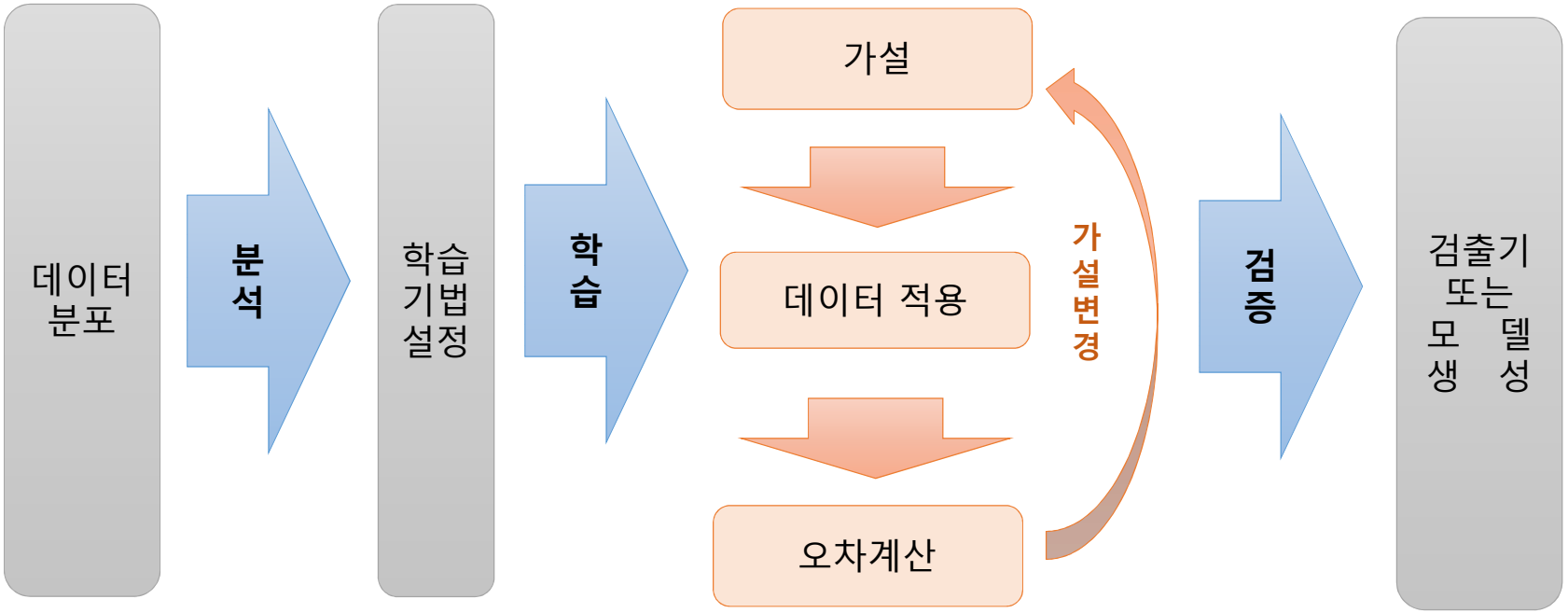
ABOUT AI

◆ 머신러닝(ML) 프로세스



ABOUT MACHINE LEARNING

◆ 머신러닝(ML:Machine Learning) 프로세스



ABOUT MACHINE LEARNING

◆ 머신러닝(ML:Machine Learning) 학습 준비

❖ 데이터셋(Dataset)

- **정학한 규칙/패턴**을 찾기 위해 필요한 데이터 집합
- 종류
  - ✓ 규칙/패턴을 찾기 위한 데이터 => **학습용 데이터셋**
  - ✓ 규칙/패턴을 찾는 과정 중 검사 데이터 => **검증용 데이터셋**
  - ✓ 규칙/패턴을 평가 위한 데이터 => **테스트용 데이터셋**

ABOUT MACHINE LEARNING

◆ 머신러닝(ML:Machine Learning) 학습 준비

❖ 데이터셋(Dataset)

- 분류
  - ✓ 규칙/패턴 찾기 위해 입력되는 데이터 ➔ **입력/문제/피쳐/특성/속성**
  - ✓ 입력 데이터에 해당하는 정답 데이터 ➔ **타겟/정답/클래스/라벨**

ABOUT MACHINE LEARNING

◆ 머신러닝(ML:Machine Learning) 학습 준비

- **모델(Model)**
  - 데이터가 가지는 **일정한 규칙 또는 패턴을 수식**으로 만든 것
  - **가장 많은 데이터를 만족하는 규칙/패턴 수식 즉 모델 찾는 것 ➔ AI 목표**

ABOUT MACHINE LEARNING

◆ 머신러닝(ML:Machine Learning) 학습 준비

❖ 지도학습 데이터셋(Dataset) : 데이터 + 라벨

이미지(데이터)	이름(라벨)
	가방
	가방
	가방
	가방
	가방
	가방
	가방
	가방

이미지(데이터)	이름(라벨)
	고양이
	고양이
	고양이
	고양이
	고양이
	고양이
	고양이
	고양이

평수(데이터)	관리비(라벨)
18	90,000
24	140,000
29	190,000
32	210,000
39	250,000
42	280,000
49	330,000
54	370,000

ABOUT MACHINE LEARNING

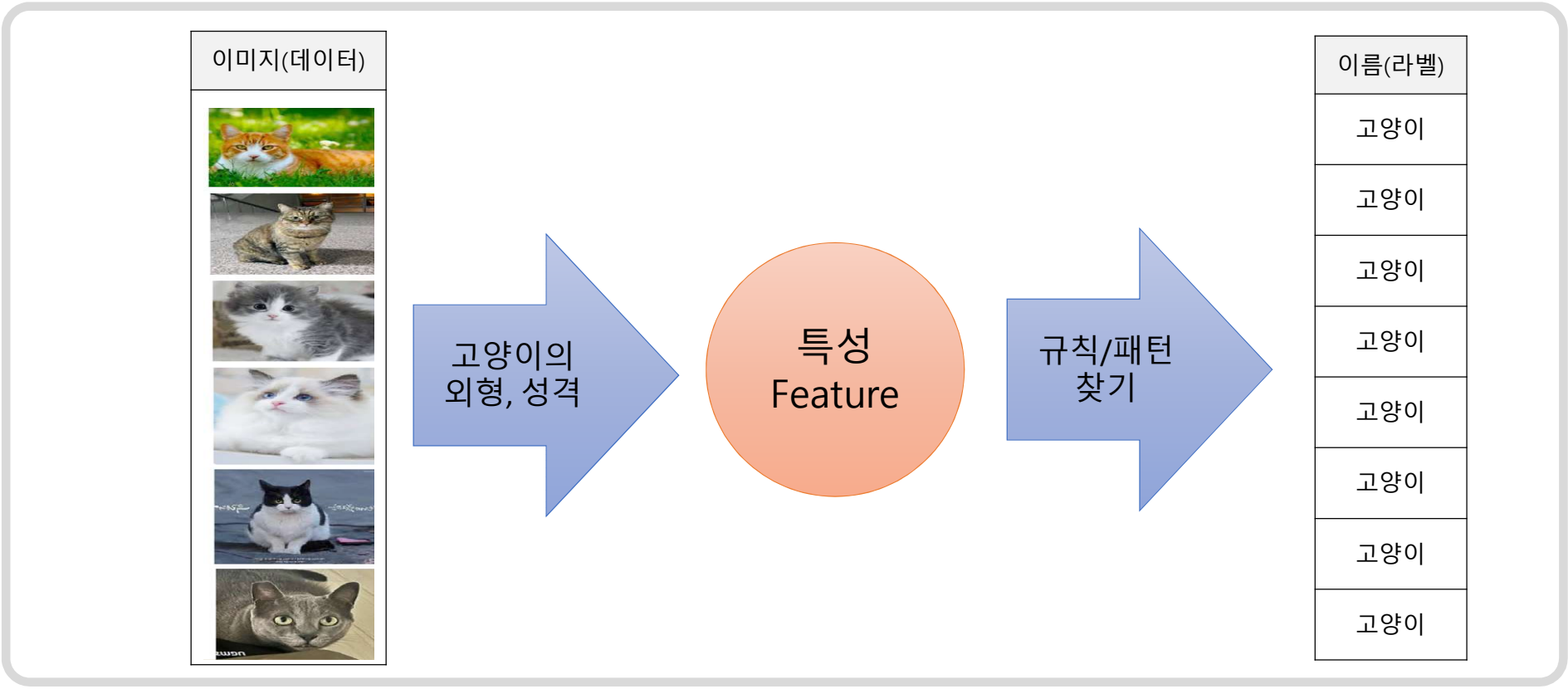
◆ 머신러닝(ML:Machine Learning) 학습 준비

❖ 비지도학습 데이터셋(Dataset) : 데이터

18
24
29
32
39
42
49
54

ABOUT MACHINE LEARNING

◆ 머신러닝(ML:Machine Learning) 학습 준비



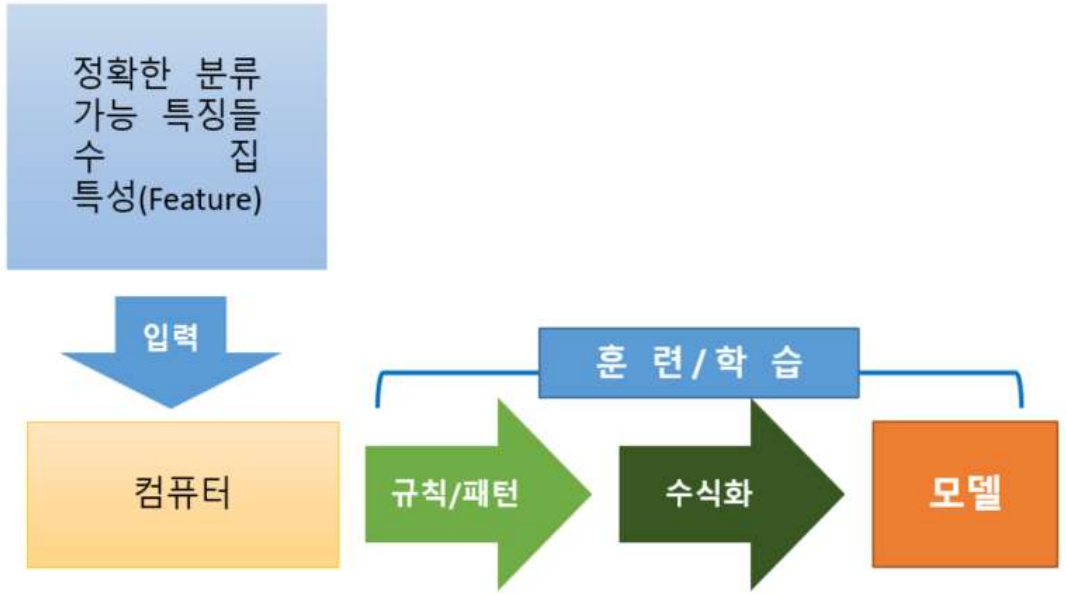
PART II

CLASSIFICATION WITH ML

CLASSIFICATION WITH ML

◆ 생선 분류

- 고등어 ➔ 생김새, 크기, 색상, 등쪽, 배쪽, 무게
- 도 미 ➔ 생김새, **크기**, 색상, 등쪽, 배쪽, **무게**
- 빙 어 ➔ 생김새, **크기**, 색상, 등쪽, 배쪽, **무게**



CLASSIFICATION WITH ML

◆ 생선 분류 - ① 목표 설정 및 데이터 수집

❖ 목표 : 도미, 빙어 생선 분류

❖ 특성 데이터 : 길이와 무게

Species	Weight	Length	Diagonal	Height	Width
Bream	242	25.4	30	11.52	4.02
Bream	290	26.3	31.2	12.48	4.3056
Bream	340	26.5	31.1	12.3778	4.6961
Bream	363	29	33.5	12.73	4.4555
Bream	430	29	34	12.444	5.134
Bream	450	29.7	34.7	13.6024	4.9274
Bream	500	29.7	34.5	14.1795	5.2785
Bream	390	30	35	12.67	4.69
Bream	450	30	35.1	14.0049	4.8438
Bream	500	30.7	36.2	14.2266	4.9594
Bream	475	31	36.2	14.2628	5.1042
Bream	500	31	36.2	14.3714	4.8146
Bream	500	31.5	36.4	13.7592	4.368
Bream	340	32	37.3	13.9129	5.0728
Bream	600	32	37.2	14.9544	5.1708

CLASSIFICATION WITH ML

◆ 생선 분류 - ② 데이터 추출

❖ 특성 데이터 : 길이와 무게

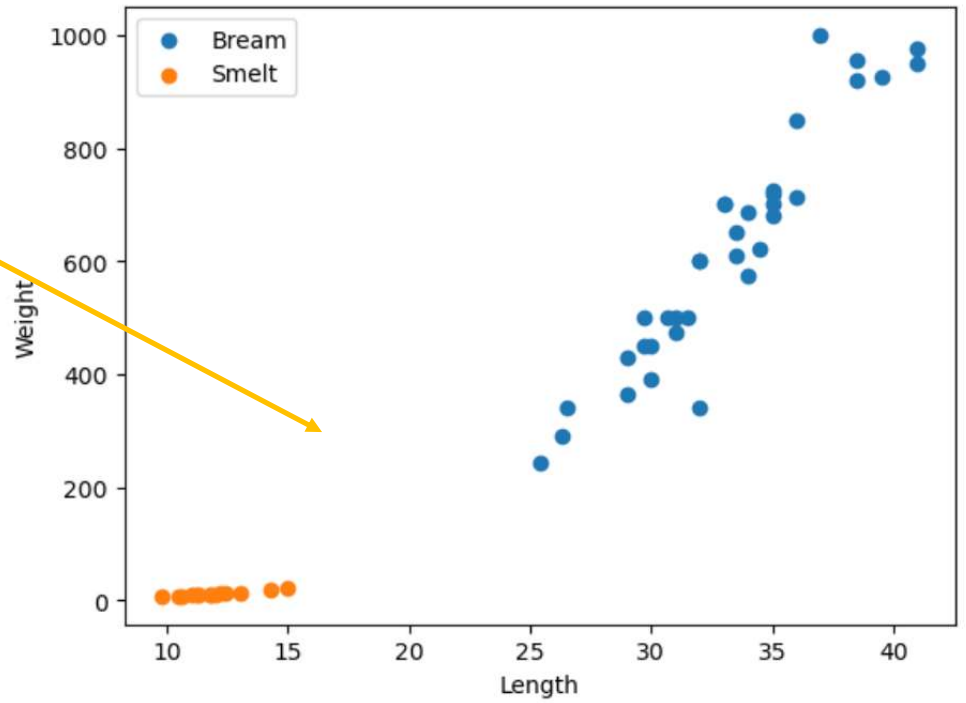
Species	Weight	Length	Diagonal	Height	Width
Bream	242	25.4	30	11.52	4.02
Bream	290	26.3	31.2	12.48	4.3056
Bream	340	26.5	31.1	12.3778	4.6961
Bream	363	29	33.5	12.73	4.4555
Bream	430	29	34	12.444	5.134
Bream	450	29.7	34.7	13.6024	4.9274
Bream	500	29.7	34.5	14.1795	5.2785
Bream	390	30	35	12.67	4.69
Bream	450	30	35.1	14.0049	4.8438
Bream	500	30.7	36.2	14.2266	4.9594
Bream	475	31	36.2	14.2628	5.1042
Bream	500	31	36.2	14.3714	4.8146
Bream	500	31.5	36.4	13.7592	4.368
Bream	340	32	37.3	13.9129	5.0728
Bream	600	32	37.2	14.9544	5.1708

```
In [1]: # 도미 데이터
        bream_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7,
                        31.0, 31.0, 31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5,
                        34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 35.0, 36.0, 36.0, 37.0,
                        38.5, 38.5, 39.5, 41.0, 41.0]
        bream_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0, 450.0, 500.0,
                        475.0, 500.0, 500.0, 340.0, 600.0, 600.0, 700.0, 700.0, 610.0, 650.0,
                        575.0, 685.0, 620.0, 680.0, 700.0, 725.0, 720.0, 714.0, 850.0, 1000.0,
                        920.0, 955.0, 925.0, 975.0, 950.0]

In [2]: #빙어 데이터
        smelt_length = [9.8, 10.5, 10.6, 11.0, 11.2, 11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3, 15.0]
        smelt_weight = [6.7, 7.5, 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4, 12.2, 19.7, 19.9]
```

◆ 생선 분류 - ③ 데이터 분석

❖ 특성 데이터의 관계 분석

도미와 빙어
분류 가능



◆ 생선 분류 - ④ 데이터 가공

❖ 지도학습용 데이터셋 형태 ➔ 데이터/특성 + 라벨/타겟

➢ 도미+빙어

길이	무게
254.	242.0
26.3	290.0
26.5	340.0
29.0	363.0
29.7	430.0
29.7	450.0
30.0	500.0
:	:
14.3	19.7
15.0	19.9

생선종류
도미
도미
도미
도미
도미
도미
도미
:
빙어
빙어

숫자로 표기

생선종류
1
1
1
1
1
1
1
:
0
0

◆ 생선 분류 - ④ 데이터 가공

❖ 지도학습용 Scikit-learn 라이브러리 데이터셋 형태

- 데이터 => **2차원**

```
# 1마리씩 길이와 무게 데이터 묶기
fish_data=[[l, w] for l, w in zip(length, weight)]
[[25.4, 242.0], [26.3, 290.0], [26.5, 340.0], ...[13.0, 12.2], [14.3, 19.7], [15.0, 19.9]]
```

- 라 벨 => 1차원

```
# 생성 종류 정보 데이터 묶기
# 도미 ==> 1,  빙어 ==> 0
fish_target=[1]*len(bream_length) + [0]*len(smelt_length)
[1, 1, 1, 1, 1, 1, ... , 0, 0, 0, 0, 0, 0, 0, 0]
```

CLASSIFICATION WITH ML

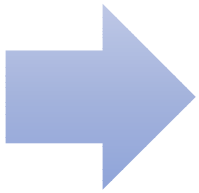
◆ 생선 분류 - ⑤ 학습 방법 선택 : Scikit-learn 라이브러리

❖ 지도학습용 Scikit-learn 라이브러리 – 분류 학습 방법

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...



분류를 위한 **여러가지 학습 방법** 중
가장 성능이 좋은 방법 선택

모든 학습 방법을 적용하기에는 많은 시간 소요
많은 개발자들의 **경험들을 기반으로 추천되는 학습 방법 선택**

PART II

SCIKIT-LEARN ML LIBRARY

SCIKIT-LEARN ML LIBRARY

◆ Scikit-learn 라이브러리

- **머신러닝 알고리즘, 전처리, 검증** 등 기능 제공 **라이브러리**
- **Numpy & Scipy 기반 속도 최적화**
- 바로 테스트 가능한 **다양한 샘플 데이터 제공**
- 많이 사용되는 다른 라이브러리와 호환 가능
- **BSD 라이선스**로 **무·료상업적 사용 가능**
- **가장 쉽고 효율적**인 개발 라이브러리
- **실전 환경**에서 **검증 & 매우 많은 환경**에서 사용

SCIKIT-LEARN ML LIBRARY

◆ Scikit-learn 라이브러리

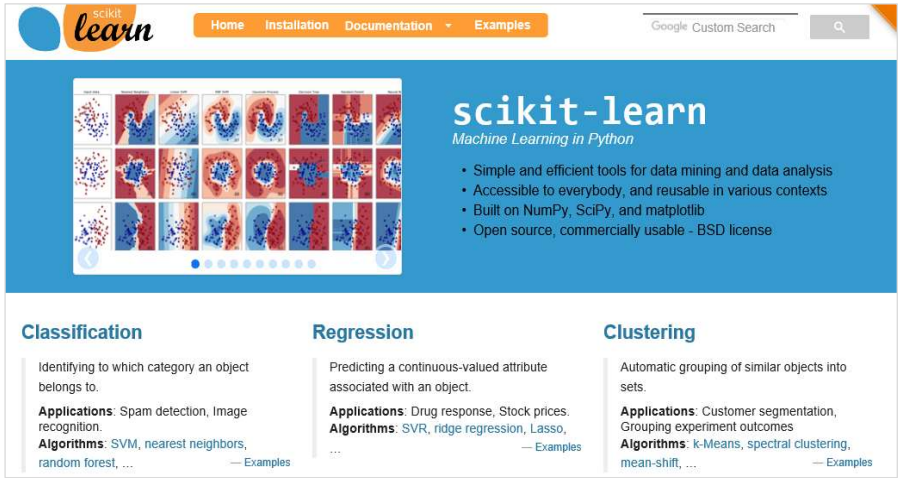
- **설치**

```
conda install scikit-learn
!pip install scikit-learn
```

- **설치 확인**

```
import sklearn

print( sklearn._ _version_ _ )
```

https://scikit-learn.org
https://scikit-learn.org/stable/modules/classes.htm



SCIKIT-LEARN ML LIBRARY

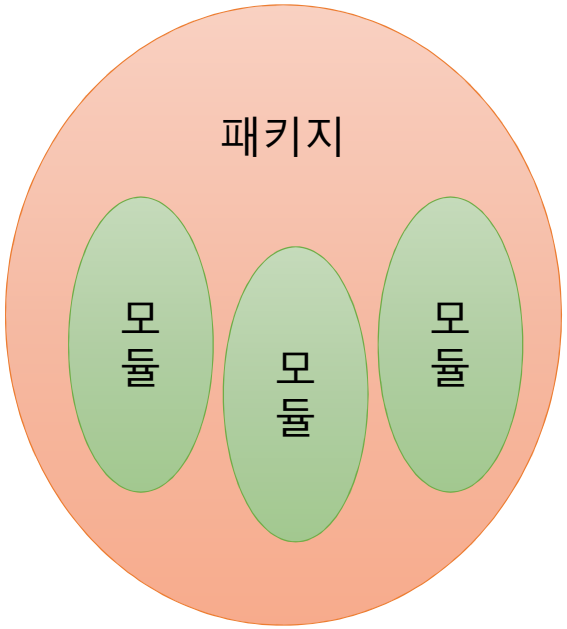
◆ Scikit-learn 라이브러리

▪ 모듈 ➔

- 파이썬 파일 1개
- 클래스, 함수, 변수 존재
- 반드시 3가지 모두 존재 하지 않을 수도 있음

▪ 패키지 ➔

- 동일한 목적의 모듈들을 하나로 묶음
- 모듈이 여러 개 존재함



SCIKIT-LEARN ML LIBRARY

◆ Scikit-learn 라이브러리

분 류	모 듈 명	내 장 기 능
예제 데이터	sklearn.**datasets**	- 연습용 데이터셋
피쳐 처리	skelarn.**preprocessing**	- 전처리 관련 기법 (원핫 인코딩, 정규화, 스케일링 등)
	sklearn.**feature_selection**	- 모델에 중요한 영향 미치는 피쳐 탐색 및 선택 기법
	skelarn.**feature_extraction**	- 원시 데이터로부터 피쳐 추출 기능 - 이미지 피쳐 추출 : 하위 모듈 image - 텍스트 피쳐 추출 : 하위 모듈 text
차원 축소	sklearn.**decomposition**	- 차원 축소 관련 알고리즘 계열 (PCA, NMF, Truncated SVD 등)

SCIKIT-LEARN ML LIBRARY

◆ Scikit-learn 라이브러리

분 류	모 듈 명	내 장 기 능
기계학습 알고리즘	sklearn.**ensemble**	- 앙상블 알고리즘 계열(랜덤 포레스트, 에이다 부스트, 배깅 등)
	skelarn.**linear_model**	- 선형 알고리즘 계열(선형회귀, 로지스틱 회귀, SGD 등)
	skelarn.**naïve_bayes**	- 나이브 베이즈 알고리즘 계열 (베르누이 NB, 가우시안 NB, 다항분포 NB 등)
	skelarn.**neighbors**	- 최근접 이웃 알고리즘 계열(K-nn 등)
	skelarn.**svm**	- 서포트 백터 머신 알고리즘 계열(SVC, SVR 등)
	skelarn.**tree**	- 의사 결정 나무 알고리즘 계열
	skelarn.**cluster**	- 비지도 학습 알고리즘 계열 (KMeans, DBSCAN 등)

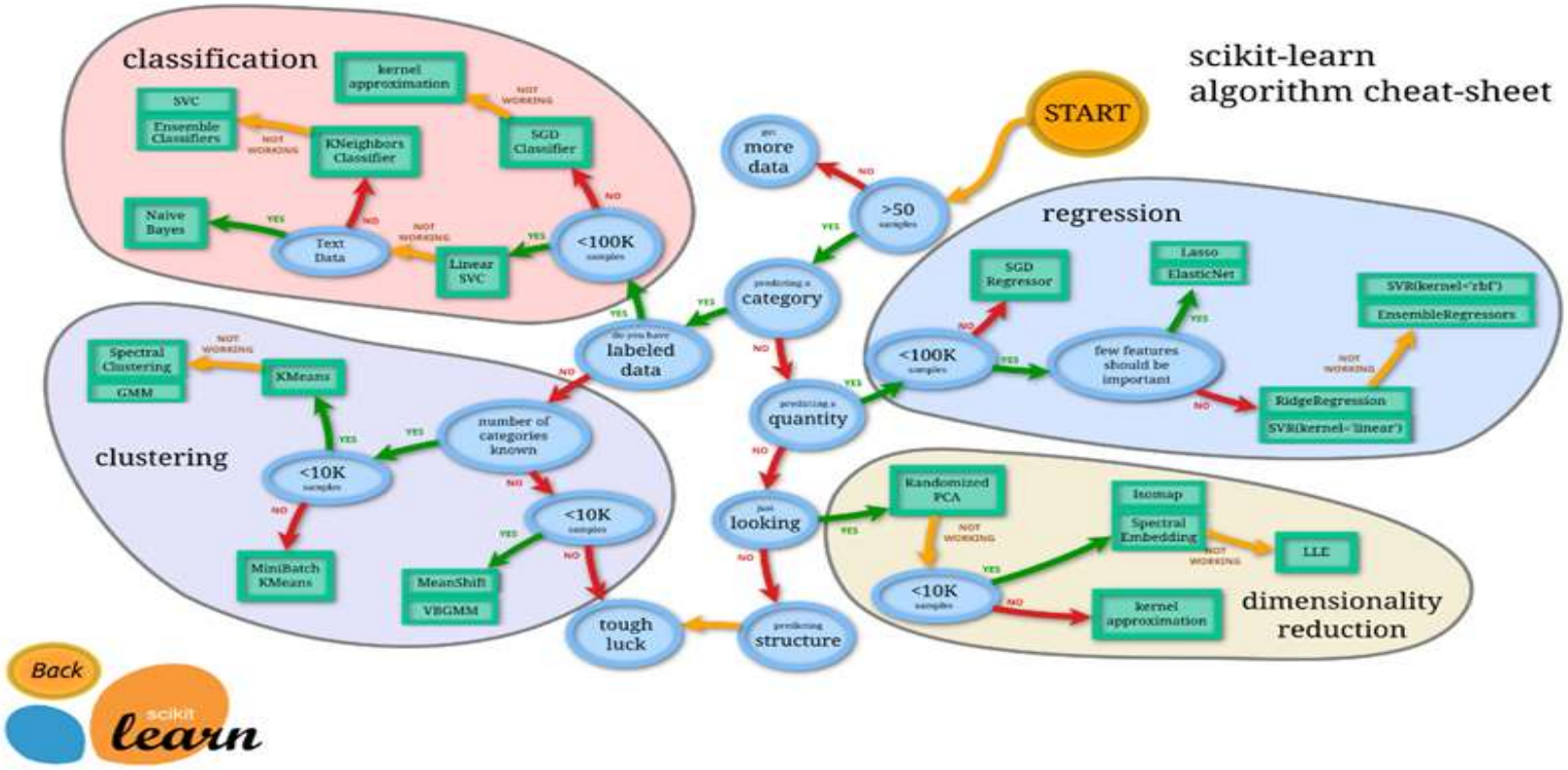
SCIKIT-LEARN ML LIBRARY

◆ Scikit-learn 라이브러리

분 류	모 듈 명	내 장 기 능
검증 및 튜닝	skelarn.**model_selection**	- 검증, 하이퍼 파라미터 튜닝, 데이터 분리 등 cross_validate, GridSearchCV, train_test_split, learning_curve 등
모델 평가	skskelarn.**metrics**	- 모델 성능 측정 및 평가 기법 accuracy, precision, recall, ROC curve 등
유틸리티	skelarn.**pipeline**	- 피쳐 처리 등의 변환과 기계학습 알고리즘 등을 연쇄적으로 실행 기능

SCIKIT-LEARN ML LIBRARY

◆ Scikit-learn 라이브러리



PART Ⅱ

SCIKIT-LEARN ML
CALSSIFICATION

CLASSIFICATION

◆ 지도학습 - 분류 (Classification)

- 데이터구성 : 데이터/특성・피쳐(Feature) + 라벨/타겟/정답/클래스
- 라벨 / 타겟 : **데이터/특성을 그룹으로 나누는 경우**
  - ➢ 이진 분류 : **2가지 결과**로 나누는 경우
  - ➢ 다중 분류 : **3가지 이상 결과**로 나누는 경우

CLASSIFICATION

◆ 지도학습 분류 (Classification)

❖ ML위한 다양한 학습 방법들

- KNN (K-nearest neighbor)
- Naive Bayes
- SVM(Support Vector Machine)
- Decision Tree
- Random Forest
- :

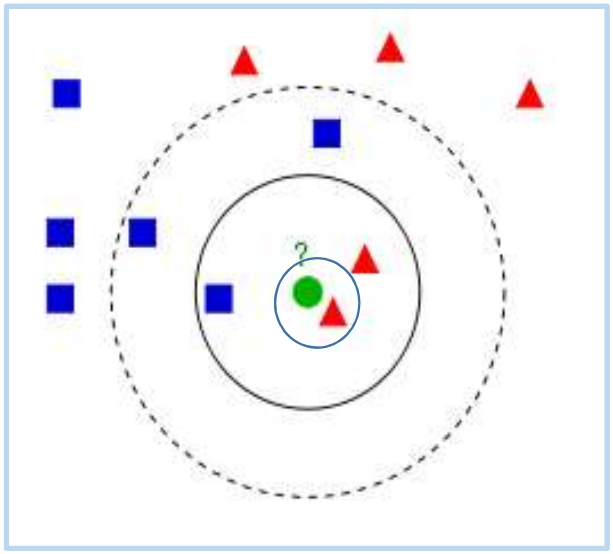
_ _ _Classification
_ _ _Classifier

- KNeighbors**Classifier**
- GaussianNB
- SV**C**, LinearSV**C**
- DecisionTree**Classifier**
- RandomForest**Classifier**
- :

CLASSIFICATION

◆ KNN : K-Nearest Neighbor

❖ 원리



녹색 점은 파랑그룹 소속일까? 빨강그룹 소속일까?

- 일정 범위 지정
- 범위내 **그룹 개수 및 거리**

➔ 녹색점 **근처에 몇 개의 점(K)을 기준으로 할 지 결정**

➔ **K-최근접이웃 알고리즘**

CLASSIFICATION

◆ KNN : K-Nearest Neighbor

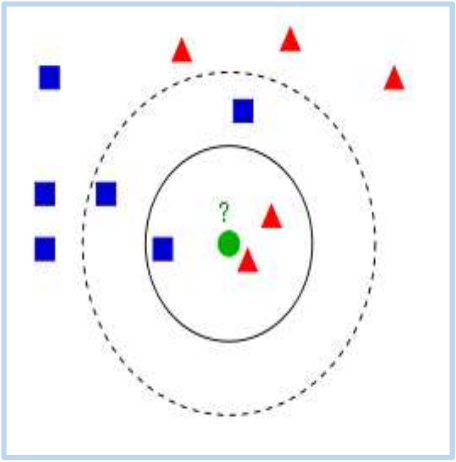
❖ 특징

- **가장 간단한 머신러닝 알고리즘**으로 **모델 생성하지 않음 ➔ 가설식 없음**
- **관측치/데이터만 이용**하여 새로운 데이터 예측 : Instance-based Learning
- 모든 관측치/데이터를 **메모리에 저장 후 예측** : Memory-based Learning
- 모델 학습 없이 **새로운 데이터 입력 시 거리 측정으로 예측** : Lazy Learning

CLASSIFICATION

◆ KNN : K-Nearest Neighbor

❖ 원리



1.**점A**와 데이터(훈련 데이터 집합)들과의 **거리 측정**
거리 측정으로는 Euclidean distance 보편적으로 이용 ➔ 유사도

2.측정한 **거리가 작은 순서대로 K개의 점 검색**

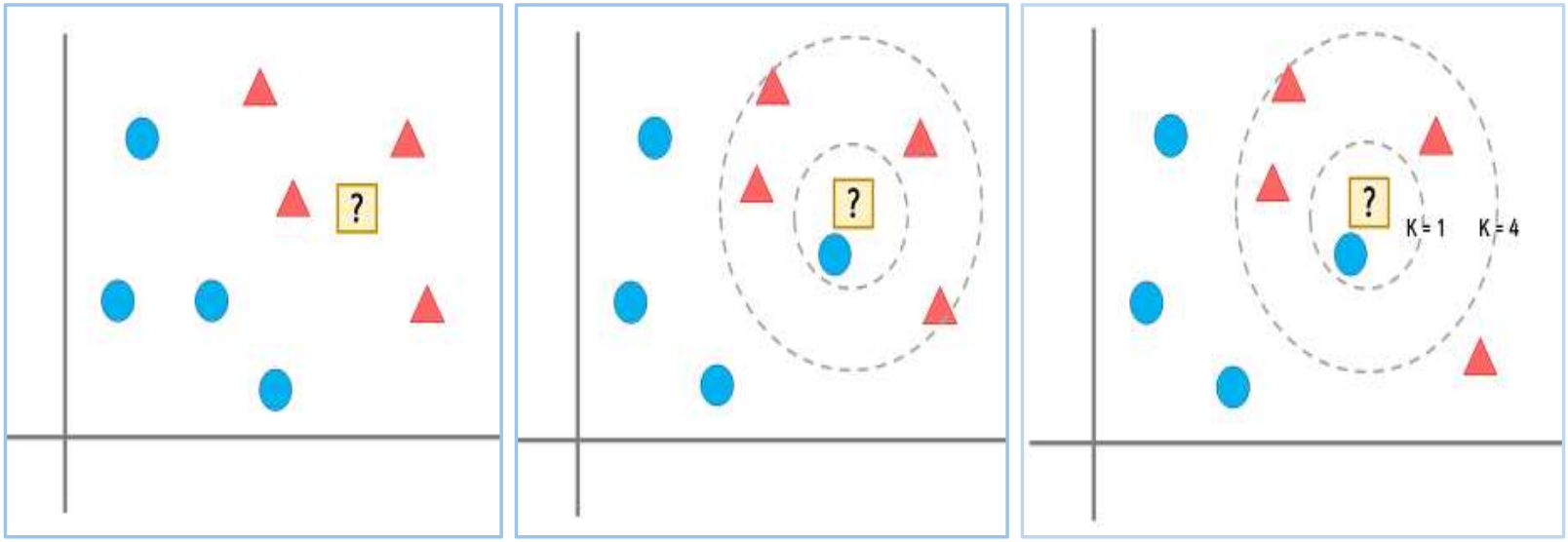
3-1. 분류 : K개 점들 속한 그룹의 **가장 많은 개수(다수결 원칙) 그룹 선정**

3-2. 회귀 : K개 점들 속한 그룹의 **K개의 값들의 대표값 즉 평균으로 추정**

CLASSIFICATION

◆ KNN : K-Nearest Neighbor

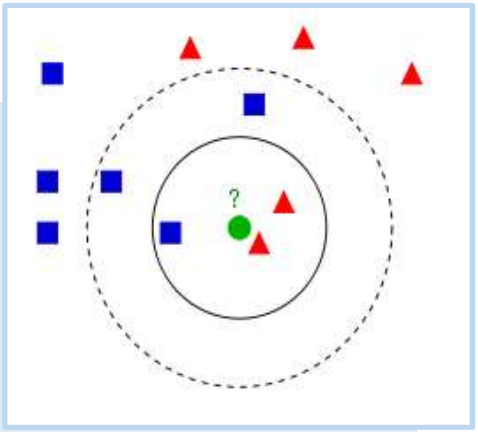
❖ 최적의 k?



CLASSIFICATION

◆ KNN : K-Nearest Neighbor

❖ k 선정 방법



- K 범위 : 1 ~ 전체 데이터 갯수
- 너무 큰 K : 과소적합(Underfitting)
- 너무 작은 K : 과대적합(Overfitting)
- 최적 K : cross-validation 통해 Error가 제일 적은 K

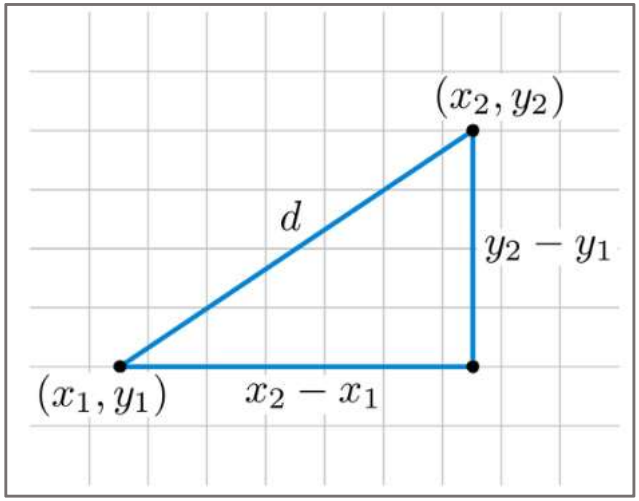
CLASSIFICATION

◆ KNN : K-Nearest Neighbor

❖ 거리 측정 방법

- **유클리드거리(Euclidean Distance: L2 Distance)**
  - ➢ 가장 흔히 사용되는 거리측도
  - ➢ 두 관측치 사이의 **직선 거리**

$$d(A,B) = \sqrt{(x2-x1)^2 + (y2-y1)^2}$$



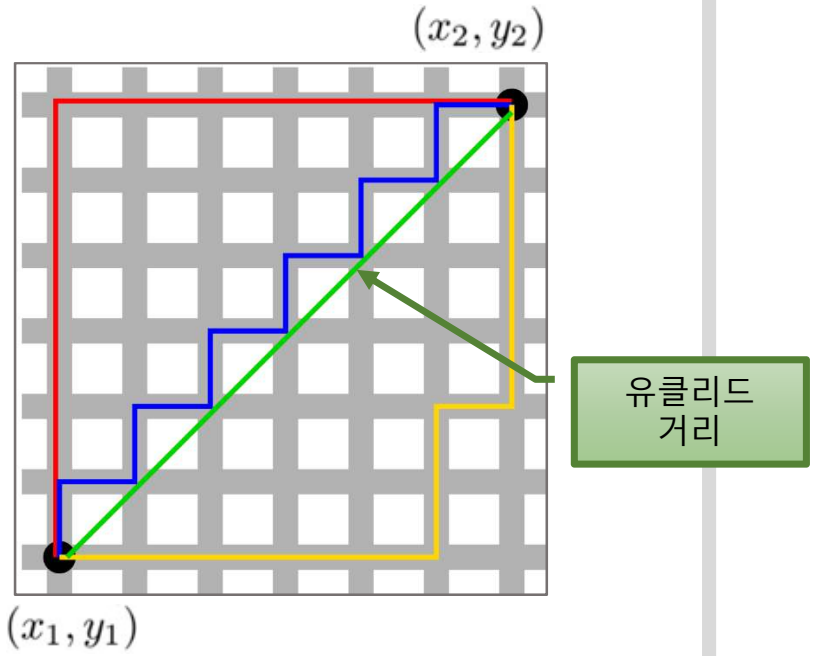
CLASSIFICATION

◆ KNN : K-Nearest Neighbor

❖ 거리 측정 방법

- 맨하탄거리(Manhattan Distance: L1 Distance)
  - ➢ **가로, 세로로만 이동**할 수 있는 물체 거리
  - ➢ **체스판 말 이동 거리** 측정
  - ➢ **택시 거리(Taxicab distance)** 라고도 함

$$d(A,B) = |x1 - x2| + |y1 - y2|$$



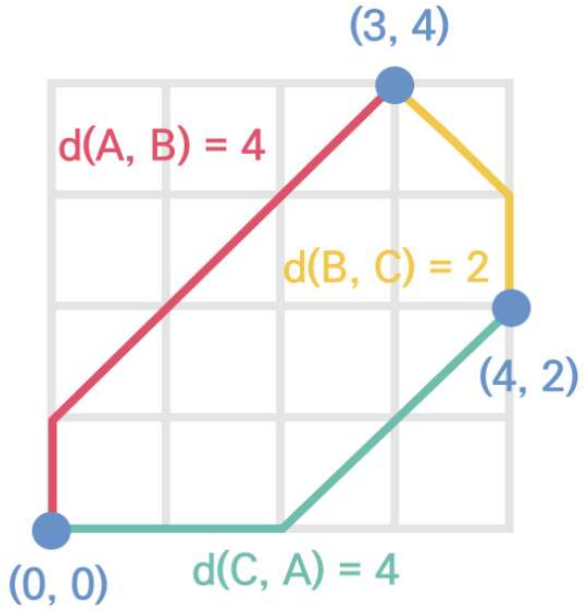
CLASSIFICATION

◆ KNN : K-Nearest Neighbor

❖ 거리 측정 방법

- 체비쇼프거리(Chebyshev Distance: L-∞ Distance)
  - ➢ **가로, 세로, 대각선으로 이동할 수 있는 물체 거리**
  - ➢ **체스판에서 가로, 세로, 대각선 이동**
  - ➢ **가장 긴 거리를 거리값으로 선택하는 방식**
  - ➢ 로지스틱에서 두 집단 유사도에 사용

$$\max\left(|x_1 - y_1|,\ |x_2 - y_2|\right)$$



CLASSIFICATION

◆ KNN : K-Nearest Neighbor

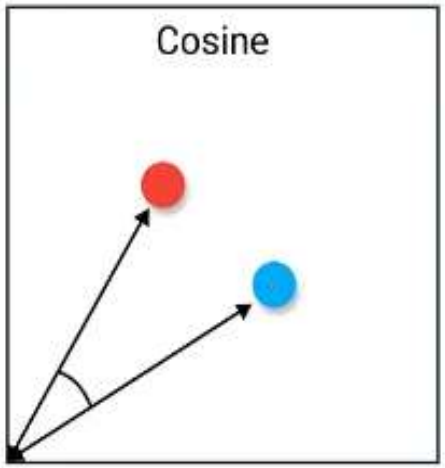
❖ 거리 측정 방법

- 민코프스키 거리(Minkowski Distance: L1 Distance)
  - **유클리드, 맨해튼 거리 일반화**
  - n 차원 점 A, B에 대해 p차 민코프스키 거리
  - p = 1 : 맨해튼 거리와 동일, L1 norm
  - p = 2 : 유클리드 거리와 동일, L2 norm
  - p = ∞: 체비쇼프 거리와 동일, L max norm

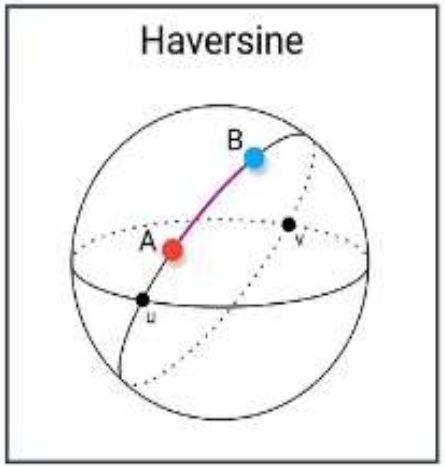
CLASSIFICATION

◆ KNN : K-Nearest Neighbor

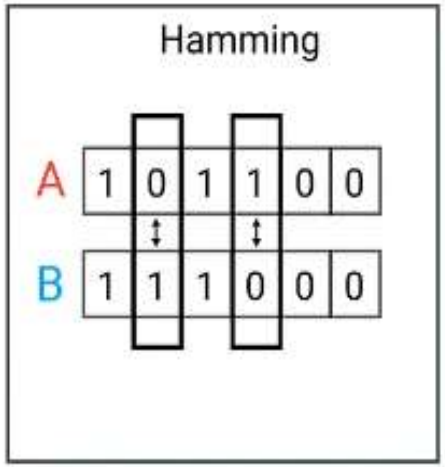
❖ 다양한 거리 측정



기울기와 방향에
대한 유사도



지리정보에 관련된 좌표
다룬 거리



같은 길이 가진 두 부호열 또는
비트열에 대해 **같은 위치에서**
서로 다른 기호 개 수

CLASSIFICATION

◆ KNN : K-Nearest Neighbor

❖ 수치 피쳐값 정규화

- **거리기반** 모델로 **피쳐/특성 데이터(길이, 무게) 값의 범위 재조정** 필요
  - **min-max normalization**
  - **z-score standardsiztion**

CLASSIFICATION

◆ Sklearn 라이브러리 - KNeighborsClassifier

❖ Sklearn Lib – 분류 모델 객체

```
from sklearn.neighbors import KNeighborsClassifier (
    n_neighbors=5,          ← 범위, 기본값
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,                    ← 거리 측정 방법 1: 맨해튼, 2:유클리디안
    metric='minkowski',     ← 거리 측정 방법
    metric_params=None,
    n_jobs=None )           ← 학습 시 사용할 CPU 코어 수, -1:모든 CPU
```

CLASSIFICATION

◆ Sklearn 라이브러리 - KNeighborsClassifier

❖ Sklearn Lib - 학습 메서드

classifer.fit(학습데이터, 타겟데이터)

- 학습데이터 : **2차원 형태**
- 타겟데이터 : **1차원 형태**

[학습 후 확인 속성]

- classifier.classes_ : 분류 타겟/라벨/클래스 수
- classifier.n_features_in_ : 특성 개수
- classifier.n_samples_fit_ : 학습 데이터 수

CLASSIFICATION

◆ Sklearn 라이브러리 - KNeighborsClassifier

❖ Sklearn Lib - 예측 메서드

예측결과 = **classifer.predict(특성데이터)**

- 특성데이터 : **2차원 형태**
- 예측결 과 : **1차원 형태**

CLASSIFICATION

◆ Sklearn 라이브러리 - KNeighborsClassifier

❖ Sklearn Lib - 성능평가 메서드

성능결과 = **classifer.score(테스트데이터, 테스트타겟)**

- 테스트데이터 : **2차원 형태**
- 테스트타 겟 : **1차원 형태**
- 성 능결 과 : **정확도 평균값**

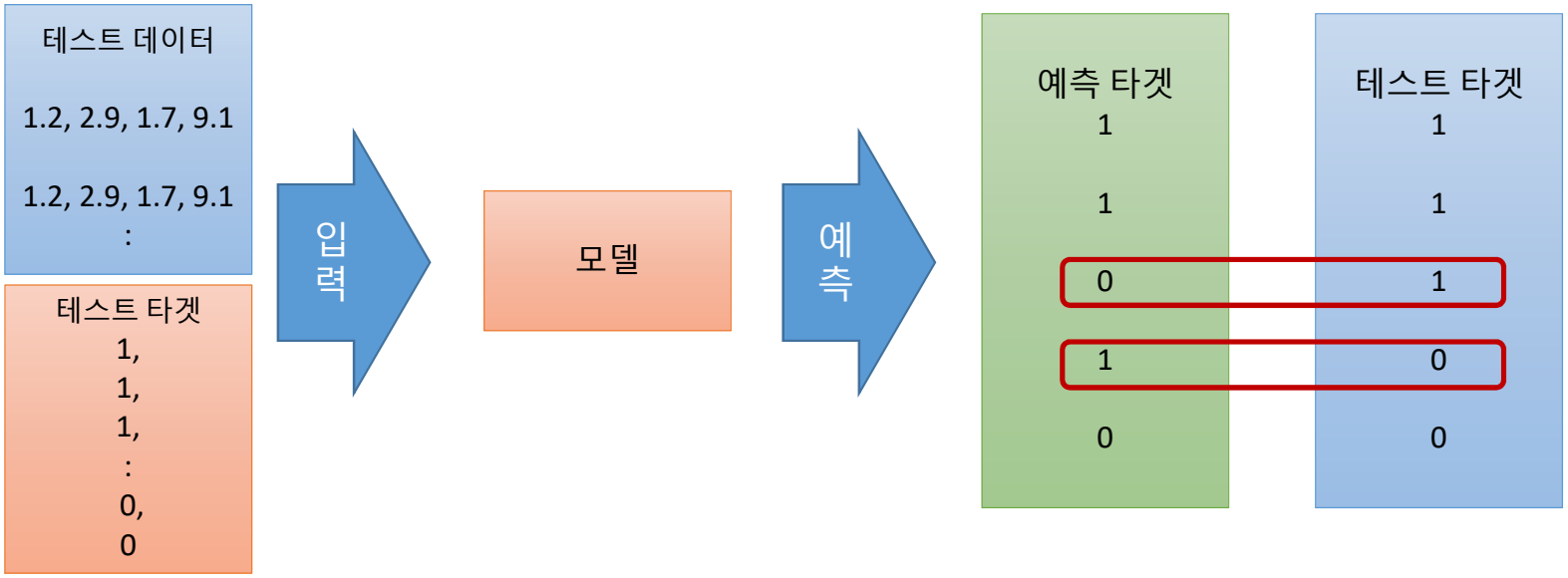
[정확도 Accuracy]

- 정답을 맞춘 개수를 백분율로 나타낸 값
- 맞춘 개수 / 전체 데이터 수
- 값의 범위 : 0.0 ~ 1.0

CLASSIFICATION

◆ Sklearn 라이브러리 - KNeighborsClassifier

❖ Sklearn Lib - 성능평가 메서드



CLASSIFICATION

◆ 머신러닝을 위한 데이터 준비

❖ **훈　련 데이터** : 학습에 사용되는 데이터와 라벨

❖ **테스트 데이터** : 학습 후 모델 성능 평가에 사용되는 데이터와 라벨

❖ **검　증 데이터** : 학습 중간에 평가하는 데이터

❖ 데이터 한　개 : 샘플

CLASSIFICATION

◆ 머신러닝을 위한 데이터 준비 - Sklearn 라이브러리

❖ Sklearn Lib – 데이터 분리 메서드 : sklearn.model_selection 모듈

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
        학습 데이터,                    ← 2차원 형태
        학습 타겟,                      ← 1차원 형태
        stratify,                       ← 타겟 비율 적용 여부
         random_state)                  ← 데이터 섞기
```

- 결 과 : X_train, y_train 훈련 데이터, X_test, y_test 테스트 데이터

PART II

SCIKIT-LEARN ML PRE-PROCESSING

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 전처리(Data Preprocessing)

❖ Garbage in, garbage out 으로 학습 시 데이터 중요성

데이터 정제 (CLEANING)	데이터 통합 (INTERGRATION)	데이터 균형 (BALANCING)	변환 (TRANSFORMATION)	축소 (REDUCTION)
* 결측치 처리 * 이상치 처리 * 노이즈 처리 * 모순값 처리 * 중복값 처리	* 여러 파일 통합	* **클래스**불균형처리 분류	*정규화(normalization) *집합화(Aggregation) *요약(summarization) *계층 생성	* 주요 특징 추출 * 불필요 특징 제거 * 데이터 축소

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 정제(Cleaning) -> 결측치 처리

❖ sklearn Lib : sklearn.imputer 모듈

sklearn.impute: Impute ¶

Transformers for missing value imputation

User guide: See the Imputation of missing values section for further details.

impute.SimpleImputer(*[, missing_values, ...])	Univariate imputer for completing missing values with simple strategies.
impute.IterativeImputer([estimator, ...])	Multivariate imputer that estimates each feature from all the others.
impute.MissingIndicator(*[, missing_values, ...])	Binary indicators for missing values.
impute.KNNImputer(*[, missing_values, ...])	Imputation for completing missing values using k-Nearest Neighbors.

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 균형(Balancing) -> 균형 데이터

❖ 클래스 간 데이터 불균형을 해결하는 방법

언더/다운샘플링(Under Sampling)

다수 클래스 데이터를 줄여서 샘플 비율 맞춤

- 다수 클래스 관측치 제거로 계산 시간 감소
- 데이터 클랜징으로 클래스 오버랩 감소
- **데이터 제거로 인한 정보 손실 발생**

  (중요 정보가 삭제될 시 치명적)

업샘플링(Up Sampling)

소수 클래스 샘플 증폭시켜 비율 맞춤

- 정보 손실이 없음
- 보통 **언더 샘플링보다 분류 정확도 높음**
- **과적합 가능성이 있음**
- 계산 시간이 오래 걸림
- 노이즈나 이상치 민감

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 균형(Balancing) -> 균형 데이터

❖ 클래스 간 데이터 불균형을 해결하는 방법 : **다운샘플링(Down Sampling)**

```
# 각 클래스의 샘플 인덱스 추출
i_class0 = np.where(target == 0)[0]
i_class1 = np.where(target == 1)[0]

# 각 클래스의 샘플 개수
n_class0 = len(i_class0)
n_class1 = len(i_class1)
```

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 균형(Balancing) -> 균형 데이터

❖ 클래스 간 데이터 불균형을 해결하는 방법 : 다운샘플링(Down Sampling)

```
# 클래스 0의 샘플만큼 클래스 1에서 중복을 허용하지 않고 랜덤하게 샘플 추출
i_class1_downsampled = np.random.choice(i_class1, size=n_class0, replace=False)

# 클래스 0의 타깃 벡터와 다운샘플링된 클래스 1의 타깃 벡터 합치기
np.hstack((target[i_class0], target[i_class1_downsampled]))

# 클래스 0의 특성 행렬과 다운샘플링된 클래스 1의 특성 행렬을 합칩니다.
np.vstack((features[i_class0,:], features[i_class1_downsampled,:]))[0:5]
```

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 균형(Balancing) -> 균형 데이터

❖ 클래스 간 데이터 불균형을 해결하는 방법 : **업샘플링(Up Sampling)**

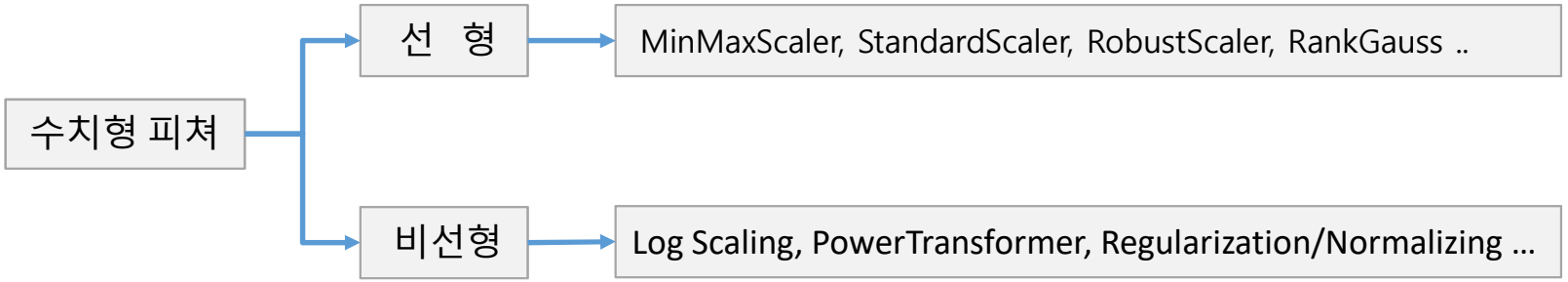
```
# 클래스 1의 샘플 개수만큼 클래스 0에서 중복을 허용하여 랜덤하게 샘플 선택
i_class0_upsampled = np.random.choice(i_class0, size=n_class1, replace=True)

# 클래스 0의 업샘플링된 타깃 벡터와 클래스 1의 타깃 벡터를 합침
np.concatenate(( target[i_class0_upsampled], target[i_class1]) )

# 클래스 0의 업샘플링된 특성 행렬과 클래스 1의 특성 행렬을 합침
np.vstack( (features[i_class0_upsampled,:], features[i_class1,:]) )[0:5]
```

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 정규화(Nomalization)

❖ 피쳐 스케일링(Feature Scaling)

- 서로 다른 Feature 값 범위를 **동일 범위로 맞추어 모든 Feature가 동일한 조건**
- 머신러닝 : 특정 Feature에 편중된 학습 해결
- 딥　러닝 : 최적화 빠르게 도달, 지역 극소값(Local minimum) 가능성 낮춤



SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 정규화(Nomalization)

❖ 피쳐 스케일링(Feature Scaling)

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 정규화(Nomalization)

❖ sklearn Lib : sklearn.preprocessing 모듈 - 선형

StandardScaler	- 본래 데이터 분포를 변형하여 표준 정규분포로 변환 (데이터 왜곡) - 이상치에 민감
MinMaxScaler, MaxAbsScaler	- 본래 데이터 분포 유지하며 모든 특성이 정확하게 0과 1사이에 위치 - 이미지 데이터 픽셀값 정규화 적용 - 이상치에 민감
RobustScaler	- **이상치 영향 최소화** - 중앙값(median)과 IQR(interquartile range) 사용
QuantileTransformer	- 기본 1000분위수로 균등분포 또는 정규분포로 0과 1사이 선형변환 - **이상치 영향 최소화**

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 정규화(Nomalization)

❖ sklearn Lib : sklearn.preprocessing 모듈 – 비선형

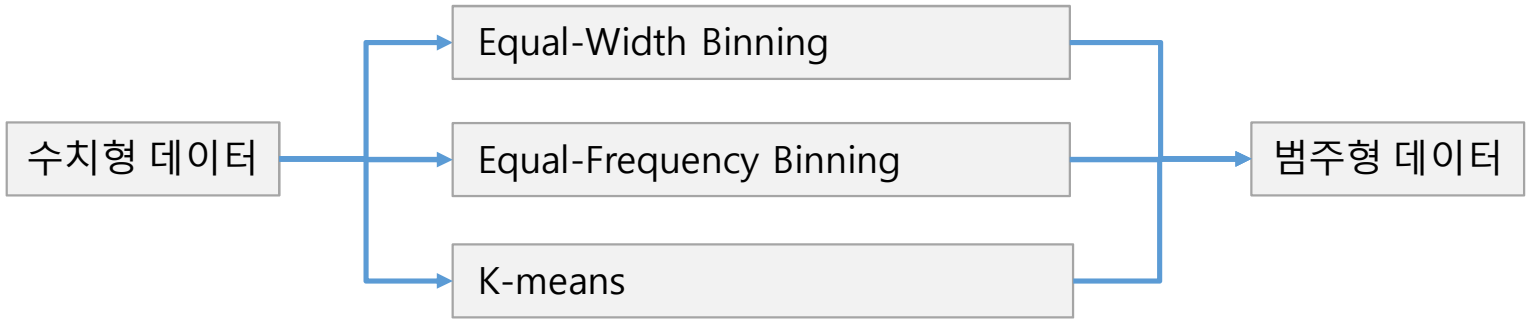
np.log10(), log(), log1p()	- 큰수의 범위를 압축하고 작은수의 범위 확장 - 두꺼운 꼬리분포(heavy-tailed distribution) 갖는 양수 데이터에 적용
PowerTransformer	- 로그변환 일반화 - 분산이 더이상 평균에 의존하지 않도록 분포 변경 * box-cox 알고리즘 : 데이터가 모두 양수 * Yeo-Johnson 알고리즘 : Box-Cox 일반화, 실수 전체
normalize	- **행 단위**로 정규화 진행 즉, 개별 데이터 크기 모두 같게 만들기 위한 변환 - overfitting을 방지기법으로 모델 구성 coefficient들이 학습 데이터에 overfitting되지 않도록 정규화 요소 더해 주는 것 ➔ Regularization(규제)

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 이산화(Discretization)/범주화(Binning)

❖ 이산화/범주화

- **숫자형 데이터를 일정 기준으로 나누어 범주형 데이터로 변환해** 주는 것
- 데이터의 단순화로 학습 프로세스 빨라지며, **과대적합 완화**
- 비선형 관계에서도 설명 가능 및 **이상치 발생 문제 완화, 결측치 문제 완화**



SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) ->이산화(Discretization)/범주화(Binning)

❖ sklearn Lib : sklearn.preprocessing 모듈 / pandas 함수

KBinsDiscretizer	- K-means알고리즘 기반 구간 나눔 - 각 구간별 One-Hot-Encoding 적용
pandas.cut()	- 등폭 이산화, 편향된 분포에 민감 - 각 구간별 One-Hot-Encoding 적용
pandas.qcut()	- 등빈도 이산화 - 각 구간별 One-Hot-Encoding 적용

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 인코딩(Encoding)

❖ 인코딩 (Encoding)

- 머신러닝은 문자열 데이터 속성 허용 불가, 숫자형 데이터 변환

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 인코딩(Encoding)

❖ sklearn Lib : sklearn.preprocessing 모듈 / pandas 함수

pandas.get_dummies()	- 범주형 피쳐의 레벨 개수만큼 가변수 생성 후 0과 1로 설정
OneHotEncoder	- 카테고리 개수 크기에 해당 카테고리만 1로 설정, 나머지 0 - 많은 카테고리의 경우 적합하지 않음 - 많은 카테로기의 경우 다른 인코딩 또는 카테고리 수를 줄여서 사용
OrdinalEncoder	- 순서가 중요한 범주형 피쳐 적용 - 0 ~ n_categories 범위 정수
LabelEncoder	- 타겟 라벨 피쳐를 정렬 후 정수로 변환 - n ~ n_categories 범위

SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 인코딩(Encoding)

❖ 사용자 수동 인코딩

value_counts(), map()	- 각 레벨의 **출현 횟수 혹은 출현 빈도로 범주형 변수를 대체** - 각 레벨의 출현 빈도와 타겟변수 관련성 있을 때 유효 - **동률 값 발생 주의!!**

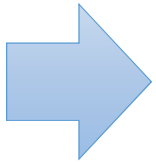
SCIKIT-LEARN ML PRE-PROCESSING

◆ 데이터 변환(Transformation) -> 인코딩(Encoding)

❖ 희소행렬 (Sparse Matrix)

행렬의 값이 대부분 0인 행렬 , 반대 표현으로 밀집행렬(dense matrix)

		column					
		0	1	2	3	4	5
row	0	15	0	0	22	0	-15
	1	0	11	3	0	0	0
	2	0	0	0	-6	0	0
	3	0	0	0	0	0	0
	4	91	0	0	0	0	0
	5	0	0	28	0	0	0



	row	col	value
a [0]	6	6	8
[1]	0	0	15
[2]	0	3	22
[3]	0	5	-15
[4]	1	1	11
[5]	1	2	3
[6]	2	3	-6
[7]	4	0	91
[8]	5	2	28

PART II

SCIKIT-LEARN ML
REGRESSION

REGRESSION WITH ML

◆ 데이터 분류

❖ 종류에 따른 분류

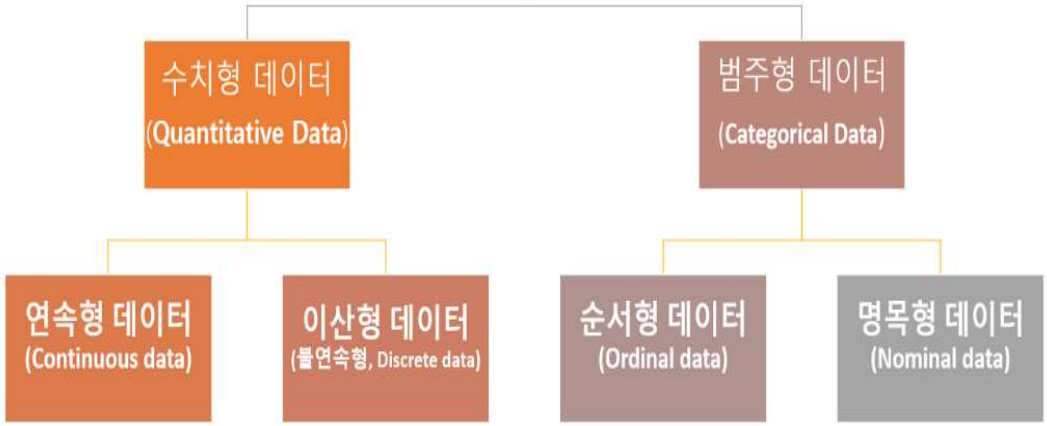
특성 따른 분류

개수 따른 분류

REGRESSION WITH ML

◆ 데이터 분류

❖ 특성



A, B, O, AB	범주형 - 명목형
만족도 ★ ★★☆☆	범주형 - 순서형
키 183.3, 몸무게 72.5	수치형 - 연속형
생산차량 수 5091대	수치형 - 이산형

REGRESSION WITH ML

◆ 데이터 분류

❖ 변수 개수

변수 (Variable Data)	- **연구 · 조사 · 관찰**하고 싶은 **대상** - 예)키, 몸무게, 혈액형, 매출액, 습도, 먼지 농도 등등....
단일변수 자료 (Univariate Data)	- 연구 · 조사 · 관찰하고 싶은 대상이 **1개로만 구성된 자료** - **일변량 자료**라고도 함
다중변수 자료 (Multivariate Data)	- 연구 · 조사 · 관찰하고 싶은 대상이 **2개 이상으로 구성된 자료** - **다변량 자료**라도도 함

REGRESSION WITH ML

◆ 데이터 분류

❖ 변수 역할

독립 변수 (Independent Variable)	- 연구, 조사에 **자극 / 원인이 되는 대상** - 연속형인 경우 , 범주형인 경우
종속 변수 (Dependent Variable)	- **독립변수에 따라** 결과가 **결정되는 변수** - **반응변수, 결과변수, 목표변수**

◆ 생선 무게 예측 - ② 데이터 수집

❖ 특성 데이터 : 농어 56마리의 길이, 무게

Fish.csv

Species	Weight	Length	Diagonal	Height	Width
Parkki	200	23	25.8	10.3458	3.6636
Parkki	273	25	28	11.088	4.144
Parkki	300	26	29	11.368	4.234
Perch	5.9	8.4	8.8	2.112	1.408
Perch	32	13.7	14.7	3.528	1.9992
Perch	40	15	16	3.824	2.432
Perch	51.5	16.2	17.2	4.5924	2.6316
Perch	70	17.4	18.5	4.588	2.9415
Perch	100	18	19.2	5.2224	3.3216
Perch	78	18.7	19.4	5.1992	3.1234
Perch	80	19	20.2	5.6358	3.0502
Perch	85	19.6	20.8	5.1376	3.0368
Perch	85	20	21	5.082	2.772
Perch	110	21	22.5	5.6925	3.555
Perch	115	21	22.5	5.9175	3.3075

◆ 생선 무게 예측 - ③ 데이터 추출

❖ 특성 데이터 : 길이와 무게

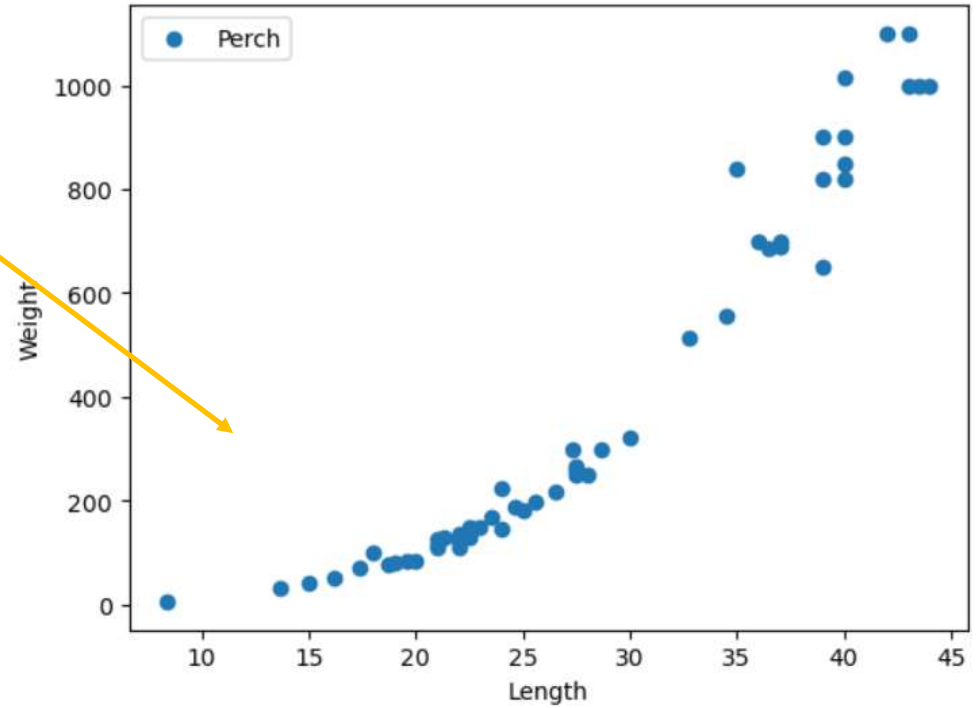
Species	Weight	Length	Diagonal	Height	Width
Parkki	200	23	25.8	10.3458	3.6636
Parkki	273	25	28	11.088	4.144
Parkki	300	26	29	11.368	4.234
Perch	5.9	8.4	8.8	2.112	1.408
Perch	32	13.7	14.7	3.528	1.9992
Perch	40	15	16	3.824	2.432
Perch	51.5	16.2	17.2	4.5924	2.6316
Perch	70	17.4	18.5	4.588	2.9415
Perch	100	18	19.2	5.2224	3.3216
Perch	78	18.7	19.4	5.1992	3.1234
Perch	80	19	20.2	5.6358	3.0502
Perch	85	19.6	20.8	5.1376	3.0368
Perch	85	20	21	5.082	2.772
Perch	110	21	22.5	5.6925	3.555
Perch	115	21	22.5	5.9175	3.3075

```
# 농어 길이, 무게 데이터 수집
perch_length = np.array(
    [8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0,
     21.0, 21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5,
     22.5, 22.7, 23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5,
     27.3, 27.5, 27.5, 27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0,
     36.5, 36.0, 37.0, 37.0, 39.0, 39.0, 39.0, 40.0, 40.0, 40.0,
     40.0, 42.0, 43.0, 43.0, 43.5, 44.0]
     )

perch_weight = np.array(
    [5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0,
     110.0, 115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0,
     130.0, 150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0,
     197.0, 218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0,
     514.0, 556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0,
     820.0, 850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0,
     1000.0, 1000.0]
     )
```

◆ 생선 무게 예측 - ④ 데이터 분석

❖ 특성 데이터의 관계 분석

길이와 무게 관계



◆ 생선 무게 예측 - ⑤ 데이터 가공

❖ 지도학습용 데이터셋 형태 ➔ 데이터/특성 + 라벨/타겟

▪ 특성 데이터

길이
254.
26.3
26.5
29.0
29.7
29.7
30.0
:
14.3
15.0

▪ 정답/라벨/타겟

무게
242.0
290.0
340.0
363.0
430.0
450.0
500.0
:
19.7
19.9

◆ 생선 무게 예측 - ⑤ 데이터 가공

❖ 지도학습용 Scikit-learn 라이브러리 데이터셋 형태

- 데이터 => **2차원**

```
# 특성 데이터 2차원 변환
perch_length=perch_length.reshape(-1, 1)

print(f'ndim : {perch_length.ndim}, shape : {perch_length.shape}')
```

- 라 벨 => 1차원

```
perch_weight = np.array(
    [5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0,
     110.0, 115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0,...
     1000.0, 1000.0]  )
```

REGRESSION WITH ML

◆ REGRESSION

- **지도학습의 학습 방법 중 하나**
- 입력 데이터에 대한 타겟 **값(수치)를 예측하는 학습 방법**
  - 예) 키에 따른 몸무게, 피자 도착 시간, 경계 성장률, 거리에 따른 도착시간 등등
- 데이터 구성 : 독립변수/특성/속성/피쳐+ 종속변수/타겟/라벨/정답/클래스

◆ 생선 무게 예측 - ① 목표 설정

❖ 목표 : 농어 무게에 따른 판매가격 결정하기 위한 정확한 농어 무게 예측 프로그램 구현

56마리 농어의 길이와 무게 데이터를 기반으로

농어 길이에 따른 무게 예측하기

◆ 생선 무게 예측 - ② 데이터 수집

❖ 특성 데이터 : 농어 56마리의 길이, 무게

Fish.csv

Species	Weight	Length	Diagonal	Height	Width
Parkki	200	23	25.8	10.3458	3.6636
Parkki	273	25	28	11.088	4.144
Parkki	300	26	29	11.368	4.234
Perch	5.9	8.4	8.8	2.112	1.408
Perch	32	13.7	14.7	3.528	1.9992
Perch	40	15	16	3.824	2.432
Perch	51.5	16.2	17.2	4.5924	2.6316
Perch	70	17.4	18.5	4.588	2.9415
Perch	100	18	19.2	5.2224	3.3216
Perch	78	18.7	19.4	5.1992	3.1234
Perch	80	19	20.2	5.6358	3.0502
Perch	85	19.6	20.8	5.1376	3.0368
Perch	85	20	21	5.082	2.772
Perch	110	21	22.5	5.6925	3.555
Perch	115	21	22.5	5.9175	3.3075

◆ 생선 무게 예측 - ③ 데이터 추출

❖ 특성 데이터 : 길이와 무게

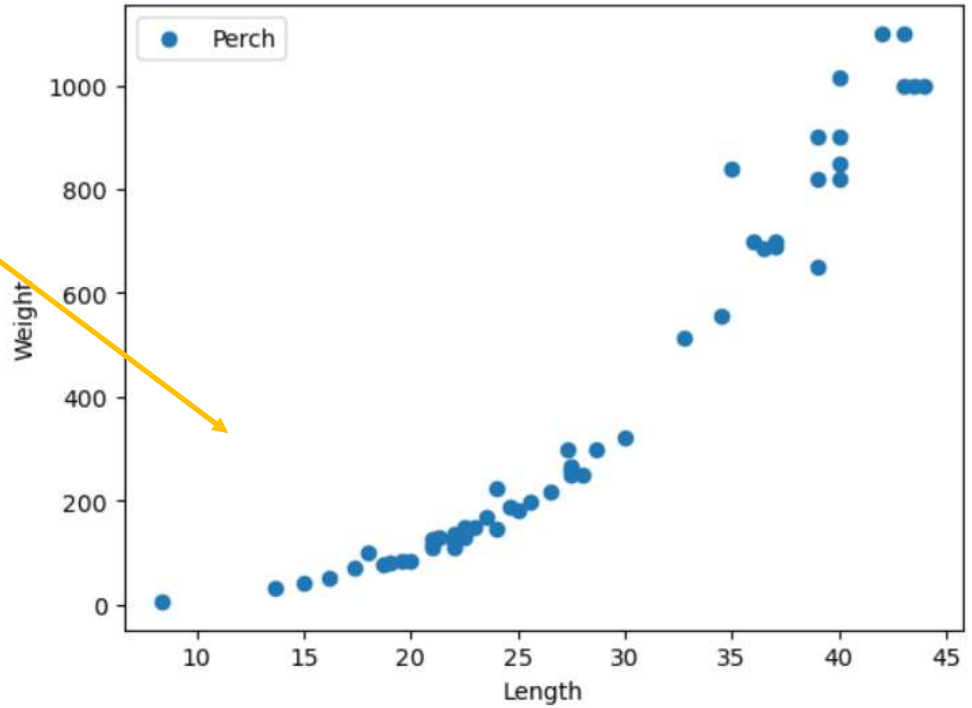
Species	Weight	Length	Diagonal	Height	Width
Parkki	200	23	25.8	10.3458	3.6636
Parkki	273	25	28	11.088	4.144
Parkki	300	26	29	11.368	4.234
Perch	5.9	8.4	8.8	2.112	1.408
Perch	32	13.7	14.7	3.528	1.9992
Perch	40	15	16	3.824	2.432
Perch	51.5	16.2	17.2	4.5924	2.6316
Perch	70	17.4	18.5	4.588	2.9415
Perch	100	18	19.2	5.2224	3.3216
Perch	78	18.7	19.4	5.1992	3.1234
Perch	80	19	20.2	5.6358	3.0502
Perch	85	19.6	20.8	5.1376	3.0368
Perch	85	20	21	5.082	2.772
Perch	110	21	22.5	5.6925	3.555
Perch	115	21	22.5	5.9175	3.3075

```
# 농어 길이, 무게 데이터 수집
perch_length = np.array(
    [8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0,
     21.0, 21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5,
     22.5, 22.7, 23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5,
     27.3, 27.5, 27.5, 27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0,
     36.5, 36.0, 37.0, 37.0, 39.0, 39.0, 39.0, 40.0, 40.0, 40.0,
     40.0, 42.0, 43.0, 43.0, 43.5, 44.0]
     )

perch_weight = np.array(
    [5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0,
     110.0, 115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0,
     130.0, 150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0,
     197.0, 218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0,
     514.0, 556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0,
     820.0, 850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0,
     1000.0, 1000.0]
     )
```

◆ 생선 무게 예측 - ④ 데이터 분석

❖ 특성 데이터의 관계 분석

길이와 무게 관계



◆ 생선 무게 예측 - ⑤ 데이터 가공

❖ 지도학습용 데이터셋 형태 ➔ 데이터/특성 + 라벨/타겟

▪ 특성 데이터

길이
254.
26.3
26.5
29.0
29.7
29.7
30.0
:
14.3
15.0

▪ 정답/라벨/타겟

무게
242.0
290.0
340.0
363.0
430.0
450.0
500.0
:
19.7
19.9

◆ 생선 무게 예측 - ⑤ 데이터 가공

❖ 지도학습용 Scikit-learn 라이브러리 데이터셋 형태

- 데이터 => **2차원**

```
# 특성 데이터 2차원 변환
perch_length=perch_length.reshape(-1, 1)

print(f'ndim : {perch_length.ndim}, shape : {perch_length.shape}')
```

- 라 벨 => 1차원

```
perch_weight = np.array(
    [5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0,
     110.0, 115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0,...
     1000.0, 1000.0]  )
```

◆ 생선 무게 예측 - ⑤ 데이터 가공

❖ 훈련용 / 테스트용 데이터셋 분리

```
from sklearn.model_selection import train_test_split

train_data, test_data, train_target, test_target = train_test_split(
                                                    perch_length,
                                                    perch_weight,
                                                    random_state=42 )
```

REGRESSION WITH ML

◆ 생선 무게 예측 - ⑥ 학습 방법 선택 : Scikit-learn 라이브러리

❖ 지도학습용 Scikit-learn 라이브러리 – 회귀 학습 방법

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

분류를 위한 **여러가지 학습 방법** 중
가장 성능이 좋은 방법 선택

모든 학습 방법을 적용하기에는 많은 시간 소요
많은 개발자들의 **경험들을 기반으로 추천되는 학습 방법 선택**

REGRESSION WITH ML

◆ Sklearn 라이브러리 - KNeighborsRegressor

❖ Sklearn Lib – 분류 모델 객체

```
from sklearn.neighbors import KNeighborsRegressor (
    n_neighbors=5,          ← 범위, 기본값
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,                    ← 거리 측정 방법 1: 맨해튼, 2:유클리디안
    metric='minkowski',     ← 거리 측정 방법
    metric_params=None,
    n_jobs=None )           ← 학습 시 사용할 CPU 코어 수, -1:모든 CPU
```

REGRESSION WITH ML

◆ Sklearn 라이브러리 - KNeighborsRegressor

❖ Sklearn Lib - 학습 메서드

regressor.fit(학습데이터, 타겟데이터)

- 학습데이터 : **2차원 형태**
- 타겟데이터 : **1차원 형태**

[학습 후 확인 속성]

- regressor.n_features_in_ : 특성 개수
- regressor.n_samples_fit_ : 학습 데이터 수

REGRESSION WITH ML

◆ Sklearn 라이브러리 - KNeighborsRegressor

❖ Sklearn Lib - 예측 메서드

예측결과 = **regressor.predict(특성데이터)**

- 특성데이터 : **2차원 형태**
- 예측결 과 : **타겟 값**

REGRESSION WITH ML

◆ Sklearn 라이브러리 – KNeighborsRegressor

❖ Sklearn Lib - 성능평가 메서드

성능결과 = **regressor.score(테스트데이터, 테스트타겟)**

- 테스트데이터 : **2차원 형태**
- 테스트타 겟 : **1차원 형태**
- 성 능결 과 : **타겟 값에 대한 적합한 정도**

[적합도 , 결정계수(coefficient of determination: R^2)]

- 예측값이 타겟값에 얼마나 적합한지 백분율로 나타낸 값
- 1- ((y_true - y_pred)** 2).sum() / ((y_true - y_true.mean()) ** 2).sum()
- 값의 범위 : 0.0 ~ **1.0**

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 알고리즘 원리

- **데이터 분포가 선형인 경우 해당 데이터를** 가장 만족하는 선형 관계식을 찾아서

  새로운 데이터에 대한 타겟값을 예측하는 알고리즘

❖ 쉽고 간단하면서 성능 좋은 머신러닝 알고리즘 중 하나

REGRESSION WITH ML

◆ 과대적합(Overfitting)

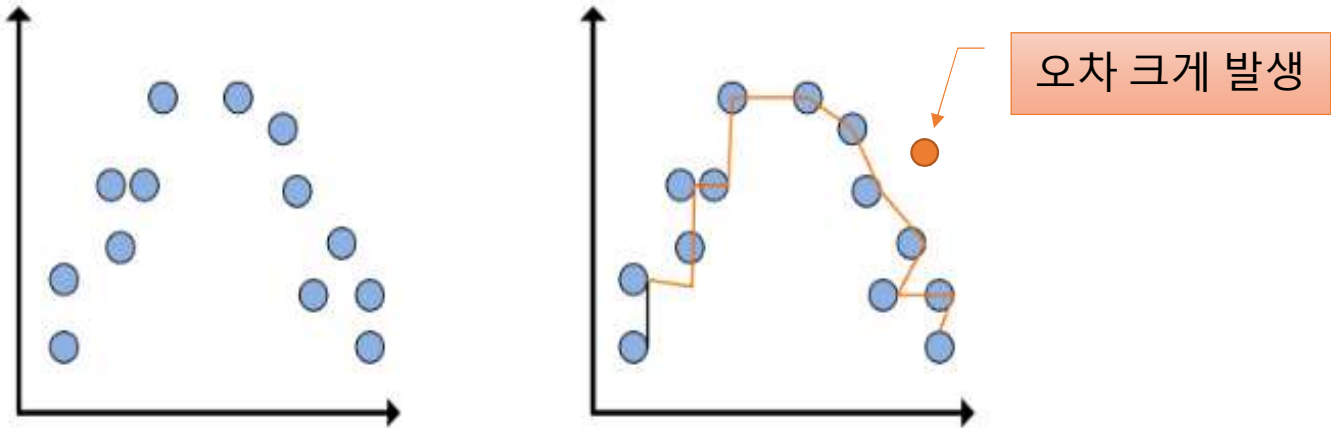
- **훈련 데이터 셋 특화된 즉 최적화된 모델**
- 새로운 데이터에 대한 **오차가 매우 커짐**

- **원인**
  - ➢ 데이터(특성)이 많아 **모델** 지나치게 **복잡**
  - ➢ 너무 **많은 학습**

REGRESSION WITH ML

◆ 과대적합(Overfitting)

- 훈　련 데이터 : 오차 없음
- 새로운 데이터 : 오차 크게 발생 확률 높음!



REGRESSION WITH ML

◆ 과소적합(Unterfitting)

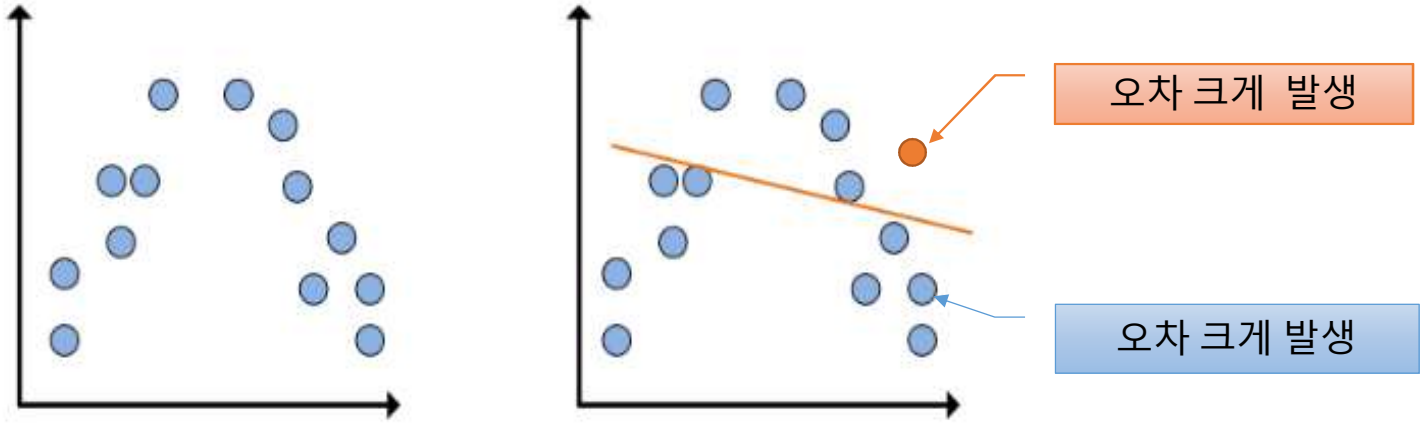
- **훈련 데이터 셋의 규칙/패턴 반영되지 않는 모델**
- 훈련 데이터와 새로운 데이터 대한 **오차가 매우 커짐**
- **훈련 데이터 오류가 줄어들지 않음**

- 원인
  - 데이터(특성)이 부족하여 **모델** 지나치게 **단순**
  - **학습** 횟수 **부족**

REGRESSION WITH ML

◆ 과소적합(Unterfitting)

- 훈　련 데이터 ➔ 오차 크게 발생
- 새로운 데이터 ➔ 오차 크게 발생 확률 높음!



REGRESSION WITH ML

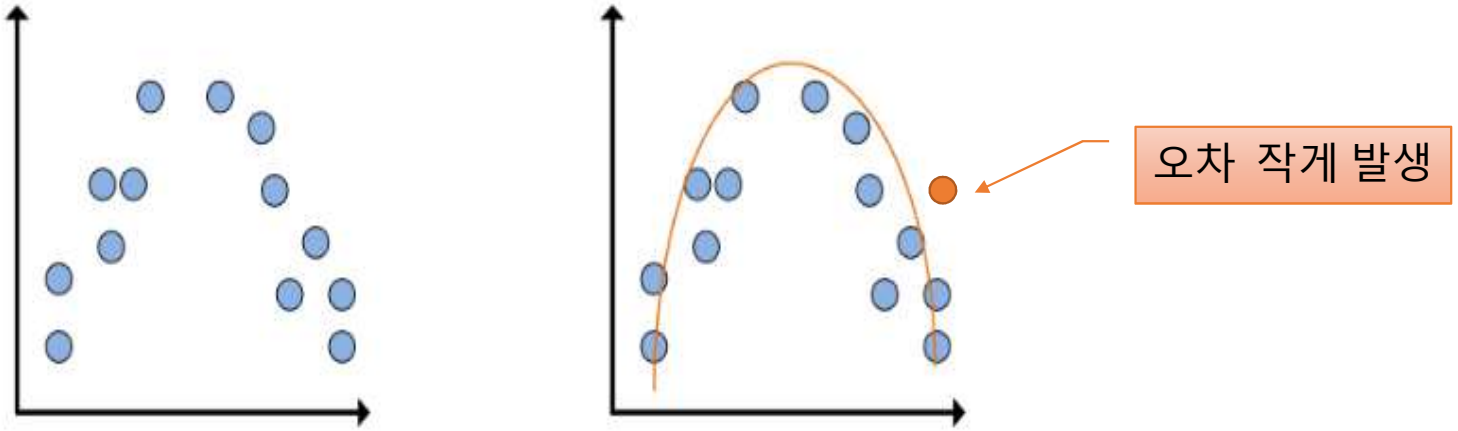
◆ 최적적합(Optimalfitting)

- **훈련 데이터 셋의 규칙/패턴이 일반화(Generalization) 된 모델**
- 훈련 데이터셋과 새로운 데이터 대한 오차 및 정확도 비슷
- **새로운 데이터에 대한 정확도 높음**

REGRESSION WITH ML

◆ 최적적합(Optimalfitting)

❖ 훈　련 데이터 : 오차 약간 발생
❖ 새로운 데이터 : 오차 약간 발생 확률 높음!



REGRESSION WITH ML

◆ 최적적합(Optimalfitting)

❖ **조건**

- 편중되지 않은 **다양성 갖춘 데이터**로 학습 진행
- **양질의 많은 데이터**
- **모델 복잡도 적정수준** 설정

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 데이터 분포가 선형인 경우 해당 데이터를 가장 만족하는 선형 관계 식을 찾아서 새로운 데이터에 대한 타겟값을 예측하는 알고리즘

❖ 쉽고 간단하면서 성능 좋은 머신러닝 알고리즘 중 하나

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

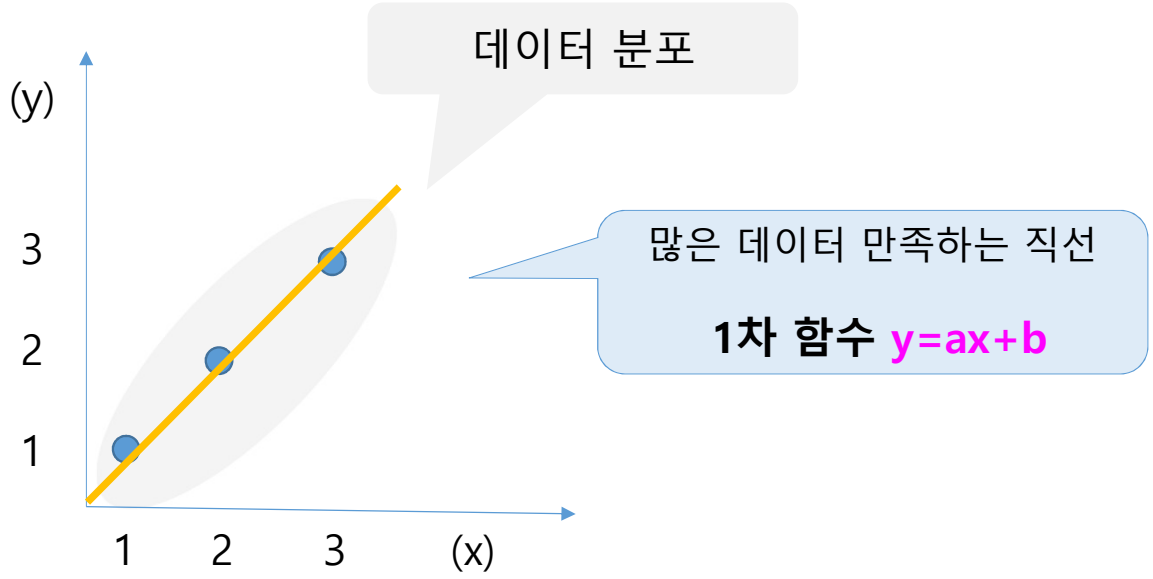
❖ 종류

- 선형회귀
- 다항회귀
- 다중회귀
- 로지스틱 회귀

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 데이터 분포 – 선형 데이터



REGRESSION WITH ML

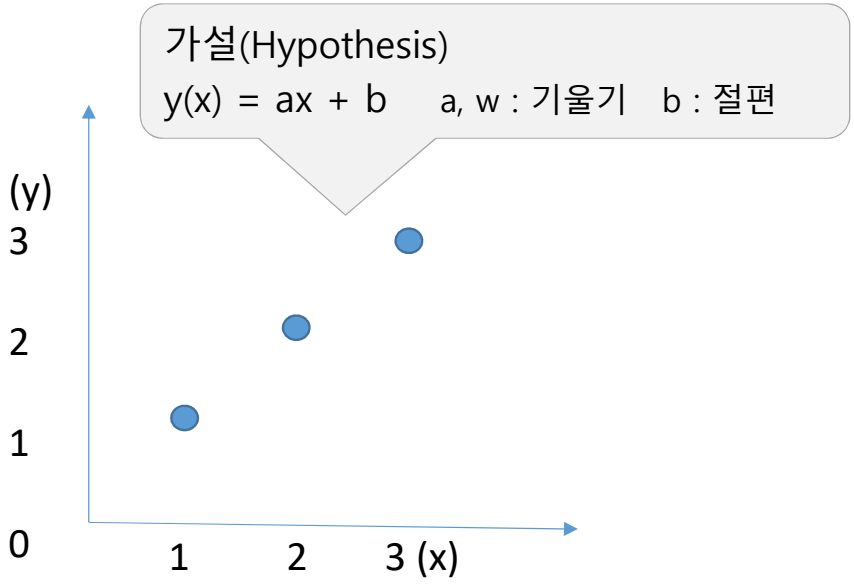
◆ 선형 회귀 : Linear Regression

❖ 데이터와 가설

x(hour)	y(score)
1	1
2	2
3	3

트레이닝 데이터

시각화



REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

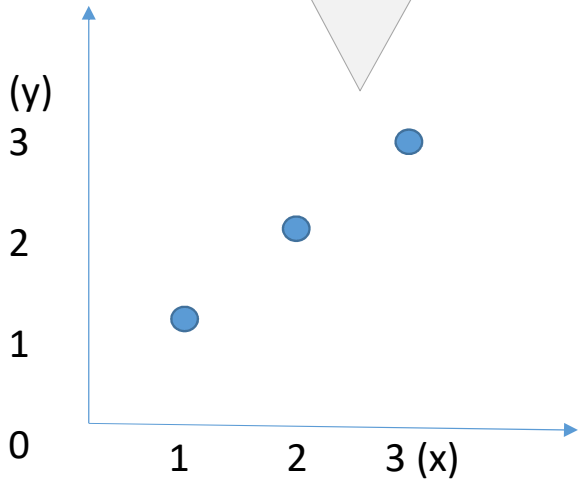
가설(Hypothesis) a= 2, b= -1
1= 2*1-1 => 1 0 0
2= 2*2 -1 => 3 -1 1
3= 2*3 -1 => 5 -2 4

❖ 데이터와 가설

x(hour)	y(score)
1	1
2	2
3	3

트레이닝 데이터

시각화



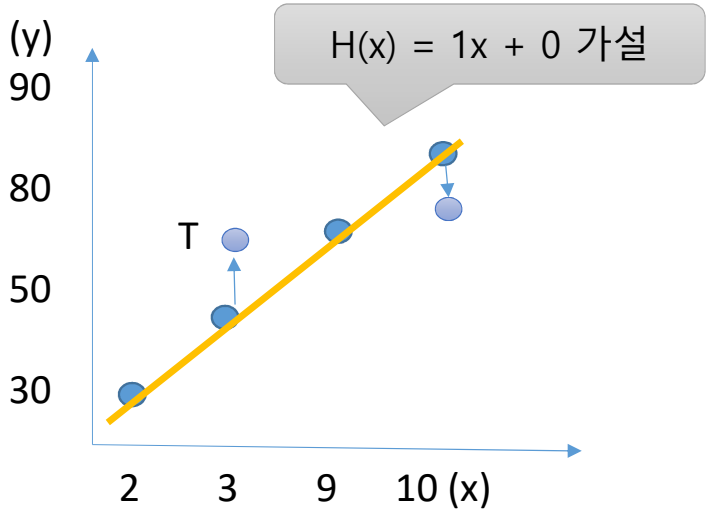
REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 비용/손실 함수 (Cost / Loss Function)

- Error : 실제 y값 - 예측 y값
- Square Error : 실제 y값 - 예측 y값 제곱

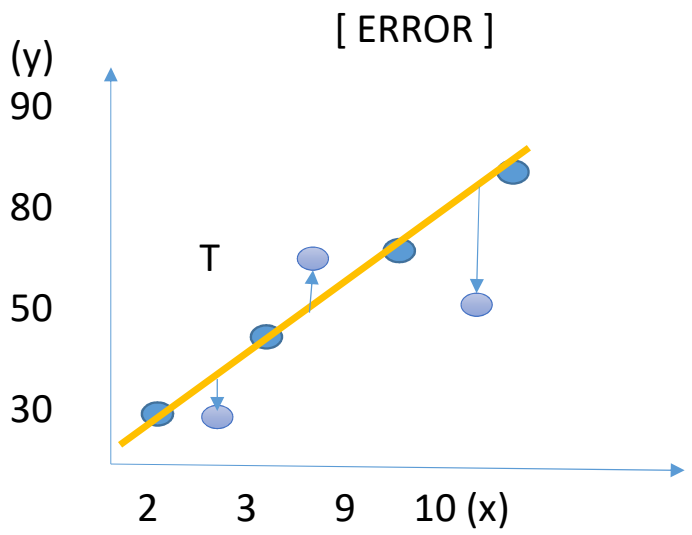
잔차 : T – y = T – (Wx+b)

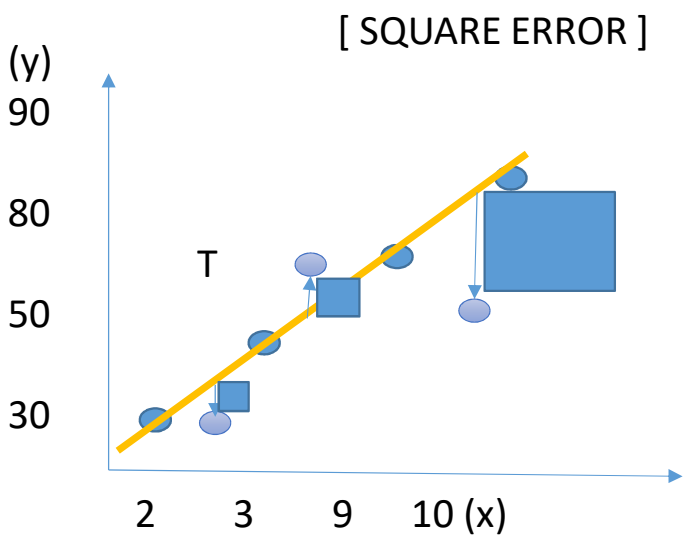


REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 비용/손실 함수 (Cost / Loss Function)





REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 비용/손실 함수 (Cost / Loss Function)

***평균제곱오차 함수 (MSE:mean squared error)**

- 대표적인 손실/비용 함수
- 예측값과 타겟값과의 차이를 제곱해서 모두 더 한 후 평균 취하는 함수

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

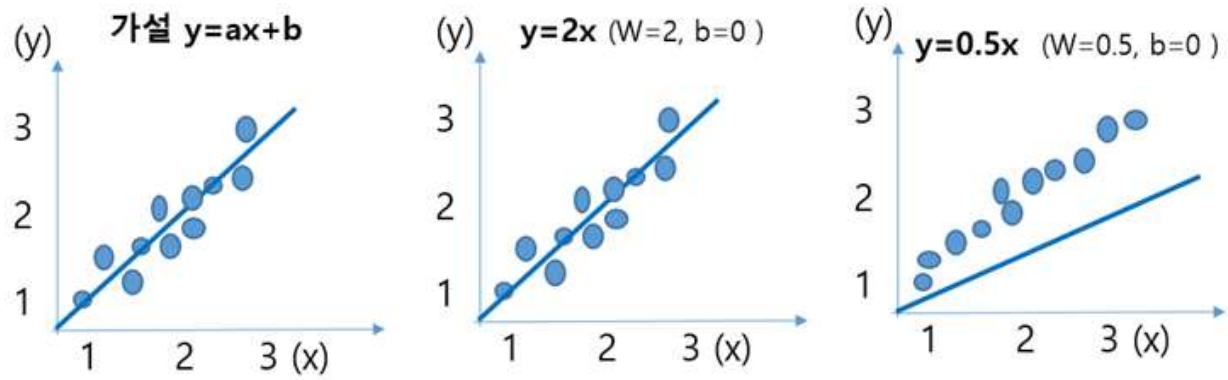
- yk : 예측값
- tk : 타겟 값

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 선형 데이터 설명 최적의 가설 찾기

[가설과 a, b]



➔ 기울기(가중치), 절편 최적값 설정 중요

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

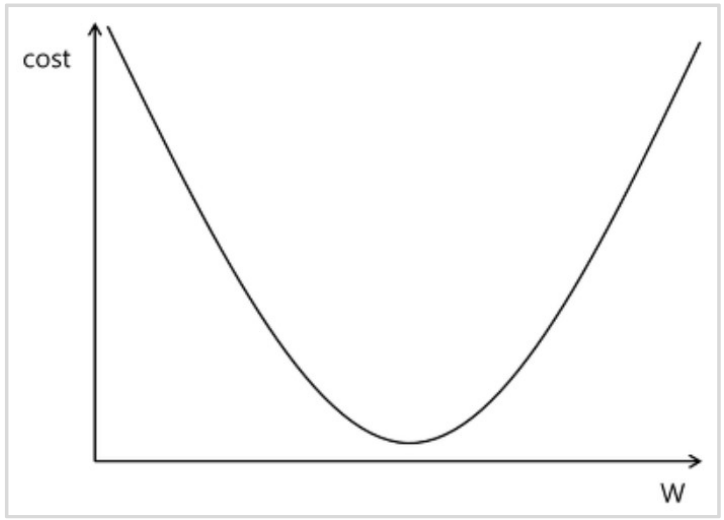
❖ 최적화 : Optimizer

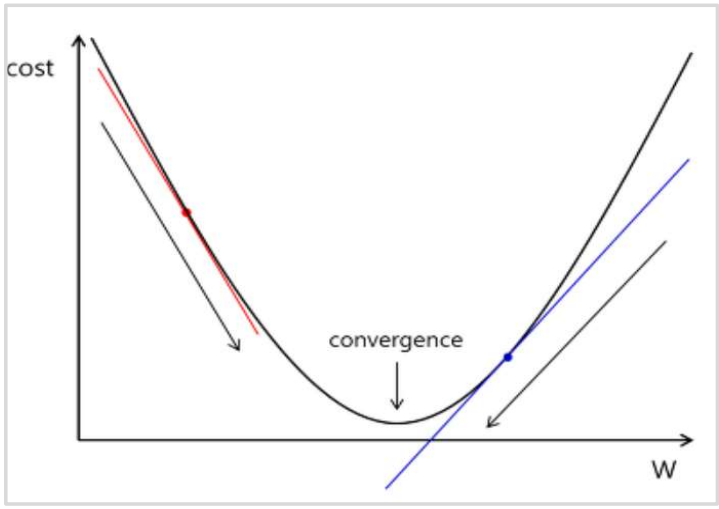
- **손실 함수 값이 최소화가 될 수 있도록 a, b 값 계속 업데이트** 필요
- 처음 랜덤한 값으로 초기화 후 적정한 값으로 계속 업데이트 진행
- 대표적인 최적화 기법
  - 경사하강법 (Gradient Descent)
  - 아담스최적화 기법
  - RMShop 기법 등등....

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 잔차와 기울기 관계





REGRESSION WITH ML

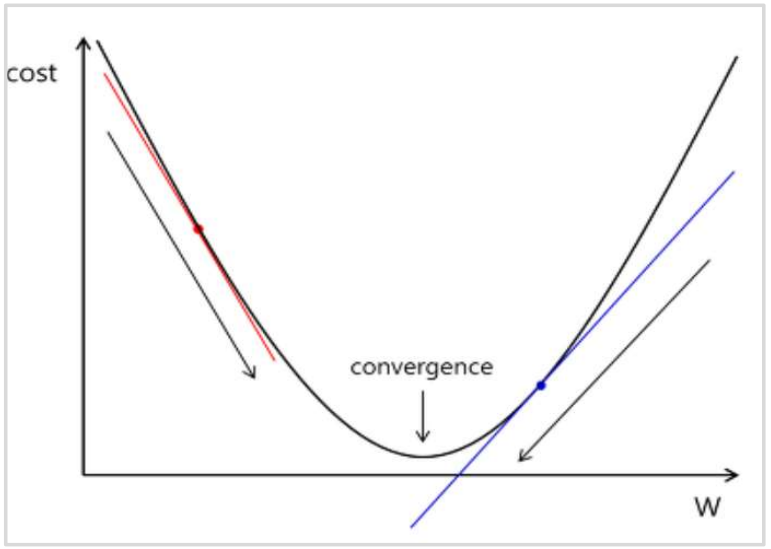
◆ 선형 회귀 : Linear Regression

❖ 잔차와 기울기 관계 ➔ 경사하강법(Gradient Descent)

최적의 선형회귀모델 직선을 긋고
그 직선의 MSE 즉 cost functin 최소로 만드는
직선 찾아야 함

학 습
경사하강법

cost를 최소로 하는 직선 구하는 과정

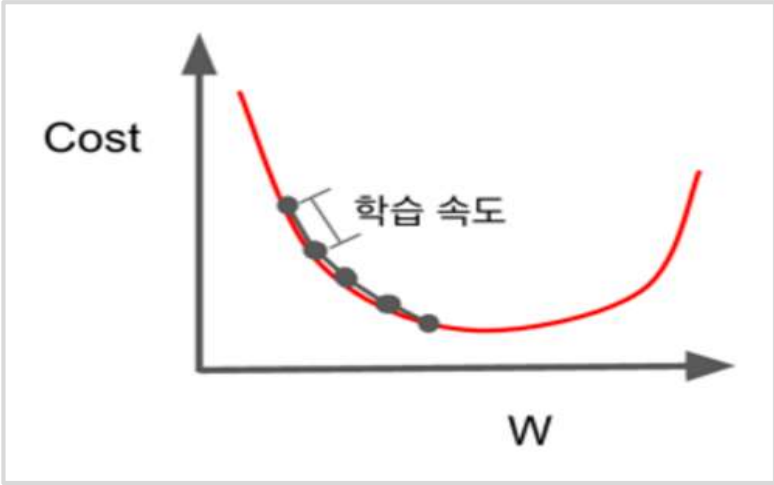


REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 학습속도 : Learning Rate

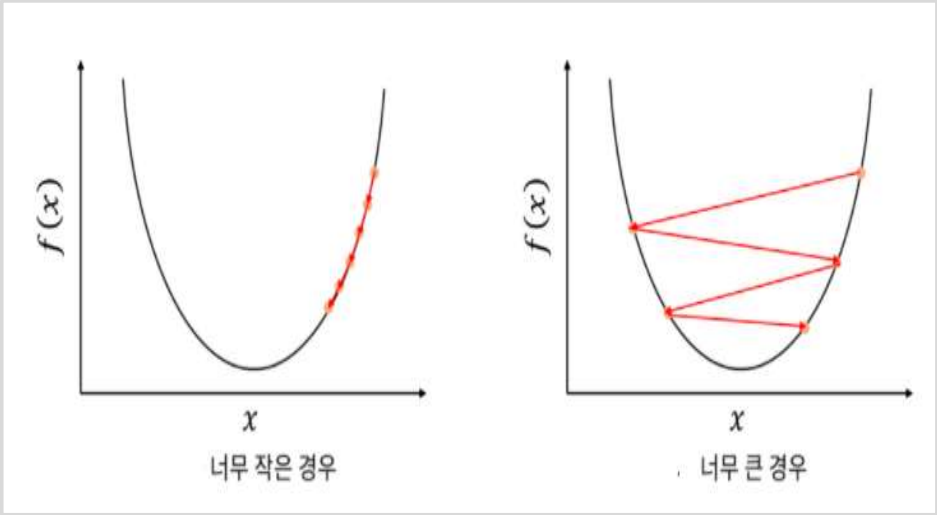
- 경사 하강법에서 **학습 단계별로 움직이는 학습 속도** 정의
- W값을 조정해 가면서 Cost 값이 최소가 되는 값을 찾기 위한 것



REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ 학습속도 : Learning Rate



- **오버슈팅(Over shooting)**
  학습속도가 큰 경우 발생
  최소값으로 내려가지 않고 반대편으로
  넘어가 무한대

- **스몰 러닝 레이트(Small Learning Rate)**
  학습속도가 매우 작은 경우 발생
  최소값 가기전 학습 종료

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ Sklearn Lib – 선형회귀 모델 객체

```
from sklearn.linear_model import LinearRegression
(  *,
   fit_intercept=True,          # 절편 사용 여부 설정
   normalize='deprecated',      # 정규화 진행 여부 설정
   copy_X=True,                 # 입력데이터 복사 사용
   n_jobs=None,                 # 구동 CPU 수, -1 : 모든 CPU
   positive=False
)
```

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ Sklearn Lib – 선형회귀 모델 객체

```
class LinearRegression
    coef_          : 기울기/가중치
    intercept_     : 절편값
```

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ sklearn Lib 학습 / 예측 메서드

regressor.fit(학습데이터, 라벨데이터)

- 학습데이터 : 2차원 형태
- 라벨데이터 : 1차원 형태

예측값 = **regressor.predict(학습데이터)**

REGRESSION WITH ML

◆ 선형 회귀 : Linear Regression

❖ sklearn Lib – 평가 메서드

regressor.score(학습데이터, 라벨데이터)

- 학습 데이터 : 2차원 형태
- 라벨 데이터 : 1차원 형태
- 반 환 값 : 타겟값에 적합한 정도를 백분율로 나타낸 값

1에 가까울 수록 성능 좋음

REGRESSION WITH ML

◆ 특성공학(Feature Engineering)

- 머신러닝 모델 위한 데이터 컬럼을 생성하거나 선택하는 작업
- 전저리 단계에서 진행
- 선택 기준
  - ➢ 결과에 **중요한 영향** 미치는 **Feature 선택**
  - ➢ **비슷한 Feature 제거**
  - ➢ **존재하지 않는 Feature는 조합** 하여 **생성**

REGRESSION WITH ML

◆ 특성공학(Feature Engineering)

❖ 필요한 이유?

- Feature 많은 경우 발생하는 문제점 해결 및 성능 개선 위해
  - ➢ 머신러닝 알고리즘 계산 시간 길어짐
  - ➢ 메모릭 공간 낭비
  - ➢ 학습 Feature에만 잘 동작하는 모델 즉 과대적합 모델이 됨

REGRESSION WITH ML

◆ 특성공학(Feature Engineering)

❖ 방법 – 차원축소

- **특징 추출**이라고도 함
- 과대적합을 피하며 **모델의 복잡도를 낮출** 수 있음

❖ 방법 – 특징 생성

- **특징 구축**이라고도 함
- 초기 주어진 데이터에서 모델 성능 높일 수 있는 **새로운 특성 생성**

REGRESSION WITH ML

◆ 특성공학(Feature Engineering)

❖ sklearn Lib

a, b,
a*a, b*b
a*b, 1

```
from sklearn.preprocessing import PolynomialFeatures
(
        degree=2,                   # 다항식의 차수
        interaction_only=False,     # 제곱값 제외 여부
        include_bias=True,          # 절편 포함
        order='C'                   # 행 우선 저장
)
```

REGRESSION WITH ML

◆ Polynomial REGRESSION

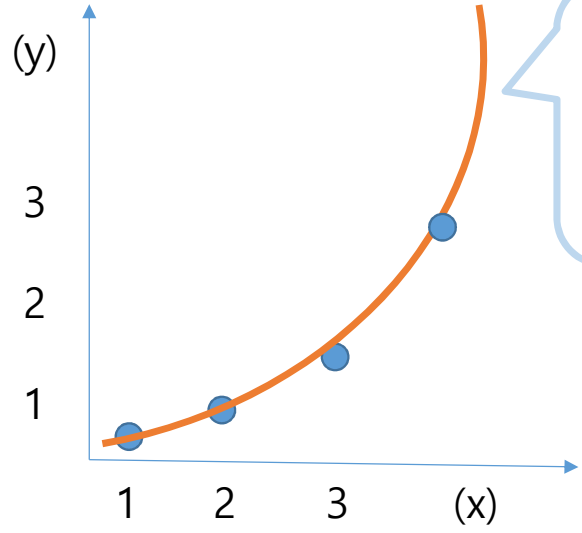
❖ 다항 회귀

- 1차 함수식으로 표현 할 수 없는 **복잡한 데이터 분포 적용**
- **비선형 데이터** 적용 선형 모델
- 극단적 높은 차수 모델 구현할 경우 **과적합 현상 발생**

REGRESSION WITH ML

◆ Polynomial REGRESSION

❖ **곡선 형태** 데이터 분포



많은 데이터 만족하는 곡선

2차 함수 $y=a_2x^2+a_1x+b$

N차 함수 $y=a_dx^d+..+a_2x^2+a_1x^1+b$